This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.

DISTRIBUTED CONTROL FOR COLLECTIVE BEHAVIOUR IN MICRO-UNMANNED AERIAL VEHICLES

by

FABIO RUINI

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing and Mathematics
Faculty of Science and Technology

June 2011

This page has been intentionally left blank

Distributed Control for Collective Behaviour in

Micro-unmanned Aerial Vehicles

by

Fabio Ruini

Abstract

The work presented herein focuses on the design of distributed autonomous controllers for collective behaviour of Micro-unmanned Aerial Vehicles (MAVs).

Two alternative approaches to this topic are introduced: one based upon the Evolutionary Robotics (ER) paradigm, the other one upon flocking principles. Three computer simulators have been developed in order to carry out the required experiments, all of them having their focus on the modelling of fixed-wing aircraft flight dynamics. The employment of fixed-wing aircraft rather than the omni-directional robots typically employed in collective robotics significantly increases the complexity of the challenges that an autonomous controller has to face. This is mostly due to the strict motion constraints associated with fixed-wing platforms, that require a high degree of accuracy by the controller.

Concerning the ER approach, the experimental setups elaborated have resulted in controllers that have been evolved in simulation with the following capabilities: (1) navigation across unknown environments, (2) obstacle avoidance, (3) tracking of a moving target, and (4) execution of cooperative and coordinated behaviours based on implicit communication strategies.

The design methodology based upon flocking principles has involved tests on computer simulations and subsequent experimentation on real-world robotic platforms. A customised implementation of Reynolds' flocking algorithm has been developed and successfully validated through flight tests performed with the *swinglet* MAV.

It has been notably demonstrated how the Evolutionary Robotics approach could be successfully extended to the domain of fixed-wing aerial robotics, which has never received a great deal of attention in the past. The investigations performed have also shown that complex and real physics-based computer simulators are not a compulsory requirement when approaching the domain of aerial robotics, as long as proper autopilot systems (taking care of the "reality gap" issue) are used on the real robots.

Contents

Abstract **5**

Table of Contents **6**

List of Figures **14**

List of Tables **16**

List of Algorithms **17**

1 Introduction **23**
1.1 Contribution to knowledge 26
1.2 Thesis outline 29

2 Evolutionary Robotics: Neural Networks and Evolutionary Computation Working Together **33**
2.1 Towards Evolutionary Robotics (ER) 33
2.1.1 Autonomous robotics 34
2.1.2 The classical approach to autonomous robotics: Shakey the robot and the problem of planning 35
2.1.3 Behaviour-based Robotics: embodiment and situatedness . . . 38
2.1.4 Evolutionary Robotics 41
2.1.5 A "cognitive" approach? The link with epigenetic robotics . . 49
2.2 The basics of control theory 51
2.2.1 Feedback (closed loop) control 51
2.2.2 Feedforward (open loop) control 57
2.3 Neural Networks (NNs) 57
2.3.1 McCulloch and Pitts' artificial neuron (TLU) 58
2.3.2 Hebbian learning 59
2.3.3 The properties of TLU networks: a classification example . . . 61
2.3.4 The limitations of perceptrons: dealing with non-linearly separable classes 63
2.3.5 Multi-layer perceptrons and the backpropagation algorithm . . 66
2.3.6 Networks with memory 67
2.3.7 Activation/transfer functions 69
2.4 Evolutionary Computation 70
2.4.1 The terminology 70
2.4.2 On natural/biological evolution 71
2.4.3 Evolutionary Computation: an overview 73
2.4.4 Evolution Strategies (ESs) 74
2.4.5 Genetic Algorithms (GAs) 76

2.4.6 Evolutionary Programming (EP) . . . 87
2.5 Incremental evolution . . . 89
2.5.1 Incremental evolution in autonomous robotics . . . 91

3 Aerial Robotics and Intelligent Control: History and Technologies 95
3.1 Unmanned flight: a brief history . . . 96
3.1.1 The drives for the UAVs in the military . . . 98
3.2 UAVs and MAVs . . . 102
3.2.1 Unmanned Aerial Vehicles (UAVs) . . . 102
3.2.2 Micro-unmanned Aerial Vehicles (MAVs) . . . 109
3.2.3 Applications . . . 113
3.3 Aerial robotics . . . 124
3.3.1 Fixed-wing MAVs: basic aerodynamics, design issues, characterisation and control techniques . . . 125
3.3.2 Characterisation and attitude determination . . . 133
3.3.3 Autopilots and autonomous control . . . 135

4 Distributed Control for Collective Behaviour in MAV Teams: Methodologies, Challenges, Ethical Considerations and Safety Issues 139
4.1 Intelligent autonomous controllers for collective aerial robots: the scientific literature . . . 140
4.2 Design methodologies . . . 148
4.2.1 Incremental geometric flight . . . 150
4.2.2 Flocking . . . 151
4.3 The main challenges . . . 153
4.3.1 Fixed-wing aircraft: motion constraints . . . 153
4.3.2 Obstacle avoidance . . . 155
4.3.3 Target tracking . . . 155
4.3.4 Collective behaviour, distributed control, and cooperation based upon implicit communication . . . 156
4.3.5 Computation time issues . . . 158
4.3.6 The reality gap . . . 159
4.4 Ethical considerations on the military employment of lethal autonomous robots . . . 160
4.4.1 The laws of war and the ethics of modern conflicts . . . 161
4.4.2 The experts' point of view on autonomous robotics . . . 163
4.5 Safety issues . . . 168
4.6 Plan for the experiments . . . 171
4.6.1 Defining "success" . . . 173
4.6.2 On the comparison with alternative design methodologies . . . 174

5 Simulation Experiments in Urban Layouts 177
5.1 Software simulator . . . 177
5.1.1 Common features of all simulation setups . . . 184
5.2 Neural network controllers . . . 187
5.2.1 Encoding of the input information . . . 188
5.3 Evolutionary algorithm . . . 189
5.4 Experiments . . . 190
5.4.1 Basic scenario (simulations A) . . . 190
5.4.2 Obstacle avoidance (simulations B) . . . 202
5.4.3 Moving target (simulations C) . . . 214

5.4.4 Implicit cooperation (simulations D) . . . 221
5.5 Conclusions . . . 233

6 Simulation Experiments in 3D Environments 237
6.1 Software simulator . . . 237
6.1.1 The evolutionary engine . . . 238
6.1.2 The viewer . . . 241
6.1.3 Computation issues . . . 243
6.1.4 Moving from 2D to 3D . . . 243
6.1.5 Common features of all simulation setups . . . 244
6.2 Neural network controllers . . . 246
6.2.1 Encoding of the input information . . . 247
6.3 Evolutionary algorithm . . . 252
6.4 Experiments . . . 253
6.4.1 Basic scenario (simulations A) . . . 253
6.4.2 Moving target (simulations B) . . . 268
6.4.3 Implicit cooperation (simulations C) . . . 273
6.5 Experiments on incremental evolution . . . 275
6.5.1 Basic results for A, B, and C setups . . . 277
6.5.2 The incremental approach . . . 281
6.6 Conclusions . . . 287
6.6.1 Multi-threading . . . 289
6.6.2 Incremental evolution . . . 289

7 Flocking Behaviour: Towards Experiments on Physical Robots 291
7.1 Robotics platform used: senseFly's swinglet . . . 292
7.2 Software simulator . . . 296
7.2.1 Initial formation . . . 299
7.2.2 Navigation algorithms . . . 301
7.2.3 Flocking algorithms . . . 304
7.3 Moving from simulations to reality: overcoming the reality gap . . . 311
7.3.1 Test field and coordinate systems . . . 313
7.3.2 Leader following behaviour through explicit communication (rendezvous) . . . 315
7.3.3 Flocking behaviour . . . 322
7.4 Conclusions . . . 328

8 Conclusions and Future Work 331
8.1 Conclusions . . . 331
8.2 Future work . . . 336

Bibliography 368

Appendices I

A Micro-unmanned Aerial Vehicles: a review I
A.1 AeroVironment Inc. . . . I
A.2 Lockheed Martin . . . VII
A.3 AAI Corporation . . . VIII
A.4 Israel Aerospace Industries (IAI) . . . XI
A.5 Insitu . . . XIII

A.6 Elbit Systems XV
A.7 AerialRobotics XVI
A.8 Miscellaneous XVIII
A.9 Classification XX

B Autopilot systems **XXIII**
B.1 SBP400/MNAV XXIII
B.2 Procerus Kestrel System XXV
B.3 MicroPilot MPxx28 XXVII
B.4 Cloud Cap Piccolo XXVIII
B.5 UNAV 35xx and PICOPILOT XXIX
B.6 FlexiPilot and EasyUAV XXXI

C P-ARTS (Plymouth Advanced Robot Training Suite) **XXXIII**
C.1 Hardware XXXIII
C.2 Software XXXIV

D Mathematical operations **XXXV**
D.1 Distances in three dimensions XXXV
D.2 Convert from degrees to radians and vice versa XXXV
D.3 Mean of circular quantities XXXVI
D.4 Convert from WGS84 to ECEF navigation coordinates XXXVI

List of Figures

2.1 Shakey the robot . . . 36
2.2 Graphical representation of the Behaviour-based approach . . . 39
2.3 Genghis the robot . . . 40
2.4 Sketch of two Braitenberg's vehicles . . . 42
2.5 Two real robots taking inspiration by Braitenberg's vehicles . . . 43
2.6 Graphical resume of how the Evolutionary Robotics approach works . 47
2.7 iCub robot . . . 51
2.8 Diagram of a typical feedback controller . . . 54
2.9 Graphical representation of a McCulloch-Pitts' neuron . . . 58
2.10 A neural network capable of achieving classical conditioning learning 60
2.11 A TLU characterised by two inputs (x_1, and x_2), a threshold function, and a y output . . . 62
2.12 The pattern space generated by a two-input TLU . . . 62
2.13 The pattern space for the XOR operator . . . 65
2.14 Example of a multi-layer NN architecture . . . 65
2.15 Example of an Elman network . . . 68
2.16 Graphical representation of the fitness-proportionate roulette wheel selection scheme . . . 78
2.17 Graphical examples of how mutation, inversion/variation, and single-point crossover genetic operators work on binary genomes . . . 81
2.18 Example of a fitness landscape for a two-gene genome . . . 84
2.19 Graphical representation of the fitness space . . . 86
2.20 Example of a genetic programming tree . . . 88

3.1 Classification of UAVs based on their role . . . 106
3.2 (a) NRL MITE2; (b) AeroVironment Black Widow . . . 111
3.3 Classification of the miniature UAVs reviewed in the appendix . . . 113
3.4 Typical action flow for a C4ISR mission . . . 114
3.5 Rotation axes for a typical fixed-wing aircraft . . . 125
3.6 Control surfaces for a typical fixed-wing aircraft . . . 126
3.7 Three types of wings commonly used for the design of MAVs . . . 127
3.8 Example of a multi-channel transmitter for RC aircraft . . . 132
3.9 An Artificial Horizon (AH) device . . . 134
3.10 Functional structure of an autopilot system . . . 137
3.11 A typical flight control system for MAVs . . . 138

4.1 Block diagram describing the control system implemented by How *et al.* . . . 145
4.2 Possible levels of decisional autonomy for MAVs involved in cooperative tasks . . . 147
4.3 Motion constraints for a fixed-wing aircraft . . . 154

5.1 Screenshot of the 2D simulator in a scenario which includes obstacles 179
5.2 Screenshots of the three tabs included in the QtTabWidget element of the application main window . . . 181
5.3 Example topology for a typical NN controller . . . 187
5.4 Discrete encodings of the MAV-target angle . . . 191
5.5 Simulation A2: average and best fitness . . . 195
5.6 Simulation A2: percentage of successful tests . . . 196
5.7 Simulation A2: average and minimum distance . . . 197
5.8 Simulation A2: amount of energy left . . . 198
5.9 Simulation A2: condition of the MAVs at the end of the "average test" 199
5.10 Simulations A: comparison for the average fitness . . . 200
5.11 Simulations A: comparison for the best fitness . . . 201
5.12 Simulations A: comparison for the success rate . . . 201
5.13 Overview of the obstacles distributed along the environment . . . 203
5.14 The four different configurations of ultra-sonic sensors tested . . . 205
5.15 Graphical representation of the NN controller used in simulations B2, B3, and B4 . . . 207
5.16 Simulations B: comparison for the average fitness . . . 209
5.17 Simulations B: comparison for the best fitness . . . 210
5.18 Simulations B: comparison for the success rate . . . 210
5.19 Simulation B4: condition of the MAVs at the end of the "average test" 212
5.20 The 2D simulated environment used by Zetule . . . 213
5.21 Zetule's experiment: average and best fitness . . . 213
5.22 The options available to the target for escaping the approaching MAV 215
5.23 Simulations C: comparison for the average fitness . . . 217
5.24 Simulations C: comparison for the best fitness . . . 217
5.25 Simulations C: comparison for the success rate . . . 218
5.26 Simulations B and C: comparison for the average and best fitness . . 220
5.27 Simulations B and C: comparison for the success rate . . . 220
5.28 Graphical representation of the NN controller used in Simulations D . 222
5.29 Simulation D1: flight paths followed by a team of MAVs cooperatively approaching the target . . . 225
5.30 Simulation D1: average and best fitness . . . 226
5.31 Simulation D1: percentage of tests concluded either successfully, with the approaching, or with the damaging of the target . . . 227
5.32 Simulation D1: condition of the MAVs at the end of the "average test" 228
5.33 Simulation D2: average and best fitness . . . 229
5.34 Simulation D2: percentage of tests concluded either successfully, with the approaching, or with the damaging of the target . . . 229
5.35 Simulation D2: condition of the MAVs at the end of the "average test" 230

6.1 Screenshot of the 3D simulator . . . 242
6.2 The simulation reference environment . . . 244
6.3 Some of the most representative NN topologies tested . . . 256
6.4 Simulation A5: flight paths followed by four individual MAVs sharing the same evolved controller . . . 258
6.5 Simulation A5: average and best fitness . . . 259
6.6 Simulation A5: percentage of successful tests . . . 259
6.7 Simulation A5: amount of energy left . . . 260

6.8 Bar plot displaying the maximum success rate obtained by the best individuals evolved with the various controller architectures deprived of memory components . . . 261
6.9 Simulations A: comparison for the average success rate in function of the sets of rotations available to the aircraft . . . 262
6.10 Simulations A: comparison for the maximum success rate in function of the sets of rotations available to the aircraft . . . 262
6.11 Simulations A: comparison for the average success rate in function of the kind of encoding used for the input information . . . 263
6.12 Simulations A: comparison for the maximum success rate in function of the type of encoding used for the input information . . . 263
6.13 Simulations A: comparison for the average success rate in function of the presence/absence of a hidden layer . . . 264
6.14 Simulations A: comparison for the maximum success rate in function of the presence/absence of a hidden layer . . . 265
6.15 Comparison between single and multi-threading in terms of execution speed on the Apple Mac Pro machine . . . 267
6.16 Comparison between single and multi-threading in terms of execution speed on the Apple Xserve machine . . . 267
6.17 Simulation B5: comparison for the success rate . . . 271
6.18 Simulation B11: comparison for the success rate . . . 271
6.19 Graphical representation of the NN controllers (a) C5; (b) C11 . . . 274
6.20 Simulation C2: flight paths followed by three individual MAVs sharing the same controller . . . 276
6.21 Simulation C5: flight paths followed by four individual MAVs sharing the same controller . . . 277
6.22 Simulation C5: percentage of successful and half-successful tests for the average and the best MAVs . . . 278

7.1 senseFly's *swinglet* MAV . . . 293
7.2 (a) the dsPic33 micro-controller upon which the autopilot has been built; (b) overall view of all the equipment hosted inside the payload bay . . . 295
7.3 Screenshot of the *e-mo-tion* main interface . . . 295
7.4 Screenshot of the flocking software simulator . . . 297
7.5 The simulation reference environment . . . 297
7.6 Different initial MAV teams formations . . . 299
7.7 Flight path followed by a single MAV taking off from the ground . . . 301
7.8 Flight paths followed by a team of four MAVs attracted to the centre of the reference environment (2D view) . . . 302
7.9 The three flocking rules elaborated by Reynolds . . . 308
7.10 Satellite image of the test field . . . 313
7.11 ECEF reference frame . . . 314
7.12 Rendezvous behaviour: flight paths followed in simulation by two MAVs 317
7.13 Rendezvous behaviour: distance in simulation between the leader and the follower . . . 318
7.14 Rendezvous behaviour: standard deviation in simulation for the MAVs' heading . . . 319
7.15 Rendezvous behaviour: flight paths followed in reality by four MAVs 321

7.16 Flocking behaviour: flight paths followed in simulation by a flock of four MAVs . . . 324
7.17 Flocking behaviour: area covered in simulation by a flock of four MAVs 325
7.18 Flocking behaviour: average and minimum distance in simulation between members of the flock . . . 326
7.19 Flocking behaviour: standard deviation in simulation for the MAVs' heading . . . 327
7.20 Inter-robot communication probability in function of the distance between the robots . . . 328
7.21 Flocking behaviour: flight paths followed in reality by a flock of nine MAVs . . . 329
7.22 Graphical representations of the impact of communication range and turn rate on heading standard deviation and intra-flock distance . . . 329

A.1 (a) AV Hornet; (b) AV Wasp Block I; (c) AV Wasp Block II; (d) AV Wasp Block III / BATMAV . . . III
A.2 (a) AV Dragon Eye; (b) AV Raven RQ-11A . . . IV
A.3 (a) AV Pointer FQM-151A; (b) AV Puma AE (All Environment) . . . VI
A.4 (a) AV Switchblade; (b) AV Nano Hummingbird; (c) AV HawkEye . . VII
A.5 Lockheed Martin Desert Hawk MAV . . . VIII
A.6 AAI Orbiter Mini UAS . . . IX
A.7 One System Ground Control Station (OSGCS) . . . X
A.8 (a) AAI-Aerosonde Mark 4.7; (b) details of the combined launch/recovery system used by the Mark 4.7 . . . XI
A.9 IAI Mosquito Micro-UAV . . . XII
A.10 IAI Mini UAS BirdEye 650 . . . XII
A.11 IAI Mini Panther . . . XIII
A.12 (a) Insitu ScanEagle; (b) the SkyHook device used for the recovery . XIV
A.13 Insitu NightEagle . . . XV
A.14 Elbit Systems Skylark I-LE . . . XVI
A.15 (a) AerialRobotics EasyUAV (modified EasyStar); (b) AerialRobotics Pteryx . . . XVII
A.16 Airborne platform . . . XIX
A.17 One of the research platforms developed by Wu and colleagues XIX
A.18 (a) Parrot AR.Drone; (b) screenshot of the AR.FlyingAce application XX

B.1 (a) Crossbow SPB400 Stargate Gateway; (b) Crossbow MNAV 100CA Inertial Measurement Unit . . . XXIV
B.2 Procerus Kestrel Autopilot System . . . XXV
B.3 Procerus Unicorn MAV . . . XXVI
B.4 MicroPilot MP2028^g . . . XXVII
B.5 (a) Cloud Cap Piccolo SL; (b) Cloud Cap Piccolo II . . . XXVIII
B.6 (a) UNAV 3550 sUAS autopilot; (b) UNAV PICOPILOT-N XXX
B.7 Two FlexiPilot autopilot systems stacked on top of each other XXXI

List of Tables

2.1 Activation/truth table for a two-input TLU implementing a logic AND 63
2.2 Truth table for the XOR function 64
2.3 The subfields of Evolutionary Computation and the corresponding level in the evolution hierarchy they model 73

3.1 USAF UAVs tier classification system 104
3.2 USMC UAVs tier classification system 105

5.1 Discretised encoding of the MAV-target distance 191
5.2 Summary of the main characteristics of simulations A 193
5.3 Simulations A: initial deployment of MAVs and target 193
5.4 Simulations A: resume of the main results 202
5.5 Simulations B: (X, Y) coordinates of the obstacles present in the simulated environment 204
5.6 Simulations B: (X, Y) coordinates of the enclosed area 204
5.7 Simulations B: values returned by the simulated ultra-sonic sensors . 206
5.8 Simulations B: initial deployment of MAVs and target 208
5.9 Simulations B: resume of the main results 208
5.10 Simulations B: condition of the MAVs at the end of the "average test" 211
5.11 Simulations C: resume of the main results 216
5.12 Simulations D: condition of the MAVs at the end of the "average test" 230
5.13 Simulations D: resume of the main results 231
5.14 Comparison between simulations D1 and E 233

6.1 Structure of the text file in which the flight paths followed by the MAVs during a test are memorised 242
6.2 3D simulations: initial deployment of MAVs and target 245
6.3 Discretised encoding of the MAV-target horizontal angle 248
6.4 Discretised encoding of the MAV-target vertical angle 249
6.5 Discretised encoding of the MAV bank angle 250
6.6 Discretised encoding of the MAV-target distance 251
6.7 Neural network controller architectures tested 255
6.8 Simulations A: resume of the main results 257
6.9 Simulation A9: comparison between single and multi-threading in terms of execution times 266
6.10 Simulations B: possible movement destinations for the target 269
6.11 Simulation B5: resume of the main results 272
6.12 Simulation B11: resume of the main results 272
6.13 Simulations C: resume of the main results 276
6.14 Simulations A: resume of the new main results 279

6.15 Simulations B (non-incremental setup, $T_s = \frac{M_s}{2}$): resume of the new main results 279
6.16 Simulations B (non-incremental setup, $T_s = \frac{M_s}{3}$): resume of the new main results 280
6.17 Simulations C (non-incremental setup): resume of the new main results 281
6.18 Simulations B (incremental setup from A to B, $T_s = \frac{M_s}{2}$): resume of the main results 282
6.19 Simulations B (incremental setup from A to B, $T_s = \frac{M_s}{3}$): resume of the main results 283
6.20 Simulations C (incremental setup from A to C): resume of the main results 284
6.21 Comparison between incremental (A to B) and non-incremental (B) evolution ($T_s = \frac{M_s}{2}$) 285
6.22 Comparison between incremental (A to B) and non-incremental (B) evolution ($T_s = \frac{M_s}{3}$) 286
6.23 Comparison between incremental (A to C) and non-incremental (C) evolution 286

7.1 Coordinates of the waypoints used in experiments with real robots . . 320
7.2 Structure of the data messages broadcasted by the leader MAV . . . 320

A.1 Classification of the miniature UAVs reviewed in the appendix XXI

List of Algorithms

1 Error backpropagation algorithm for a network consisting of a single hidden layer 67
2 Basic functioning of a GA 83
3 Flocking simulator: V-formation MAVs deployment (3D) 300
4 Flocking simulator: random MAVs deployment (3D) 300
5 Flocking simulator: speed adjustment for the i-th non-leader MAV . 305
6 Flocking simulator: heading alignment for the i-th MAV 306
7 Parker's pseudocode for flocking: cohesion rule for boid i 310
8 Parker's pseudocode for flocking: separation rule for boid i 310
9 Parker's pseudocode for flocking: assembling the rules together 310
10 Parker's-inspired pseudocode for flocking: assembling the rules together for boid i 311
11 Flocking simulator: speed adjustment for rendezvous behaviour (i-th follower) 316
12 Flocking simulator: navigation and flocking algorithm for boid i . . . 323

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Prof. Angelo Cangelosi, for the scientific and human support he provided me since I started my Ph.D. programme. His career is taking off, and I am confident that one day I will be proudly able to say, "I was there".

Then several other dedications are necessary.

To Prof. Dimitar Kazakov for the extremely detailed feedback he has given me about this thesis, that contributed enormously to make it a better piece of work.

To Dr. Leonid Perlovsky and the European Office of Aerospace Research and Development for the support provided at various level.

To Prof. Roberto Serra, and Prof. Domenico Parisi, who ignited the fire of science in me.

To Dr. Davide Marocco and Dr. Tony Morse, for the countless insightful philosophical discussions we have had together.

To Dr. Ghassan Abdalla, Dr. Paul Baxter, Prof. Tony Belpaeme, Dr. Guido Bugmann, Dr. Eduardo Coutinho, Dr. Phil Culverhouse, Joachim De Greeff, Frederic Delaunay, Prof. Jose' Fernando Fontanari, Christopher Ford, Dr. Peter Gibbons, Chris Larcombe, Magdalena Leschtanska, Dr. Zoran Macura, Martin Peniak, Salomon Ramirez-Contla, Robin Read, Marek Rucinski, Francesca Stramandinoli, Dr. Liz Stuart, Dr. Vadim Tikhanoff, Dr. Fan Wo, Dr. Joerg Wolf, and Dr. Rachel Wood. Working with you has simply been a great experience.

To Prof. Dario Floreano, Dr. Sabine Hauert, Dr. Severin Leven, and Dr. Jean-Cristophe Zufferey, who made my stay at the EPFL in Lausanne a very enriching experience, both from a professional and a human point of view.

To Lucy Cheetham, Elena Dell'Aquila, Sue Kendall, Julie Taylor, and Carole Watson for the help and support provided at various stages during my Ph.D. course.

To Simon Oliver, who was my competitor when I applied for the Ph.D. studentship that had in turn led to this work. Despite the fact that this event could have turned into a potential source of friction he never took it personally. Rather, in the following years he has always been the first to advise me about interesting papers to read, potential new research directions to undertake, and so on. An evil disease brought him away too soon, but may his example remain with me forever.

To Edward Baczynski, Paul Baxter, Barry Bentley, and Christopher Boyd-Swann, who proofread various parts of my thesis and provided me with rich insights about its contents. In wishing them the best of luck for their future careers, I must stress how any mistake should have remained in this work must be considered my fault only.

To the iCub robot and to all the people loudly whining and complaining while struggling to make it work, who kept reminding me how lovely it is working with computer simulations rather than real robots. To the Swinglet MAV platform I carried out experiments with, which reinforced in me that feeling.

To all of those who do not want to believe poker is a game based on skill rather than on mere luck. Your belief has allowed me to keep a well above average lifestyle over the last few years, as well as avoid the typical "Ph.D. student lack-of-money induced depression syndrome". Thank you.

To my family. Close or far it might be, it never lets me down.

To all the persons who have had to deal with my swinging mood during the writing of this thesis. I swear that hundreds of hours spent analysing data, writing, deleting, and re-writing chapters could even impact on Dalai Lama's mindset.

To x (whichever your name will be). You are not born yet and you are already one the most important things in my life. That's a good start, isn't it?

Update: you are born. And now you have a lovely name too: Samuel John. And yes, you have made this wait worthwhile!

Author's declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered to any other University award without prior agreement of the Graduate Committee.

This work has been carried out by Fabio Ruini, during his Ph.D. course in evolutionary robotics and distributed control for MAVs, under the supervision of Professor Angelo Cangelosi.

The research described in this thesis has been supported by the Air Force Office of Scientific Research, Air Force Office Material Command, USAF, through the European Office of Aerospace Research & Development (EOARD), under grant number FA8655-07-1-3075. Support has also been provided by euCognition (network action 097-3) and by the University of Plymouth.

Publications

Hauert, S., Leven, S., Varga, M., Ruini, F., Cangelosi, A., Zufferey, J.-C., and Floreano, D. (2011), Reynolds Flocking in Reality with Fixed-Wing Robots: Communication Range vs. Flight Dynamics. In *Proceedings of IROS 2011, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5015-5020.

Ruini, F., and Cangelosi, A. (2010), An Incremental Approach to the Evolutionary Design of Autonomous Controllers for Micro-unmanned Aerial Vehicles. In *Proceedings of TAROS 2010, the 11th Conference Towards Autonomous Robotic Systems*, pp. 239-246.

Ruini, F., and Cangelosi, A. (2010), An Evolutionary Robotics 3D Model for Autonomous MAVs Navigation, Target Tracking and Group Coordination. In *Proceedings of IJCNN 2010, the International Joint Conference on Neural Networks*, pp. 760-767.

Ruini, F., and Cangelosi, A. (2010), Intelligent Autonomous Controllers based on Genetically Evolved Neural Networks for Flying Robots: Experiments in Two and Three Dimensions (abstract). In *Proceedings of PCCAT 2010, the Postgraduate Conference for Computing: Applications and Theory.*

Ruini, F., and Cangelosi, A. (2009), Extending the Evolutionary Robotics approach to Flying Machines: an Application to MAV Teams. *Neural Networks (Advances in Neural Networks Research: IJCNN 2009)*, 22, 812-821.

Ruini, F., Cangelosi, A., and Zetule, F. (2009), Individual and Cooperative Tasks performed by MAV Teams driven by Embodied Neural Network Controllers. In *Proceedings of IJCNN 2009, the International Joint Conference on Neural Networks*, pp. 2717-2724.

Ruini, F., and Cangelosi, A. (2009), Un Modello 3D di Robotica Evolutiva per lo Sviluppo di Controller Autonomi per Robot Volanti. In Miglino, O., Ponticorvo, M., Rega, A., and Rubinacci, F. (eds.), *Modelli, Sistemi ed Applicazioni di Vita Artificiale e Computazione Evolutiva. Atti del VI Workshop Italiano di Vita Artificiale e Computazione Evolutiva (WIVACE 2009)*, pp. 177-185.

Ruini, F., and Cangelosi, A. (2008), Distributed Control in Multi-Agent Systems: a Preliminary Model of Autonomous MAV Swarms. In *Proceedings of FUSION 2008, the Eleventh International Conference on Information Fusion*, pp. 1043-1050.

Ruini, F., and Cangelosi, A. (2008), Evolutionary Algorithm based Neural Network Controllers: an Application to MAV Swarms (abstract). In *Proceedings of WIVACE 2008, the V Italian Workshop on Artificial Life and Evolutionary Computation.*

Word count for the main body of this thesis: 75,975

Signed: _____

Date: _____

Chapter 1

Introduction

In this thesis we introduce two new approaches to the design of distributed autonomous controllers for the collective behaviour of teams of unmanned aerial vehicles.

The previous sentence highlights the main keywords of this work. We look at autonomous intelligent control with a focus on distributed control and collective behaviour, while always having fixed-wing Micro-unmanned Aerial Vehicles (MAVs) as a reference.

Autonomous intelligent control is the discipline which is focused on the design of automatic systems able to make a robotic vehicle perform a certain complex task without the need for a human expert to be part of the control loop. The robot gathers information about the environment in which it has to perform using its embedded sensory apparatus or receives this information from an external element capable of data collection and transmission. The available information is then interpreted by the robot and a proper behavioural response, in accordance to the task given, is generated and executed.

Distributed control is a notably interesting problem from a scientific perspective and has therefore been the subject of countless numbers of publications. The idea behind distributed control consists in governing the behaviour of a team of robots (*i.e.* collective behaviour) by sharing the tasks of information gathering and of its following processing amongst all the members of the group. The main advantages provided by this approach can be identified in the non-critical reliance of the system

upon any central element, with the obvious consequences in terms of reliability (fault-tolerance) and flexibility in front of variations in group sizes.

Aerial vehicles differ in many ways from ground-based wheeled robots. This difference is more pronounced for fixed-wing aircraft due to their very characteristic motion constraints. The most common robots used for experiments in collective robotics are either omni-directional in their motion (as the Khepera and e-puck platforms, widely employed in Evolutionary Robotics), or they can modify their speed to significant extents going as far as being able to stop, change heading direction and then start moving again (this is true for both ground-based wheeled robots and aerial vehicles capable of hovering, such as helicopters). The motion constraints of fixed-wing aircraft makes the control task significantly more challenging. For example, not being able to stop in mid-air (this would cause a stall), fixed-wing aircraft require every movement to be carefully planned in advance taking into account the potential consequences of any single action.

The first of the two approaches proposed herein to implement a distributed control system capable of dealing with the aforementioned constraints is a hybrid one, based upon Evolutionary Robotics and Multi-Agent Systems methodologies. What we aim to demonstrate is how, in the light of the latest technological innovations, Evolutionary Robotics can be considered as a valid candidate methodology for the design of distributed autonomous controllers for groups of Micro-unmanned Aerial Vehicles. The technological innovations we are referring to can be identified in: the availability of affordable, robust and easily operated MAV platforms, the widespread introduction of miniature sensors and electro-mechanical components to activate the control surfaces of the aircraft, and the accuracy in flight stabilisation provided by most of the autopilot systems available on the market. Such autopilots can be easily integrated into the fuselage of a small aerial vehicle and connected to an onboard computer which, in addition to having access to the readings coming from all the sensors installed on the aircraft, can also issue orders (flight instructions) to it to be executed. The peculiarity of the proposed approach is its reliance on simplistic non-physics based computer simulators that approximate from a very high level the

flight dynamics of MAV platforms. The use of simpler simulators makes it possible to use the Evolutionary Robotics paradigm for tackling problems significantly more complex than simple flight planning, with the evolutionary process taking place in a reasonable time-frame. The author's belief is that the autopilot systems available nowadays make it possible, as long as the software simulators used for the evolution of the controllers implement "plausible" flight motion dynamics, to overcome the "reality gap" issue, *i.e.* the mistakes made by a controller designed in simulation when facing the complexity of the real world.

The second proposed approach is based instead upon the principles of "flocking" as originally introduced by Craig Reynolds. This part of the study will not receive the same amount of attention dedicated to the exploration of the Evolutionary Robotics based methodology. Originally, this thesis was thought in fact to be exclusively oriented towards ER. Then, thanks to a collaboration with the Laboratory of Intelligent Systems at the EPFL in Lausanne, the author has had the chance to carry out some experiments on physical robots. Unfortunately it was not possible to replicate the scenarios elaborated in simulation, due to several reasons. For example, no obstacle-avoidance behaviours could have been implemented on physical robots because of the impossibility of carrying out experiments within urban areas (or to deploy obstacles about $100m$ tall in the test field used), and because of the lack of sensors with specifications matching those of the ultra-sonic sensors simulated in the computer models. Deploying a target to be followed by the aircraft was a non-trivial task as well. The decision upon the research direction to pursue has therefore been made in agreement with the scientific interest of the colleagues working at EPFL. What we decided to do was to implement a flocking algorithm (*i.e.* a distributed control system) on computer simulation (to test how the original Reynolds' algorithm could be adapted to the motion constraints of fixed-wing aircraft) and then on real robots. The significance of the experiments carried out goes far beyond obtaining flocking behaviour in reality. As the hand-designed controllers run on the aircraft only produce a single output value, this condition matches exactly the scenario we have simulated in the first of the two Evolutionary Robotics

computer models developed. Considering that the simulator used for testing the flocking behaviour has been built on the same design principles as the two models employed for the ER experimentations, we obtained indirect confirmation of how the evolved controllers could be successfully applied to real robots.

1.1 Contribution to knowledge

This thesis contributes to knowledge in several ways. A short summary, which the reader can use as an additional taster of the contents that will follow, is provided herein:

- *extension of the Evolutionary Robotics approach to the domain of collective aerial robotics*: the main contribution of this thesis consists in demonstrating how the Evolutionary Robotics design principles can be successfully applied to the domain of collective aerial robotics. This is a case in which a relatively well-known methodology (Evolutionary Robotics principles) has been employed on a different subject than usual, not properly investigated by the available scientific literature.

  The demonstration has been done in an indirect way. Most of the work has been carried out using computer simulations, where autonomous neural network controllers were evolved to deal with different experimental scenarios. At a later stage only tests have been conducted on real robotics platforms. Although, most of the experimental setups studied in simulation could have not been reproduced in the real world due to several constraints (*e.g.* impossibility to carry out experiments within urban environments due to safety reasons, impracticability in recreating obstacles $50m$ tall, etc.). Thus, once the simulations had demonstrated that autonomous controllers could have been successfully designed relying on Evolutionary Robotics design methodologies, knowing that real robotics platforms (thanks to modern autopilot systems) could be governed using the same set of output values generated by the neural networks controllers evolved in simulation has been interpreted as a demon-

stration of the feasibility of the chosen approach.

The research focus has been oriented towards fixed-wing aircraft as these aerial platforms are considered by the author the most challenging ones (compared for example with helicopters and multi-rotor aerial vehicles) from a control point of view. The main associated motion constraints typical of fixed-wing aircraft have been taken into account, as well as the issues associated with distributed control. Having focused the research on fixed-wing aircraft, the findings identified should smoothly apply to simpler aerial platforms too.

The prospect of employing Evolutionary Robotics techniques in the aerial robotics domain makes it possible to benefit of all the typical advantages provided by this approach (maximum exploitation of the reference environment, identification of non-obvious solutions, easy implementation of sophisticated control algorithms, etc.), thus extending in a considerable way the array of potential applications of autonomous fixed-wing aircraft.

Most of the scientific publications derived by the work the author carried out during his Ph.D. programme discuss the advantages associated to the chosen approach. Amongst them notable mentions go to: "*Distributed Control in Multi-Agent Systems: A Preliminary Model of Autonomous MAV Swarms*" [319] presented at FUSION 2008 (International Conference on Information Fusion); "*Evolutionary Algorithm based Neural Network Controllers: an Application to MAV Swarms*" [320] presented at WIVACE 2008 (Italian Workshop on Artificial Life and Computational Evolution); "*Extending the Evolutionary Robotics Approach to Flying Machines: An Application to MAV Teams*" [321], appeared in 2009 on the Neural Network journal;

- *software simulators and reality gap*: the investigations carried out demonstrate that using complex, realistic, and computationally heavy physics-based software simulators is not a compulsory requirement for the design of autonomous controllers to be eventually transferred onboard real fixed-wing aerial robots. As components such as autopilot systems that can take care of the low-level control issues (as well as executing instructions provided by several means) ex-

ist, the task of designing autonomous controllers for high-level behaviours becomes significantly easier. Software simulators implementing "realistic" flight dynamics, although not necessarily 100% accurate from a physics viewpoint, can be successfully employed for this purpose. This opens the door for a wide range of computational intelligence methods previously thought to be excessively time-consuming (as for example artificial evolution), to be applied to the domain of aerial robotics.

Other than in most of the publications cited above, this point has also been discussed in "*Intelligent Autonomous Controllers Based on Genetically Evolved Neural Networks for Flying Robots: Experiments in Two and Three Dimensions*" [325], presented at PCCAT 2010 (Postgraduate Conference for Computing: Application and Theory);

- *flocking behaviour on real aircraft*: as far as the author is aware, the experiments carried out in collaboration with the Laboratory of Intelligent Systems of the EPFL have been the first in which Reynolds' flocking algorithm was adapted and tested on real fixed-wing aerial robots. As this algorithm is fairly simple to implement and light from a computational point of view, an average onboard computer can employ it and still be left with enough available resources to run other tasks in parallel with navigation. This is an extremely interesting point, with a potentially significant impact on the field of distributed control, as the designer of such systems can rely on this algorithm as a solid base for navigation, thus concentrating instead on the more important tasks.

  The joint work in which Reynolds' flocking applied to fixed-wing aircraft has been investigated has resulted in the publication of a paper titled "*Reynolds flocking in Reality with Fixed-Wing Robots: Communication Range vs. Flight Dynamics*" [157], presented at IROS 2011 (International Conference on Intelligent Robots and Systems);

- *investigations on input encoding, genetic operators and incremental evolution*: although not providing conclusive results, the investigations carried out along with the various experiments add more elements to the technical discussion about methodologies both in Evolutionary Robotics and the broader evolutionary computation fields.

  The results obtained on incremental evolution are the main topic of a paper presented at TAROS 2010, titled "*An Incremental Approach to the Evolutionary Design of Autonomous Controllers for Micro-unmanned Aerial Vehicles*" [324];

- *availability of the software simulator*: the software simulators developed by the author during his Ph.D. research all rely on open-source libraries and have been made freely available on the Internet. Since an extremely careful modelling of the specific aerial platform used as reference is not required, the very same simulators can easily be modified and employed by any researcher willing to carry out investigations in the area of intelligent control for aerial robots.

  Careful descriptions of the software simulators have been published. For what concerns the 2D model the reference publication is "*Individual and Cooperative Tasks Performed by Autonomous MAV Teams Driven by Embodied Neural Network Controllers*" [326], presented at IJCNN 2009 (International Joint Conference on Neural Networks). The 3D simulator has been described in detail in "*An Evolutionary Robotics 3D Model for Autonomous MAVs Navigation, Target Tracking and Group Coordination*" [323], presented at IJCNN 2010.

1.2 Thesis outline

The thesis is structured as follows:

- Chapter 2 introduces the field of Evolutionary Robotics, contextualising it along an historical perspective that started with "Shakey the robot" and

the classic approach to autonomous robotics which was revolutionised by Rodney Brooks' Behaviour-Based Robotics, and eventually ended up in the evolutionary-based approach we adhere to. Two main sections are dedicated to Neural Networks and Evolutionary Algorithms respectively, as they are the two main instruments upon which Evolutionary Robotics is based. For what concerns neural networks, a simple mathematical treatment of its core components is provided;

- Chapter 3 concludes the introductory part of this thesis by presenting the field of aerial robotics. The historical development of the field is outlined as well and the most commonly applied scenarios in which UAVs and MAVs can be employed are described. The chapter goes on to illustrate the main design issues involved in the development of miniature aerial platforms, as well as the basic aerodynamics and the most compelling motion constraints related to fixed-wing aircraft. Finally, the topics of remote and autonomous control are introduced;

- Chapter 4 links the two introductory chapters with the following ones related to the experimental part of this thesis. This chapter provides a scientific literature review focused on the publications dealing with the issue of designing intelligent controllers for aerial robots. The author then describes the characteristics of the two alternative approaches he has proposed (introducing at the same time the concept of Reynolds' flocking), examines the main challenges involved and introduces the plan for the simulation experiments described in the following three chapters;

- Chapter 5 presents the first of the three computer simulators developed for the purposes of this thesis. The one analysed here is a 2D simulator that approximates from a high-level perspective the flight behaviour of real aircraft. Nevertheless, it incorporates what we believe is the most relevant trait of fixed-wing aerial robots, *i.e.* the constraint of continuous motion associated to a limited turn rate. Several experimental setups have been elaborated and their

results presented throughout the chapter, including: navigation through an obstacle free environment, obstacle avoidance behaviour within an urban-like environment, tracking of targets moving at different speeds and cooperative behaviour based on implicit communication strategies;

- Chapter 6 introduces a new three-dimensional simulator which, although being developed on the basis of completely different software instruments, is closely related to the 2D one presented in the previous chapter. The 3D simulator has been designed to replicate, with a different level of approximation in the flight dynamics, the results obtained previously (with only the exception of obstacle avoidance). Side by side with the experiments on navigation, additional aspects are analysed. We refer in particular to passing from a single to a multi-thread computer simulator and to the adoption of the incremental evolution paradigm;

- Chapter 7 concerns flocking behaviour and the issues arising when moving from computer simulations to experimentations on real flying robots. This is the second of the two approaches proposed in this thesis. Despite not relying on Evolutionary Robotics, this part of the research has also required the development of a software simulator in order to test the various variations of the flocking algorithm proposed. The characteristics of the simulator are described in detail, as well as the results obtained in it and when the controllers have been transferred to real robots;

- in the Conclusions section we look back at the various topics touched upon this thesis, attempting to objectively evaluate the results obtained. Plans for future work, both in terms of possible improvements and completely new research directions, are also described;

- the Appendix is structured into four sections. It starts by presenting two detailed reviews respectively focusing (1) on the most prominent MAV platforms developed for military, scientific and entertainment purposes, and (2) on the most complete autopilot systems for aerial robots commonly available on the

market. One specific section describes the functioning of P-ARTS (Plymouth Advanced Robot Training Suite), the computer grid facility used for carrying out the most computation intensive of the simulation experiments presented in this work. The remaining part contains some of the mathematical/trigonometrical formulas used across the pages of this thesis that are often referred to.

Chapter 2

Evolutionary Robotics: Neural Networks and Evolutionary Computation Working Together

This chapter lays the foundations for the methods that will be used in the rest of this thesis. Since the work presented herein deals with Evolutionary Robotics, we will provide a detailed overview of the Evolutionary Robotics field. This will include a short historical background, exploring both the domain in itself, and the two main subfields that constitute it: Neural Networks and Evolutionary Computation. A short section has been dedicated to control theory in order to put the Evolutionary Robotics approach in the proper context.

2.1 Towards Evolutionary Robotics (ER)

Evolutionary Robotics (ER) [112, 113, 141, 280] is a methodology for the automated design of control systems ("controllers") for autonomous robots. The field of ER can be introduced by defining some of the keywords mentioned in the previous sentence: control systems, autonomous, robots.

2.1.1 Autonomous robotics

The roots of the word "robot" originate from the Old Church Slavonic term "rabota[1]", a word which can be effectively translated as both "servitude", and "work". These two possible translations reflect quite accurately the aim of those who first seriously started to work on building robots. Aim which consisted in creating "slaves" capable of carrying out physical hard work in place of their human "masters". Even modern dictionaries provide us with definitions oriented towards the professional role played by robots, stating for example that they are "*machines capable of carrying out complex series of actions automatically*[2]".

Given the above considerations it is hardly surprising that the most common examples of robots we can see nowadays are the industrial robots used across factories all over the world. Industrial robots are typically "blind" with respect to the environment in which they operate, *i.e.* they only have scarce interactions with it and they are generally programmed to perform the same job at all times. They do not have the chance to decide, autonomously, how to behave in relation to the requirements of the environment, thus accurately reflecting the lexical meaning of the word "rabota".

In this thesis we are looking at different kind of robots. We refer to robots that can interact in more flexible ways with the surrounding environment (whether it is a merely physical environment, or one involving interactions with living entities) and that can exhibit several behaviours, *i.e.* be considered "multi-purpose". We also refer to robots that can often move across the reference environment and that are more flexible, in the sense they can take decisions about how to reach a certain goal. All of these characteristics, especially self-determination, are characteristics we look at when we talk about "autonomous intelligent robots".

The above definitions do not mean that an autonomous robot must necessarily be able to achieve different goals (multi-goal controllers are still an active area of research, often associated to high-level psychological concepts as motivations and emotions [209, 327]). What we stress here is that an autonomous robot has at

[1] `http://www.etymonline.com/index.php?search=robot`
[2] `http://www.oxforddictionaries.com/definition/robot?view=uk`

least some degrees of flexibility in deciding how to behave at any given time. This flexibility is granted by the robot's controller. Throughout this thesis we will define as "controller" (or "control system") the software that controls the behaviour of a robot, making it possible for it to achieve a certain goal. A more specific definition comes from the control theory field [18], according to which a controller is:

> "*[...] a device which monitors and affects the operational conditions of a given dynamical system. The operational conditions are typically referred to as output variables of the system which can be affected by adjusting certain input variables.*"

Interesting additional considerations on the concept of autonomous controllers can be found in the seminal work by Patcher and Chandler [290], while the topic of "intelligent control" can be investigated by looking at the research done by Antsaklis and colleagues [14, 15]. Section 2.2 will provide the reader with a quick introduction to the domain of control theory.

The controller is the crucial part of any robotics application. Designing intelligent controllers for autonomous robots is, in essence, what this thesis is all about.

2.1.2 The classical approach to autonomous robotics: Shakey the robot and the problem of planning

The earliest methodologies employed to design controllers for autonomous robots, developed during the 70s of last century, will be defined in the rest of this chapter as the "classical approach" to robotics.

The classical approach builds upon a cognitivist interpretation of intelligence, according to which there is a clear separation between the body, intended as a physical entity, and the mind, something which has the ability of processing information. According to this school of thought, an autonomous intelligent system is constituted by three independent sub-systems: a perceptual system, a planner, and a motor system. The perceptual system extracts useful information from the external environment through sensory readings generated by a set of sensors. The planner, based on the current situation, has the task of planning a sequence of actions allowing the

robot to achieve its goal. Finally, the motor system must translate the sequence programmed by the planner into a set of motor actions that implement the desired behaviour.

Shakey the robot [278] - developed at Stanford as a practical demonstration of the applicability of the problem solver STRIPS (Stanford Research Institute Problem Solver) [108] to the domain of autonomous robotics - is arguably the best example of the classical approach to autonomous robotics.

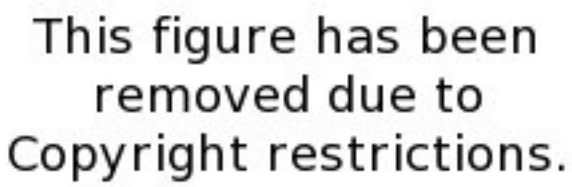


Figure 2.1: Shakey the robot. Source: `http://www.ai.sri.com/shakey/`

Shakey was a mobile robot with the task of executing simple actions inside a house-like building, *i.e.* a structure composed of one corridor and a series of rooms containing certain sets of objects. The typical action Shakey was required to do was to look for an object in a specific room and take it to a different room.

In order to find the solutions to the different problems it was asked to tackle, Shakey had to define a planning algorithm for each of them. A planning algorithm essentially consists of a sequence of actions that the robot has to perform in order to achieve the desired goal, *i.e.* it represents a solution for the current problem. Such a solution must satisfy several criteria, as for example: *effectiveness* (the solution must lead to the solving of the task), *completeness* (any precondition implied by the various actions must be verified before proceeding through the sequence), and *consistency* (the solution can not contain contradictory actions).

The technique employed for Shakey - which involves the use of a high-level planner fed with sensory information and controlling a motor system - can be defined as "planning". One important characteristic of this approach lies in the independence of the three components of the control system. Since the planner does not have direct access to the external environment in which the robot operates, it must rely on descriptions of the surrounding world elaborated by the sensory (perceptual) system. Output motor actions are separated from the planner as well, and are executed sequentially once the solution identified by the planner has been properly translated into simple motor directives.

As discussed in more details by Beer [32], the independence of the different layers constituting the entire control system leads to several limitations. These can be summarised in five different categories. 1) *environment representations*: having a planner independent from the perceptual system implies that the latter must create an extremely accurate representations of the environment and communicate it to the planner, which is not an easy task by any means; 2) *egocentric descriptions*: the difficulty of generating accurate environment representations is further complicated by the fact that robots like Shakey use egocentric perceptual systems, thus not having a bird's-eye view of the environment in which they are operating, but merely a very limited one (although, it might be argued, a three-layer architecture does not necessarily mean that the perceptual system has to be an egocentric one); 3) *noise*: when the input information is incomplete, noisy or partly wrong, there is no guarantee that the planner will elaborate an efficient plan; 4) *dynamically-changing environments*: since any non-controlled environment can change at any given time, the correct execution of a plan developed in advance by a robot cannot be guaranteed; 5) *adaptiveness*: given the dynamic nature of the real environments, an additional problem arises: how can a robot deal with an unexpected modification of the environment that occurs during the execution of an action and requires an immediate response (*e.g.* an obstacle suddenly falls in front of the robot while it is moving forward)?

2.1.3 Behaviour-based Robotics: embodiment and situatedness

In order to solve the issues of classic robotics, Rodney Brooks proposed in 1991 a detailed critique to the current state of research in artificial intelligence, introducing at the same time a novel approach to autonomous robotics, called Behaviour-based Robotics [54].

The core of Brooks' criticism [52] is based on the idea that the AI field (thus reflecting on the classic approach to autonomous robotics) has been developed focusing too much on human intelligence, and particularly on what human intelligence looks like today, ignoring it being the result of thousands of years of evolution. Abstract reasoning and symbolic representations are usually considered as "granted", but the entire evolutionary process that has led to their appearance is ignored. Furthermore, relying on the theoretical frameworks elaborated for example by Gibson [135] and Varela [379], Brooks criticised the use of abstract representations and symbolic manipulation as the proper tools to identify the most prominent characteristics of natural intelligence. According to Brooks, an intelligent system can not be interpreted as a completely abstract system separated from the physical world. It necessarily has to account for its own body and for the real environment in which it operates, rather than on a representation of it.

Later, Pfeifer and Scheier [297] reinforced the theoretical ground of this approach stressing the concepts of "embodiment" and "situatedness". With the term *embodiment*, the two scientists refer to the fact that, having a body, an intelligent system/agent is continuously subject to physical forces, to energy dissipation, to being damaged, and, more in general, to any kind of influence exercised by the environment. *Situatedness* refers instead to the property of an intelligent system that, being situated inside a real environment, can directly interact with it without the need for symbolic representations.

Brooks believes that a system can be considered "intelligent" only if it can implement sensorimotor behaviours within a dynamic and changing environment as the real world generally is. The building blocks of an intelligent system must then

be the simplest sensorimotor behaviours, on top of which increasingly sophisticated behaviours can be built. Each of these building blocks defines a basic but complete behaviour and can control the robot directly accessing the information contained in the real world. These basic behaviours can be seen as working in parallel, as opposed to the seriality characterising the classical approach to autonomous robotics. Figure 2.2 graphically illustrates this point.

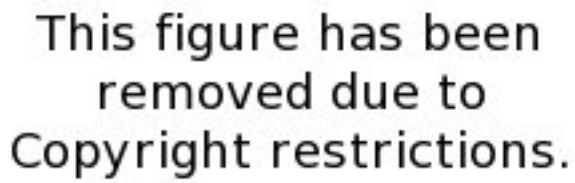


Figure 2.2: Graphical representation of the Behaviour-based approach. Source: [33]

Steels [357] reinforced the theorisation of artificial intelligence introduced by Brooks by coining the term "Behaviour-oriented AI" to identify the pool of scientific disciplines that study how the behaviour of agents emerges and becomes intelligent and adaptive:

> "*[...] the success of the field is defined in terms of success in building physical agents that are capable of maximising their own self-preservation in interaction with a dynamically changing environment.*"

Genghis [51] (see Figure 2.3), a six-legged (hexapod) walking machine capable of walking over rough terrain and following a person passively sensed in the infrared spectrum, was among the first examples of robots built by Rodney Brooks and his group relying on their new design principle, summarised in what they defined as the "subsumption architecture" [50].

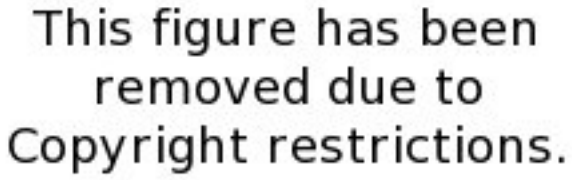


Figure 2.3: Genghis the robot. One of the first robots built by Rodney Brooks' research group relying on the Behaviour-based robotics approach. Source: `http://www.ai.mit.edu/projects/genghis/`

Behaviour-based robotics differs from the planning approach in a number of aspects. The most important one is related to the decomposition of the overall behaviour exhibited by the robot into a series of simpler behaviours, from the most basic (*e.g.* move along a straight line) to the complex ones (*e.g.* avoid an obstacle). Any sub-behaviour independently determines the robot's activity and takes control over the entire system when the contingent situation requires it (*e.g.* the module dedicated to moving the robot along a straight line leaves the control to the obstacle-avoidance module whenever the robot's sensors detect an obstacle in front of it). The outcome is that the overall robot behaviour is not determined by a pre-planned rigid sequence of actions that must be updated whenever changes happen in the environment, rather it is the emergent result of a highly complex interaction between all the sub-behaviours the robot is capable of.

Furthermore, at no time during the execution of a given behaviour does the robot have an explicit and complete internal representation of the environment in which it operates. Rather it relies on very partial and continuously changing depictions of the surroundings, depending on their relevance to the behavioural module in use. These "representations" are essentially free of any descriptive character, and can be seen instead as the series of stimulus-response reactions the robot can implement,

i.e. the sensorimotor behaviours generated by each module.

However Behaviour-based robotics suffers from three limitations. The designer of a behaviour-based system still plays a crucial role in the development stage. It is the experimenter who has to define how a certain behaviour must be decomposed in its subparts, something which is not objective thus strongly depending on the designer's point of view. Second, there is the problem of categorisation. In order, for example, to trigger an obstacle-avoidance behaviour, the robot must know that the object it is facing is indeed an obstacle. How to classify obstacles is an activity that the designer has integrated into the robot, relying on what he believes the sensorial experience of the robot might be. Overall, even in Behaviour-based robotics the designer's footprint is present and (often indirectly) visible. Finally, as reported for example by Cliff [73], as the number of dedicated modules in a behaviour-based architecture increases, the complexity of the interactions between the individual parts arises quickly, soon becoming intractable.

2.1.4 Evolutionary Robotics

The Evolutionary Robotics approach represents for the experimenter a departure from the task of analysing and designing the controller for an autonomous robot, introducing an automated design procedure instead. ER is inspired by the thought experiments carried out by Valentino Braitenberg [49] during the 1970s and the 1980s, and by the field of Artificial Life [202] that at the time was receiving a great deal of interest from the scientific community.

Braitenberg's vehicles

"*Vehicles: Experiments in Synthetic Psychology*" [49] is the title of a book written by Valentino Braitenberg during the early 1980s. In his book, the Austro-Italian neuroscientist describes a series of thought experiments in which small autonomous "vehicles", driven by extremely simple internal controllers interacting with the external environment, behave in unexpectedly complex ways.

His research can be seen as one of the first "demonstrations" (although, as men-

tioned above, the work was a thought experiment) of how complex behaviours can emerge from the interaction between the environment and extremely simple controller architectures. Braitenberg stretched (possibly a little bit too far according to some critics, as [63]) the interpretation of the behaviours displayed by his (imaginary) robots, using terms as "fear", "aggression", "love", "foresight", and even "optimism". Nonetheless, the issue highlighted by the scientist is that if we do not already know the principles behind the vehicles' operation (that are extremely simple indeed), just looking at the high level behaviour exhibited by the vehicles we might end up with a complete misunderstanding of their basic working principles. Braitenberg describes this phenomenon as the "*law of uphill analysis and downhill invention*", meaning that it is much more difficult to try to guess internal structure just from the observation of behaviour than it is to create the structure that gives the behaviour.

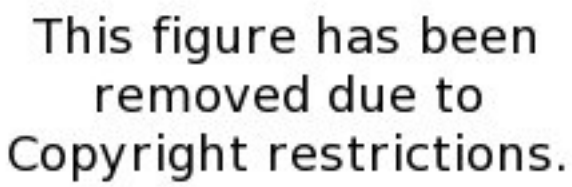
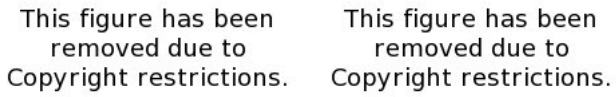


Figure 2.4: Sketch of two Braitenberg's vehicles respectively attracted (a) and repulsed (b) by a light source. Source: [49]

The importance of Braitenberg's work is noteworthy from a philosophical perspective and it has provided a great source of inspiration for research in autonomous robotics. While the philosophical implications have been briefly mentioned above, Braitenberg's research has helped to spread out an optimistic way of looking at the research in autonomous robotics, suggesting that complex (intelligent) behaviours

may be easier to achieve than we believe. Moreover, the vehicles Braitenberg imagined have proven to be fairly simple to build, leading to the appearance of at least two families of robots built on similar principles, *i.e.* the Khepera [260] and the e-puck [259] (see Figures 2.5(a) and 2.5(b) respectively). The research in Evolutionary Robotics has made great usage of robots belonging to these two families [280].



(a) (b)

Figure 2.5: Two real robots taking inspiration by Braitenberg's vehicles: (a) Khepera; (b) e-puck. Sources: [260, 259]

The influence of Braitenberg's work has spanned almost three decades. Although his thought experiments never involved artificial evolution (although references were made to this possibility across the test), some researchers, for example Salomon [331], recently proposed interesting work focusing on the design/optimisation of Braitenberg-like controllers through evolutionary methods. Other work, focusing on the evolution of the robots' sensors, are exemplified in [228].

Artificial Life

The term "*Artificial Life*" was first introduced by Christopher Langton [202] who defined it as:

> "*Artificial Life ("AL" or "ALife") is the name given to a new discipline that studies "natural" life by attempting to recreate biological phenomena from scratch within computers and other "artificial" media. ALife complements the traditional analytic approach of traditional biology with a synthetic approach in which, rather than studying biological phenomena by taking apart living organisms to see how they work, one attempts to put together systems that behave like living organisms.*"

The above definition is nicely summarised by the one provided, later on, by Domenico Parisi [288]. According to the Italian scientist, Artificial Life is:

> *"[...] the study of living systems, carried out not dissecting and analysing living systems existing in reality (as biology does), rather building living artificial systems from the scratch."*

These two definitions look similar to each other (and both of them fail in defining with accuracy what "life" actually is) and relatively general. The research in the field carried out over the years has extended to cover several different areas concerning the study of the systems related to life, its processes, and its evolution. Many natural phenomena can now be examined through the lens of AL. The key concept which operates as an umbrella linking together apparently unrelated areas of research into artificial life is "emergentism" [48], which represents the idea that the properties exhibited by a system are not necessarily the "sum" of the properties exhibited by its sub-parts, rather the result of complex and often unpredictable interactions between them.

Nowadays at least three main AL research streams can be identified:

- *computer models*: complex phenomena too difficult to be approached through analytical methodologies can be investigated via the development of dedicated computer models. Cellular Automata [395] and Agent-Based Modelling (ABM) [44], thanks to their innately complex nature, are the primary methodologies used to study social phenomena. Computer models of Artificial Life do not necessarily need to involve multitudes of individuals, since complex aspects can also emerge from the interaction between single individuals and the environment in which they live. At the same time, various other phenomena can be studied in computer models adopting a hybrid artificial life/complex systems approach (e.g., volcanic eruptions [329], stock markets [286], etc.). The scientific career of the aforementioned Domenico Parisi focused on the study of psychological (and sociological) phenomena through their replication in computer simulation models [5, 256, 340]. To achieve this goal, the approach proposed by the Italian scientist often consisted in building artificial organisms according to the same metaphor being employed by Evolutionary Robotics, *i.e.* with a neural network representing the "brain" of the organism (to differentiate between these neural networks and "standard" ones, Parisi

coined the term "*ALNN*", standing for "*Artificial Life Neural Networks*") and an evolutionary algorithm as the tool required for its evolution. Langton, as well, has been working on computational models for most of his career (see, as an example, the popular "*Langton's ant*" [201]);

- *robotics*: the role played by robotics in Artificial Life is a quite a controversial one [53]. The main reason for using robots in AL simulations consists in the possibility of extending (wherever possible) the scope of the experiments carried out using simulated computer models, taking into account the concepts of embodiment and situatedness. In reality, even artificial agents "living" within small agent-based models can often be seen as embodied and situated, thus blurring the line between computer simulated organisms and physical robots operating in real environments;

- *biochemistry*: biochemistry, theoretical biology and complex systems were considered the most relevant areas with regard to artificial life when, in the early 1990s, the ECAL (European Conference on Artificial Life) conference was held for the first time. Examples of work falling into these categories consist of models of protocells studied by Serra et al. [343]. More general approaches to the synthesis of life can be found in [307]. This area of Artificial Life is regaining a prevalent role today[3].

Although we will not focus extensively on the relation between the work presented herein and the Artificial Life field, the computer models presented throughout this thesis can be considered to be, to a certain extent, Artificial Life models.

Evolutionary Robotics: what it is and how it works

The Evolutionary Robotics approach aims to provide a solution to the issues related to Rodney Brooks' Behaviour-based robotics highlighted in Section 2.1.3. Although

[3]The website for the 2011 edition of ECAL (`http://www.ecal11.org/`) says, "*Back then, in the early 1990s, the first two ECAL conferences in Paris and Brussels were mainly centred on theoretical biology and the physics of complex systems. Today, we feel that Alife can look back on these origins and take more inspiration from new developments at the intersection between computer science and theoretical biologythus it is our wish to refocus the conference on complex biological systems.*"

nowadays the two approaches can sometimes appear to be in contrast with each other, ER can effectively be seen as an extension of Behaviour-based robotics in which even the basic behaviours - the building blocks - are left to an evolutionary design process rather than being manually specified by the experimenter.

The procedure for the design of an autonomous controller for a robot - according to the ER approach - relies on the employment of Evolutionary Algorithms (typically Genetic Algorithms, which is the name used to describe the category of algorithms that implement evolutionary processes similar to the one described herein (see Section 2.4.5). It begins with the creation of an initial set (population) of different[4] neural networks. These networks are subsequently associated, one by one, to the real robot. They act as controllers, processing the input information fed through the robot's sensorial apparatus[5] and producing an appropriate motor response which is then executed by the dedicated apparatus of the artefact. The robot is deployed in the environment and the performance of its controllers is evaluated in relation to the execution of a specific task. The initial population is called, in biological terms, the first generation. At the end of the evaluation phase, the controllers that have scored the best results are preserved, while the others are discarded. Several copies of the selected controllers are made, although slight changes in their behaviour are applied through the introduction of random modifications in the neural network parameters. This procedure leads to the appearance of a new population of controllers, of which individuals are again evaluated. The process is reiterated until control systems that are able to solve the reference task in an optimal way have appeared (see Figure 2.6 for a graphical representation of this approach).

As mentioned above, the preponderant aspect of this methodology consists in the fact that the experimenter/designer does not play any role during the evolutionary process, leaving to an automated procedure the responsibility of identifying the optimal solution to the problem he wants to solve. It is not uncommon for the

[4]At the beginning of the process, the various neural networks typically differ amongst them because of different connection weights and biases. Most of the time the topology is fixed, although this is not necessarily the case (see for example the NEAT framework [355]).

[5]In case of a pure sensorimotor controller. Neural network controllers can be much more sophisticated than these, dealing with input coming from inside the robot ("internal" environment), memories, predictions, etc.

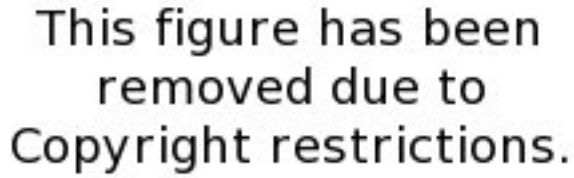


Figure 2.6: Graphical resume of how the Evolutionary Robotics approach works. Source: `http://nelsonrobotics.org/evolutionary_robotics_web/`

experimenter to find out that the evolutionary algorithm has discovered a completely unexpected, yet simple and efficient solution. This is partly due to the fact that the system analyses the environment in a much more accurate way than what the experimenter could do (thanks to the egocentric perception used by the robot), attempting to exploit any possible source of evolutionary advantage, often in ways that the external designer was oblivious to.

Other differences, highlighted among others by Marocco [229], concern for example the contrast between distal and proximal behaviour. The term "distal behaviour" can be used when referring to the behaviour as seen by an external observer and "proximal behaviour" can be used when we look at the same behaviour from inside the robot, namely through its sensory apparatus. Behaviour-based robotics tends to work from a distal behaviour perspective, because the designer uses this point of view to deconstruct the problem and design the dedicated modules. In reality, displaying a specific distal behaviour is not the goal of an agent. What an agent can really do is to implement proximal behaviours that allow it to achieve its task. Distal behaviour emerges as side effect of a series of proximal behaviours[6], each of these strongly depending on the structure of the environment. Evolutionary robotics does

[6]It is worth considering that the same distal behaviour can be the results of several alternative combinations of proximal behaviours.

not take into account the distal behaviour components (because the human designer, which is generally the reason for the introduction of such a bias, does not intervene during the evolutionary process) focusing exclusively on proximal behaviours and consequently stressing the concepts of embodiment and situatedness. The strong relationship with the environment has been carefully studied by Nolfi and Parisi [282], highlighting among several other aspects how every single movement performed by an agent affects the sensorial perception that it will experience in the immediate future, thus triggering a whole chain of subsequent behaviours.

As a side note, from a technical and historical point of view it should be mentioned how training neural networks using genetic algorithms rather than "traditional" learning algorithms was an idea investigated by Montana and his colleagues since the late 1980s [262], following the return of interest in neural networks pushed by the introduction of the backpropagation algorithm. A more detailed survey of the ways in which neural networks and evolutionary algorithms have been combined together across the scientific literature can be found in [334].

Of course the Evolutionary Robotics approach is not free of issues and limitations. For example, Miglino et al. [253] published an interesting analysis on the role played by the computation time factor, thus highlighting the need for computer simulations (which nowadays is, by far, the most commonly used approach) for the evaluation of robots' performances. Furthermore, for practical applications there is still a general skepticism upon the use of neural network controllers for robots dealing with sensible tasks (*e.g.* in the military domain). Despite studies as the one recently published by Hauert and colleagues [160], where it has been demonstrated that the reverse engineering of evolved neural controllers is possible, many researchers still believe that neural networks work as unpredictable black boxes, thus suggesting that their use should be avoided in tasks for which a high degree of accuracy is required. Nonetheless, several modifications of the "basic" ER approach are possible to address some of its drawbacks. These alternative methodologies consist for example in avoiding the "*embodied trials*" as suggested by Ficici and Pollack [107] to speed up the evolutionary process. Furthermore, several works have focused on

reducing the so-called "reality-gap", either mapping the reference environment using robots' sensors and integrating this information within the computer simulators used (as suggested by Nolfi *et al.* [281]), or applying Pareto-Based Multi-Objective Evolutionary Algorithms [194]. We will return on the reality-gap issue later on.

2.1.5 A "cognitive" approach? The link with epigenetic robotics

Although we have introduced the field of evolutionary robotics focusing on its role as an instrument for the automated design of autonomous controllers for robots, the ER approach can also be used to play more "cognitive" roles.

Robotics in itself (see the brief discussion we have made when introducing the field of Artificial Life) can be seen as a modelling tool to create, test and validate theories about cognitive phenomena [408]. As suggested for example by Marocco [229], working in close contact with reality, robotics forces the models to be both solid and rigorous (as for any computer model of a psychological theory), and to necessarily take into account the complexity of a real environment. Thus providing a platform for the study of the role played by embodiment and situatedness.

A theoretical framework for the application of evolutionary robotics to research in psychology has been provided by Harvey *et al.* with the introduction of the "minimal cognition" theory [155], according to which:

> "*Cognition [...] can be broadly defined as the capability of an agent of interacting with its environment so as to maintain some viability constraint. It is not an internal property of the agent, but a relational property that involves both the agent, its environment and the maintenance of some constraint. Living organisms are naturally cognitive according to this definition as they need to engage in interaction with their environment so as to stay alive - but the term can also be applied to some artificial non-living systems, as long as we can clearly treat them as agents and their viability constraints are well specified (and these could be as artificial as maintaining certain relations with the environment, self-preservation, or the fulfilment of a pre-specified goal).*"

Applications of Evolutionary Robotics to the study of cognitive phenomena have touched several aspects as for example the emergence of communication [219,

230], the evolution of cognitive scaffoldings [409], the learning through organism-environment interactions (agent-based approach) [339], issues related to perception in children [396], etc. More generally, when focused on the development of cognitive systems, evolutionary robotics is often classified side by side with the emergent field of (Cognitive) Developmental Robotics [392] (now more popular under the name of Epigenetics Robotics [35, 185]) under the umbrella term of Cognitive Robotics[7] [71, 364]. Developmental Robotics, inspired indirectly by the vision of Turing [372] and more closely by the scientific fields of neural development and developmental psychology, focuses on the understanding of the cognitive developmental processes that a robot would have to experience in order to exhibit resulting "intelligent" (defined as "requiring cognitive capabilities") behaviours. This approach shares some aspects with Evolutionary Robotics (neural networks are one of the main instruments used, and the embodiment element is stressed), though it does not necessarily rely on evolutionary methods.

The principles behind Developmental Robotics have also inspired robot builders in designing humanoid platforms flexible enough to be used for experimentations in the area [17]. The most prominent example of this approach is certainly the iCub robot [333], built as an outcome of the European project Robotcub [332].

Across this thesis we mainly focus on neural networks as controllers for robots, and Evolutionary Robotics as a methodology for adjusting the connection weights and biases of these controllers in order to make them able to perform specific tasks. As we have already introduced, ER is based upon two main components, neural networks and evolutionary algorithms. In the next sections we will provide a detailed overview of both these fields. Before getting there, though, we must introduce the field of control theory.

[7] An attempt to define cognitive robotics has been made by British researcher Paul Baxter on his Internet blog, `http://paul-baxter.blogspot.com/2007/01/what-does-cognitive-robotics-mean.html`.

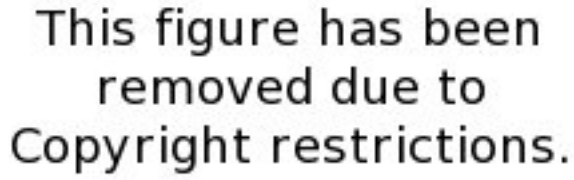


Figure 2.7: iCub robot. Source: `http://www.robotcub.org/index.php/robotcub/gallery/pictures`

2.2 The basics of control theory

The research domain specialised in the study and design of robots' controllers is control theory [177], one of the founding areas of modern robotics. In this section we briefly introduce some of the most important concepts in control theory, basing most of our considerations upon the work published by Maja Mataric in 2007 [233].

Within control theory we can identify two main families of control models: feedback control (or "closed loop" control), and feedforward (also "open loop" control). We will start this journey with a brief description of the concept of feedback control, together with some of the basics issues these sorts of controllers constantly face, before quickly introducing the concept of open loop control.

2.2.1 Feedback (closed loop) control

Feedback control [96] is a means of getting a robot (or, more in general, any sort of system) to achieve and maintain a desired state (which is usually called the "set point") by continuously comparing its current state with its desired state. This continuous comparison is made possible by the "feedback" mechanism, meaning that pieces of information helpful in determining the current state of the system are collected and sent ("fed") back to the systems controller. The aforementioned

desired state, also called "goal state", is where the system wants to be (or, in different terms, where its designer wants the system to be).

We can distinguish amongst two kinds of goals: achievement goals (states that the system attempts to reach, as for example finding its way out for a robot navigating through a maze) and maintenance goals (states that the system must maintain, as for example keeping a biped robot balanced and walking). The main difference between these two goals is in the amount of work to be performed by the control system to satisfy them. Achievement goals can simply be achieved or not, once the robot (or the system) reaches its goal the job is done. Conversely, maintenance goals require ongoing active work by the system. From an "historical" perspective, it might be interesting to note that the control theory field has traditionally been mostly involved with the design of controllers dedicated to maintenance goals, while AI has always been more interested in developing systems concerned with achievement goals.

It is important to keep in mind that the goal state of a system can be related either to internal or external states, or even to a combination of both. Think for example to the popular iRobots Roomba robots[8]. In their control systems we are able to identify both internal (keep the battery power level above a certain threshold, move to the docking station to get recharged when that level becomes too low) and external (vacuum the entire area) states that the robot must either maintain or reach. Of course, most of the time a system of any sort will not be in its desired state, but more or less far from it. This is where controllers kick in.

The error: magnitude and direction

Of fundamental importance for any kind of control system (either open or closed loop) is the concept of "error", which represents the difference between the current and the desired states of a system. The goal of any control system is, in its very essence, to minimise this error. In feedback control the error is explicitly computed and used by the system to modify its current state in order to get it closer to the desired state. The error can in fact constitute a great source of information.

[8] `http://www.irobot.com/uk/roomba/`

From a theoretical perspective the error can be characterised by two components: "direction" and "magnitude". To clarify these two terms let's think to the popular "hot and cold" game played by children all over the world. When the person running the game responds vocally (*i.e.* provides feedback) to the players actions, he is providing information about both the direction of the error (close or far to the target) and its magnitude (how close or far to the target). This information is processed by the player and used to correct his guess accordingly in order to get closer to the (unknown) target location. The very same thing happens in autonomous systems governed by feedback controllers. The system (*e.g.* a robot) collects information about the current state (for example through its sensors) and produces a response (*e.g.* a movement) aimed to switch to a new state closer to the desired one. In control theory the parameters that determine the magnitude of the systems response are called "gains".

Figure 2.8 graphically shows a typical feedback controller for a generic industrial plant. In this example the controller, receiving information about the current state from the sensors, evaluates whether the system currently lies in the desired state or not. If not, it operates a compensation sending motor commands to the actuators of the plant. Once implemented in their corresponding motor operations these commands drive the system to a new current state. The controller reads the new current state through its sensors and repeats the above procedure, until a certain goal is reached (in case it is an achievement rather than a maintenance goal).

When a human component plays a role somewhere within this control loop (for example in determining whether the system should perform a certain operation or not), it is said that we are facing a "man-in-the-loop" configuration.

There are several types of feedback control architectures. The mostly used ones are arguably proportional control, proportional derivative control, and proportional integral derivative control. These are commonly referred to as P, PD, and PID control respectively. In the next few sections we will explore their main characteristics.

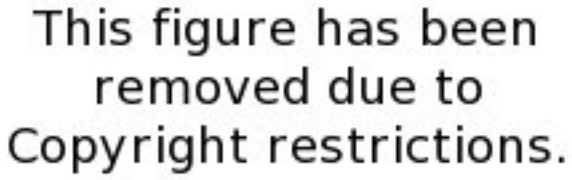


Figure 2.8: Diagram of a typical feedback controller. Source: `http://soundlab.cs.princeton.edu/learning/tutorials/RealTime/realtime.html`

Proportional control (P)

The idea behind proportional control simply consists in having the system to respond in proportion to the measured error (*i.e.* applying proportional gains), relying on both its direction and its magnitude. From a formal point of view a proportional controller generates an output o correlated to the input i thanks to the use of a certain proportionality constant (K_p in Equation 2.1, following the notation used by Mataric [233]).

$$o = K_p i \tag{2.1}$$

The value of K_p is task-specific and typically needs to be identified by the system designer going through a trial and error methodology.

With a little bit of imagination it must be easy to figure out how a proportional controller works in reality. Think for example of a robot whose aim is to drive along a wall, keeping a certain distance from it. A proportional controller governing this robot would measure the distance to the wall at specific time intervals and produce in response a steering manoeuvre which is proportional in terms of magnitude to the one of the error (*i.e.* the current distance to the wall) and which makes the error decrease depending on its direction (*i.e.* steering the robot towards the wall if the

error is bigger than desired, away from it otherwise). Intuitively, such a controller will make the robot continuously go closer and farther from the wall, in other words oscillating (in a way progressively less "intense") around the optimal distance until this is eventually reached. Given these considerations it might therefore be interesting at this stage to discuss about the problem of "damping". A system is said to be properly damped if it does not oscillate out of control, meaning its oscillations are either completely avoided (which happens very rarely in any real-life application) or they gradually decrease towards the desired state within a reasonable period of time.

Derivative control (D)

Derivative control is intended to fix the oscillation problem implicit in proportional control. To do so, a derivative controller operates on the momentum of the system by controlling its velocity. As the system approaches the desired state, an amount proportional to the velocity (the "derivative term", calculated as $(gain * velocity)$) is subtracted in the calculation of the required correction, thus compensating for the momentum of the system as it nears the desired state.

A derivative controller generates an output o proportional to the derivative of its input i according to Equation 2.2 (where K_d is again a task-specific proportionality constant, not necessarily set on the same value as in Equation 2.1 [233]).

$$o = K_d \frac{di}{dt} \tag{2.2}$$

Integral Control (I)

Integral control offers an additional improvement over P and D control introducing the so-called "integral term". The idea is that the system is able to keep track of its own errors (in particular those repeatable, fixed errors called "steady state errors") and behave accordingly. The systems does so by summing up these incremental errors over time until the sum reaches a certain threshold decided by the designer. Once this happens the system performs a certain operation in order to compensate for the error accumulated. An integral controller produces an output o which is

proportional to the integral of its i input as for Equation 2.3 (where K_f is our usual proportionality constant [233]).

$$o = K_f \int i(t)dt \tag{2.3}$$

It should be highlighted how integral control cannot be applied to all systems, but only to those in which steady state errors can build up. For example, the controller of the wall-following robot we discussed earlier would not be a valid candidate. A better example would come from a lawn-mowing robot operating inside a rectangular garden. The robot covers a straight line and, when it reaches the edge of the garden it turns by a 90° angle, moves forward, and then rotates an additional 90° to get to the next strip of grass. If there is a steady error lying in the rotation process (*e.g.* the robot consistently turns 85° rather than 90°) the lawn-mowing operation will not be performed correctly. But if the robot has a way of measuring its error, it can apply integral control to recalibrate itself.

PD and PID control

PD and PDI controllers, often seen in real-world applications, are simply combinations of the types of control described above. Specifically, PD control is a combination (sum) of proportional (P) and derivative (D) control terms which can be formalised as for Equation 2.4 [233].

$$o = K_p i + K_d \frac{di}{dt} \tag{2.4}$$

The very same thing can be said about PID control, in which proportional (P), integral (I), and derivative (D) control terms are mixed together as it can be seen in Equation 2.5 [233].

$$o = K_p i + K_f \int i(t)dt + K_d \frac{di}{dt} \tag{2.5}$$

2.2.2 Feedforward (open loop) control

The reason why feedback control is also called "closed loop control" is that the controller stays between the input and the output, effectively closing a circuit which goes on as long as required (a "loop" indeed). The relationship between these three elements is easily visible in Figure 2.8.

Although, a different control paradigm is possible. One of the most popular alternatives to feedback control consists in feedforward, or also "open loop" control. In feedforward control there is no feedback and the state of the system is not fed back into it. The resulting "open loop", despite being a useful metaphor, is not even a loop indeed. Rather than looking at the current state of the system and adjusting it accordingly to the distance from a specific desired state, in open loop control the system makes a certain number of predictions. In order to decide how to act in advance, the controller determines set points (or sub-goals) for itself ahead of time, *i.e.* it predicts the series of states that it should go through in order to satisfy the various sub-goals in sequence, and, eventually, the overall goal.

As the reader must certainly have noticed, we have not dedicated a huge amount of time to describe open loop control. The reason lies in the fact that, although interesting, this approach to the design of robots controllers tends to work effectively if the reference environment is predictable and does not change in a meaningful way. Unfortunately, these conditions are as far from the truth as they can be. Furthermore, the evolutionary neural network controllers that we will see at work in most of this thesis can be classified as particular types of feedback controllers [214].

2.3 Neural Networks (NNs)

The neural networks field originated in the 1940s, contemporarily to the birth of Artificial Intelligence, when McCulloch and Pitts proposed the idea of an artificial neuron [239]. Since the beginning, neural networks have played a significant role on the AI research. In this section we will take a closer look at this domain. The basic mathematical treatment provided is based upon Freeman & Skapura's [124],

and Gurney's [147] books.

2.3.1 McCulloch and Pitts' artificial neuron (TLU)

The model introduced by McCulloch and Pitts (sketched in Figure 2.9), which permits implementation on a digital machine, is characterised by three elements: one or more inputs, an internal activation function, and one output. McCulloch-Pitts' neurons can either be in an "on" or "off" state, and are switched on in response to input stimulations that are high enough in terms of intensity to exceed a specific threshold. The existence of a threshold led to define this computational model as a Threshold Logic Unit (TLU[9]).

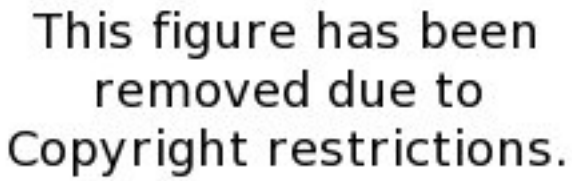


Figure 2.9: Graphical representation of a McCulloch-Pitts' neuron. Source: [147]

We label the output of the neuron (*i.e.* its activation, or the value generated by its activation function based on the input received) with the symbol y, while the value of the threshold used is represented with the θ symbol.

When McCulloch and Pitts introduced the model of the artificial neuron, they were using the real brain as a model, thus implying that neurons should not be working in isolation, but rather connected to others, thus forming networks of neurons (neuronal or neural networks). The connections are modelled through so called

[9]As we will see later, a threshold function is not the only possible activation function for an artificial neuron. Though, the McCulloch-Pitts' model explicitly mentioned on/off neurons.

"synaptic links" each of which having a numerical value associated to it representing the "strength" (or "weight") of the connection. Usually, the output y of one given neuron constitutes either the output of the entire network or the input to a different neuron. The overall contribution to the activation of one specific neuron comes from the sum of all of its input values (x_i), modulated through the weights (w_i) of the synaptic links carrying the signals. The activation value a for the neuron can then be calculated according to Equation 2.6, where n corresponds to the number of input connections.

$$a = \sum_{i=1}^{n} w_i x_i \tag{2.6}$$

As we mentioned before, the use of a binary threshold as activation function implies that the output y of the neuron (on/off, or 1/0) varies depending on whether the threshold has been reached or not. In mathematical terms, this can be expressed as in Equation 2.7.

$$y = \begin{cases} 1, & \text{if } a \geq \theta \\ 0, & \text{if } a < \theta \end{cases} \tag{2.7}$$

2.3.2 Hebbian learning

One of the first algorithms to be developed to make a neural network learn how to perform a certain task was inspired by the work done by Donald Hebb. In 1949, describing his hypothesis about "synaptic plasticity[10]", the scientist stated [161]:

> "*When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*"

Nowadays this quote is often stated in a simpler form as, "*neurons that fire together wire together*". According to Hebb, the existence of synaptic plasticity within animal and human brains is what allows for associative learning to be achieved.

[10]We define as "synaptic plasticity" the ability of the connection, or synapse, between two neurons to change in strength in response to either the use or disuse of transmission over synaptic pathways.

This form of learning can also be seen at work in artificial neural networks. When applied within this context Hebb's principle dictates that the modification of a synaptic weight w_{ij}, connecting neuron i to j, are dictated based upon the similarity between the activation levels of the two units. The magnitude of the modification to be applied is a topic which Hebb did not cover in his work. Later mathematical formulations of his hypothesis have led to several equations. An example, which includes a "learning rate" η (a parameter included between 0 and 1, which modulates the "speed" at which the learning is achieved), can be seen in Equation 2.8.

$$\Delta w_{ij} = \eta ij \tag{2.8}$$

A classical and often cited example (*e.g.* in [124]) of Hebbian learning in neural networks focuses on classical conditioning, using the familiar experiment of Pavlov [292]. Figure 2.10 is a representation of a neural network composed by two input neurons and one output neuron, respectively representing a sound input, a sight input, and a salivation (output) signal.

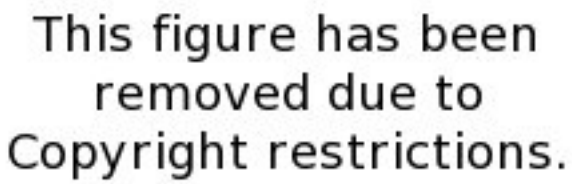


Figure 2.10: A neural network capable of achieving classical conditioning learning. Source: [124]

Assume that the excitation of neuron C, caused by the sight of food, is sufficient to activate B, which is the neuron generating a salivary response. In the absence of further stimulations, the ringing of a bell excites A, but its level of activations

is not enough to make B fire. If we generate the sound stimulus at the same time the food is shown, A is now participating in the excitation of B. The Hebbian rule will increase, time after time, the strength of the synaptic link connecting A to B, making possible for A to activate B acting alone (*i.e.*, in absence of stimulus coming from C.)

From a more general point of view, Hebbian learning has several particularly interesting features. First, it is an instance of an unsupervised learning procedure; second, it is a local learning rule, meaning that it can be applied to a network in parallel; third, it is simple and therefore requires very little computation; fourth, it is biologically plausible.

2.3.3 The properties of TLU networks: a classification example

What a TLU does is to separate its input patterns into two categories according to its binary response (0 or 1) to each pattern. Looking from a geometric perspective, these two categories can be seen as two different regions in a multidimensional space, separated by a straight line or a plane (or a higher dimensional equivalent). If we consider TLUs dealing with input patterns composed of two components only (x_1 and x_2, see Figure 2.11) we might represent these patterns in a two dimensional space, as shown in Figure 2.12, where each pattern determines a point in this so called "pattern space."

A simple TLU like the one seen in Figure 2.11 can easily compute the most common logic functions. Table 2.1 shows for example the activation/output table for a TLU implementing a logic AND (the weights w_1 and w_2 are supposed to be equal to 1). From a geometrical perspective, what the TLU does in such an example is to draw a straight line inside the pattern space (Figure 2.12), separating the point of coordinates $(1, 1)$ from the other three points. This is what is called a "linear separation" of classes. In more mathematical terms, the goal of a TLU is to correctly classify a set of externally applied stimuli $x_1, x_2, ...x_n$ into one of two classes, $\mathscr{C}_1$ or $\mathscr{C}_2$, thus working as a linear binary classifier.

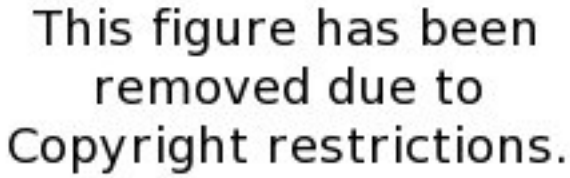


Figure 2.11: A TLU characterised by two inputs (x_1, and x_2), a threshold function, and a y output. Source: [147]

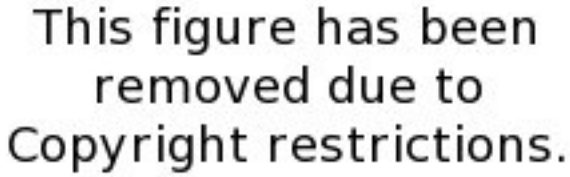


Figure 2.12: Pattern space generated by a two-input TLU. Source: [147]

Table 2.1: Activation/truth table for a two-input TLU implementing a logic AND

x_1	x_2	Activation	Output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

2.3.4 The limitations of perceptrons: dealing with non-linearly separable classes

Soon after the introduction of the McCulloch-Pitts neuron, the research in neural networks concentrated on identifying common properties shared by different network topologies, as well as learning rules that could be applied despite differences between neural architectures.

One of the most popular network models introduced was the "perceptron"‘ first described by Rosenblatt [316, 317], with which the "perceptron training rule" was soon associated. The consequent work of the "perceptron convergence theorem" demonstrated how a perceptron trained this way can always end up with the proper vector of connection weights to correctly classify between two classes of inputs that are linearly separable [318]. A significant extension of the perceptron training rule came from the introduction of the Delta Rule [171, 393], implementing a gradient descent method. The Delta Rule provides several advantages when compared with the perceptron rule. This is particularly true when facing problems that are not linearly separable, *i.e.* for which an optimal solution does not exist. In such situations, the Delta Rule is capable of making the weights vector converge towards a certain value of w_0, which differs minimally from the desired w. The solution is sub-optimal, but is still a solution. The perceptron rule can not deal efficiently with those situations, since it does not stop until the perfect solution is found, thus making the weights vector oscillate between two (or more) alternative states. On the other hand, the Delta Rule implements a never-ending learning process for which the termination condition must be arbitrarily chosen by the experimenter. This may be a trivial task in several domains.

The topic of non-linear separability was analysed in detail by Minsky & Papert in their book "*Perceptrons*" [255], published in 1972[11]. The work carried out by the two researchers constituted a fierce critic to the neural networks field, in particular against the perceptron architecture[12]. The main issue raised through a careful mathematical analysis by the two scientists focused on the impossibility for TLU network architectures to deal with problems that are not linearly separable. To illustrate the first point, Minsky and Papert used the now popular XOR (exclusive-OR) example. The XOR is a logic function for which the truth table shown in Table 2.2 applies.

Table 2.2: Truth table for the XOR function

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Looking at a graphical representation of the problem (Figure 2.13), it is noticeable how the two classes of output (0 and 1) are not linearly separable, *i.e.* they can not be separated through a single line or a plane (or an higher-dimensional equivalent). The XOR constitutes the simplest example of a linearly inseparable problem.

The idea that a single TLU could not implement every Boolean function was a known fact since McCulloch-Pitts' work, as well as the fact that a more complex network architecture, made of a multitude of interconnected neurons, could do the job instead. What Minsky and Papert demonstrated was that such a complex network architecture must be divided into layers in order to work. Layers imply a hierarchy within the network, which can be thought of as being composed of one layer of input neurons[13], one or more layers of intermediate units, and one output

[11] A preliminary version of the book came out in 1969, but missing many details that were added only a few years later.

[12] The term "perceptron" is used herein to indicate single-layer TLUs rather than perceptrons in Rosenblatt's sense.

[13] It is interesting to consider how there is no general agreement within the scientific community about how to "count" the number of layers present into a network. Some authors consider the input units as constituting a specific layer (despite the fact that the input neurons do not have any activation function and the synaptic weights connecting them to the next layer have a fixed 1 value), while others do not.

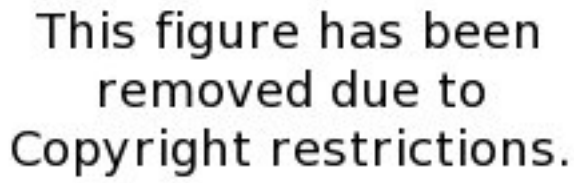


Figure 2.13: The pattern space for the XOR operator. Source: `http://www.morphiles.com/ann/XOR/`

layer (see Figure 2.14 for an example of a three-layer architecture).

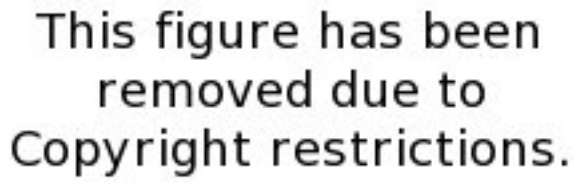


Figure 2.14: Example of a multi-layer NN architecture. Source: `http://nnf.sourceforge.net/nnf1/doc/neuralnetworks.html`

Consequently the problem became the training. We have seen how a network using linear threshold function requires a layer of internal units to solve problems as the XOR. But since there were no algorithms available to train the connections between the input and the internal units at that time[14], a perceptron could not

[14]The underlying problem was the so called "credit assignment problem", *i.e.* how to find a way to assign the proper "credit" to the hidden layer in the determination of the output of a network.

learn how to perform this classification. The XOR problem was chosen specifically because of its simplicity. If a neural network cannot learn such a simple task, how can it be expected to deal with far more complex domains?

Further criticisms raised by Minsky and Papert can be found in the much applauded review on the history of connectionism written by Medler [243].

2.3.5 Multi-layer perceptrons and the backpropagation algorithm

The problem consisting in training multi-layer networks was solved only a few years later, with the introduction of the "error backpropagation" principle by Rumelhart, Hinton, and Williams [328] (though it had first been thought up by Bryson & Ho in 1969 [55]). This principle, consisting in "backpropagating" the adjustments to the connection weights of a network during the training, going from the output to the input layer, was implemented in a supervised learning algorithm that proved to work and being able to overcome the linear separation problem underlined by Minsky and Papert.

We will avoid reporting herein the complete mathematical treatment that made it possible to extend the gradient descent methods to multi-layer networks. However, for the sake of completeness, we list the main resulting formulas.

Equation 2.9 is the general formula which calculates the Δw correction for any node k (either hidden or output) of the network. α represents the usual learning rate, and x^p_{ki} is the i,k-th component of pattern p.

$$\Delta w_{ki} = \alpha \delta^k x^p_{ki} \tag{2.9}$$

The parameter δ^k depends on the node considered. For output nodes δ^k must be calculated according to Equation 2.10, however when the node considered belongs to the hidden layer, the formula to use is expressed in Equation 2.11. Within these two equations, t^p_k is the target output for the k output node, y^p_k is the actual output generated by the network, w_{jk} is the connection weight connecting the j neuron of the hidden layer with the k neuron of the output layer, δ_j is the contribution that

intermediate node j plays on the determination of the output, and I_k is the set of nodes that take an input from the hidden node k.

$$\delta_k = \sigma'(a_k)(t_k^p - y_k^p) \tag{2.10}$$

$$\delta_k = \sigma'(a_k) \sum_{j \in I_k} \delta^j w_{jk} \tag{2.11}$$

In pseudocode form, the backpropagation algorithm can be expressed as in Algorithm 1.

Algorithm 1 Error backpropagation algorithm for a network consisting of a single hidden layer

```
initialise the weights of the network;
repeat
  for each training pattern do
    calculate the error between the network output and the teacher output for
    the current training ptattern;
    compute Δw_h for the weights from the hidden layer to the output layer;
    compute Δw_i for the weights from the input layer to the hidden layer;
    update the weights in the network;
  end for
until the stopping criterion is satisfied;
```

2.3.6 Networks with memory

The introduction of the backpropagation algorithm and with it the awareness that any kind of function could be approximated using neural networks, provided the definitive boost to the field. Researchers started to investigate several different subfields of the NN domain.

John Hopfield was among the first to think in terms of networks supporting a non-unidirectional flow of information. This was the idea behind his "Hopfield network" [167], a neural architecture exploiting recurrences in the internal data flow to replicate phenomena such as "associative memories". This property, which in turn allows for the temporal dimension of information to be implemented in neural networks, has been extensively investigated, after Hopfield's work, resulting in the introduction of two entire families of network architectures, *i.e.* the Simple Recur-

rent Networks (SRNs) and the Recurrent Back-Propagation Networks (RBPs) [237].

The SRNs field originated with the work carried out by Jeffrey Elman eventually published in 1990 [101] with his seminal article "*Finding structure in time*". Elman introduced a simple network architecture (an example of which can be seen in Figure 2.15) made of three layers (input, hidden, and output), plus a set of "context units" (the number of which is the same as the number of neurons contained in the hidden layer). The network is fully connected, meaning that each neuron of one given layer is connected to all the neurons of the next (forward) layer, except for the context units. These are connected via a bidirectional link to the hidden layer of the network, with a full "forward" and a 1-to-1 "backward" connectivity.

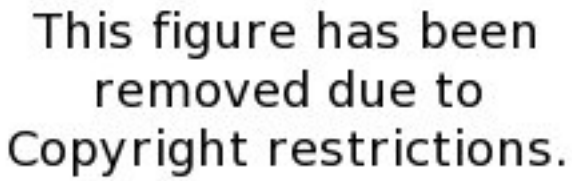


Figure 2.15: Example of an Elman network. Source: `http://wiki.tcl.tk/15206`

The context units provide a representation of the state of the hidden units from the previous time step. Given the input at time t and the activation state of the hidden layer at time $t-1$ (stored in the context units), the entire network can be trained via backpropagation to learn how to predict the next element $(t+1)$ of a sequence which follows a certain generative rule. To reach this goal, the backpropagation algorithm works on the connection weights that go from the input and from the context units to the hidden layer, and to those going from the hidden to the output units. The connections going backward from the hidden layer to the context units are not trained as they are fixed on the value 1, only providing a mean of

"copying" values between the two layers.

Other than those presented by Elman, an additional example of SRN architecture with a recurrent hidden layer can be found in [344].

Another kind of architecture belonging to the SRN family is the Jordan network [182]. Jordan networks are similar to Elman's except for two characteristics[15]. First, the recurrent layer of the network is not the hidden but the output one. Second, the values the context units receive are not a plain copy of those generated by the output layer at the previous step (as in Elman networks), rather the values present in the context units are scaled down by a certain factor and then added to those produced by the output units at time $t-1$.

2.3.7 Activation/transfer functions

We close this section about neural networks by describing the most commonly used activation functions. In this chapter we have mainly discussed TLUs that are characterised by a so-called "step function". Neural networks often rely on more sophisticated transfer functions for their units, as for example the pure linear function (2.12), the piecewise-linear function (as the linear relative, with the only difference being that they only map a subset of the all possible values of x), the sign function (Equation 2.13), the logistic (sigmoid) function (which returns only positive values, as seen in Equation 2.14), and the hyperbolic tangent function (Equation 2.15).

$$y = x \tag{2.12}$$

$$y = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{2.13}$$

$$y = \frac{1}{1 + e^{-x}} \tag{2.14}$$

[15] `http://pyneurgen.sourceforge.net/recurrent.html`

$$y = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.15}$$

All of these basic equations can be modified with the introduction of new terms, working either as offsets (as for Equation 2.12) or as modifiers of the function "steepness" (as for the logistic/sigmoidal and the hyperbolic tangent functions.) Depending on the task the neural network is subject to, any of these activation functions can be useful. During recent years sigmoid and hyperbolic functions have become a default choice among neural network designers, for both supervised and unsupervised learning approaches.

2.4 Evolutionary Computation

This section introduces the field of evolutionary computation and explores its most representative exponents. As this domain is inspired by natural evolution, we will first discuss the evolutionary processes as they take place in nature amongst animal species, and the specific terminology used.

2.4.1 The terminology

The current section aims to clarify the meaning of the terms that will be used when discussing natural and artificial evolution.

Most of the living organisms can be seen, from a biological perspective, as collections of an extremely large number of cells. Each cell contains the same set of one (or more) "*chromosomes*". A chromosome can be divided into "*genes*", functional blocks of DNA, each of which encode a particular protein. To some extent, we can see the genes as "*traits*" of living organisms (e.g. for a human being, the colour of the hair or eyes). Each trait can assume a specific "setting" (continuing with the previous example, "blue", or "black"), which in turn gets the name of "*allele*". Since chromosomes are sequences of genes, a further term can be introduced to identify the location/position of a gene within the sequence as its "*locus*".

The whole collection of genetic materials that an organism is based on is named

"*genome*". "*Genotype*", although sometimes used as synonym for genotype, generally refers to a few specific genetic traits contained inside a genome. During an organism's development, the genotype gives rise to the individual's "*phenotype*", *i.e.* the expression of the genotype in terms of bodily characteristics.

When it comes to evolutionary computation, and to genetic algorithms in particular, according to convention used [257] the term chromosome is typically referred to as a candidate solution to a problem. The genes are blocks of one or more adjacent bits encoding a particular element of the candidate solution and the alleles are the possible values that a gene can assume at each locus. The genotype of an individual in a genetic algorithm is simply the configuration of bits (or different symbols, according to the encoding alphabet used) in that individual's chromosome. Sometimes a genotype has to be converted into its correspondent phenotype in order to be evaluated as a candidate solution for the problem. Often there is no need for such translation, and therefore there is no explicit notion of phenotype at all. All of these aspects will be seen in the section dedicated to genetic algorithms.

2.4.2 On natural/biological evolution

The modern theory of species evolution, still the object of controversy amongst Christian integralists today, dates back to Charles Darwin's "*On the Origins of Species by Means of Natural Selection*" [83].

Inspired by the studies carried out by Malthus on the dynamics of populations [226] and by countless hours spent studying animal breeders, Darwin elaborated a simple but nonetheless detailed framework accounting for the two easily observable phenomena of adaptation (how a population becomes suited to its habitat) and speciation (the arising of new species). The British scientist introduced the concept of "natural selection" as an equivalent (though governed by the nature) of artificial selection. According to Darwin, spontaneous variations that occur within a population of organisms in relation to their traits may cause some individuals to survive and reproduce more successfully than others in their current environ-

ment[16]. In this way, the advantageous traits are transmitted to the offspring and increase their concentration over time. His ideas, despite being less radical than what it is commonly thought were nonetheless shocking, since they were challenging the religion-inspired view according to which animal species are "fixed". Darwin demonstrated, with an abundance of evidence, that species continuously change over time. Later on, Herbert Spencer introduced the definition of "survival of the fittest" [353] to summarise Darwin's hypothesis.

Why changes happen and how they are inherited by the future generations was not clear to Darwin. To some extent - although he disagreed with him on several topics[17] - he was in agreement with Lamarck's theory on "the influence of circumstances" (or "use or disuse") [65], thus implicitly accepting the idea according to which favourable traits can be developed during the life of an individual and subsequently transmitted to the offspring. It is interesting to consider how, despite being a contemporary of Gregor Mendel, father of modern genetics, Darwin developed his hypothesis without any notion of genetic inheritance available.

It was only during the 20th century that the so called "modern evolutionary synthesis" [200] (also known as "neo-Darwinism"[18]) managed to create a link between Darwin's ideas and the knowledge of genetics available at the time. According to this interpretation, natural selection operates on the phenotype of an individual, *i.e.* the observable characteristics of an organism, rather than on its genotype. The genotype is instead considered as the (inheritable) basis of any phenotype and can mutate (sometimes generating variations in the phenotype) because of several reasons, mainly migration between populations (gene flow), and reshuffling of genes through sexual reproduction. Variation also comes from exchanges of genes between different species; for example, through horizontal gene transfer in bacteria, and hybridisation in plants. This view also implies that not every aspect of a phenotype can be transmitted to subsequent generations, as well as the fact that traits acquired in life can not. Furthermore, neo-Darwinism emphasises the role played by

[16]This phenomena is also known today by the name of "differential evolution."

[17]Particularly on Lamarck's hypothesis about the "ladder of complexity" followed by animal species during evolution.

[18]It is interesting to consider how there has been, until few decades ago, a general skepticism in defining these views as "theories", as demonstrated for example by Lovtrup [223].

chance [168], particularly with regard to the genetic drift[19] phenomena.

2.4.3 Evolutionary Computation: an overview

The term Evolutionary Computation refers to a wide range of computer-based approaches that tackle specific problems mimicking what takes place in natural-biological evolution. In their essence, evolutionary computation approaches can be considered optimisation/search methodologies applied to domains that are too complex to be approached with the traditional instruments. As we will see in the following section and in the rest of this thesis, evolutionary computation tools have been and can be applied to countless domains.

The field of evolutionary computation is traditionally considered to be divided into three subcategories, namely Evolution Strategies (ESs), Genetic Algorithms (GAs), and Evolutionary Programming (EP) [118, 257]. Fogel & Fogel [118] explain the reason behind this classification looking at the level in the hierarchy of evolution being modelled: respectively the individual, the chromosome, and the species (see Table 2.3).

Table 2.3: The subfields of Evolutionary Computation and the corresponding level in the evolution hierarchy they model

Subfield	Level in the hierarchy of evolution being modelled
Evolution Strategies	The individual
Genetic Algorithms	The chromosome
Evolutionary Programming	The species

This classification is not universally agreed. Muhlenbein [269] and Goldberg [138], for example, tend to put evolution strategies and evolutionary programming under the umbrella term of Evolutionary Algorithms (EAs), as opposed to genetic algorithms (a point backed up with philosophical reasoning by Fogel [117]). Other authors, such as Back [19] seem instead to be using the term "evolutionary algorithm" as a more general category which refers to every computer algorithm that mimics, in

[19]Genetic drift can be roughly defined as the change in the frequency of a gene variant (allele) in a population.

silico, the mechanisms of biological evolution[20]. Koza and Poli [197] tend instead to consider Evolutionary Programming (and Genetic Programming more specifically) as a subcategory of Genetic Algorithms.

In the next sections GAs will be discussed in more detail, since they play a crucial role in Evolutionary Robotics, while we will provide a brief overview on Evolution Strategies and Evolutionary Programming/Genetic Programming.

2.4.4 Evolution Strategies (ESs)

Evolution strategies (ESs) [39, 40] are the first of the three categories in which we have decided to divide the evolutionary computation field[21]. To some extent and from a historical perspective, despite appearing only few years later and being developed independently, Genetic Algorithms can be seen as an improvement over the earlier experimental evolution strategies algorithms. Thus, the reason for starting our survey about evolutionary computation from the evolution strategies.

The ES field originated with the work carried out by Rechenberg during the 1960s [308], in which he introduced an evolutionary-like (*i.e.* one that was inspired by biological evolution principles) approach to optimising real-valued parameters related to the aerodynamic design of minimal drag bodies in a wind tunnel. Rechenberg's ideas were further refined and developed by Schwefel in the 70s. Originally, evolution strategies (also known at that time as "cybernetic solution paths") were developed as a [40]:

> "*[...] set of rules for the automatic design and analysis of consecutive experiments, [carried out] with stepwise variable adjustments driving a suitable flexible object/system into its optimal state in spite of environmental noise.*"

These rules were just two indeed:

1. change all variables at a time, mostly slightly and at random;

[20]Back also inserted Classifier Systems [133] (CFS) into the evolutionary algorithms family.

[21]Consider for example how Beyer [39] sees evolution strategies as a sub-class of Evolutionary Algorithms, rather than an independent one.

2. if the new set of variables does not diminish the goodness of the device, keep it, otherwise return to the old status.

Rule number 1 resembles random mutation as it happens in nature according to Darwin's theory of evolution. The second rule reverts instead the "survival of the fittest" principle. From these two points comes the reference to evolution. In this basic form, the evolutionary process generates - at each generation - one offspring from one parent. Since the selection takes place among these two, this kind of evolution strategy is referred to as "two-membered evolution", often shortened as $(1+1)-ES$.

Further developments introduced Gaussian distributed mutations in place of the "traditional" binomially distributed ones, mutation rates adapting in real time, and the use of multi-individual (or multimembered) populations. These modifications led to the birth of the so called $(\mu+1)-ES$ evolution strategies (also known as "steady-state ESs") [309]. Later on, Schwefel [342, 341] proposed two additional modifications of the ESs then used, *i.e.* the $(\mu+\lambda)-ES$ and the $(\mu,\lambda)-ES$. In the first one, at any generation not just a single offspring is created, but rather λ (with $\lambda \geq 1$); in order to keep the population size constant, the λ worst individuals out of the entire $\mu+\lambda$ population are discarded. In the $(\mu,\lambda)-ES$ the selection takes place instead among the λ offspring only, with the parents "forgotten" no matter what their fitness is compared to that scored by the new generation.

Since then, countless modifications of the canonical evolution strategies outlined in this section have been proposed, discussed and evaluated. The applicability of ES has also been extended to different fields. Schewefel, for example, participated with Rechenberg and Peter Bienert to the development of FORO 1 (FOrschungsROboter), an evolvable robot [245]. An outdated but nonetheless detailed review on techniques and applications of evolution strategies can be found in Back *et al.* [20].

2.4.5 Genetic Algorithms (GAs)

The most representative examples of evolutionary algorithms are the Genetic Algorithms (GAs) [257, 354], introduced by John Holland during the 1960s [166]. Holland's work was mainly a theoretical one. What he was interested in studying was the phenomenon of adaptation as it occurs in nature, aiming to import it into computer systems. For most of his career Holland kept working on the theoretical foundations of evolution/adaption eventually developing the "schemas framework[22]." Lately - and independently from this mainstream - De Jong [87] demonstrated how GAs could be successfully used for parameter optimisation, thus extending the applicability range of genetic algorithms to countless domains.

GAs mimic species evolution in the sense that they are algorithms relying on two complementary mechanisms similar to those observed in nature: selection (survival) of the fittest individuals, and genetic variation intervening during reproduction. Selection operates based upon a "fitness function", which returns a numerical value (or a relative rank) representing "how fit" a certain individual belonging to a given population is according to the problem investigated (*i.e.* the "environment"). Genetic variation takes place through "genetic operators" instead, *i.e.* mathematical constructs that provide to recombine the chromosomes in the population while moving from one generation to the next one.

As mentioned above, a common use of Genetic Algorithms nowadays is in parameter optimisation/function approximation. In such applications, an initial population is made of randomly generated chromosomes representing candidate solutions (individuals) to the problem tackled. The population is then evolved until a satisfactory solution (*e.g.* an acceptable approximation or set of parameters, evaluated through the use of the fitness function) is identified.

Encoding

The process through which the "informative content" of a solution/individual is translated into a genome is known by the term "encoding". The most popular

[22] `http://en.wikipedia.org/wiki/Holland\%27s_Schema_Theorem`

kind of encoding used in GAs is the binary one, introduced as the default choice by Holland, in which the candidate solutions are encoded as strings of 0 and 1 values. When the original values are integer digits, the encoding can take place as a pure mathematical operation (decimal to binary conversion). Dealing with real values which have a decimal part can make things slightly more complex. A common solution adopted by researchers working in the evolutionary computation community consists of dividing the genome into an integer and a decimal part and dedicating a (fixed) specific amount of bits to each of them. Of course, in order to evaluate the validity of a certain solution (*i.e.* to compute its fitness value) when a binary or a similar encoding has been used, a "decoding" operation is needed. Genetic algorithms do not necessarily require binary genomes, although this kind of encoding allows for a wide range of genetic operators to be used. Although quite rarely used, genomes of real values can nonetheless be found in a significant amount of scientific publications.

Selection methods

Selection is the process which guarantees that the fittest individuals have, on average, a larger amount of descendants compared to the less fit individuals. Various mechanisms (often defined as "schemes") can be used to implement selection. Originally, Holland used a fitness-proportionate selection method in which the expected value (*i.e.* the expected number of times an individual is selected for reproduction) was simply calculated as the fitness of the individual divided by the average fitness of the entire population. Over the years, many more sophisticated selection methods have been introduced. A list including the most popular ones nowadays follows.

- *fitness-proportionate selection*: "modern" fitness-proportionate selection methods are often implemented using the method of "roulette wheel" sampling. According to this methodology each individual is assigned a slice of a circular roulette wheel, the size of the slice being proportional to the individual's fitness (see Figure 2.16).

  The wheel is then rotated N times (where N is the size of the population)

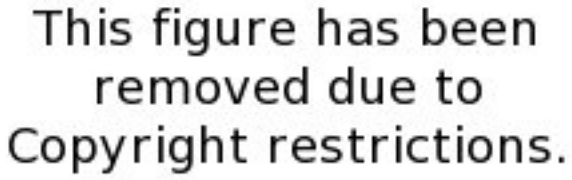


Figure 2.16: Graphical representation of the fitness-proportionate roulette wheel selection scheme. Source: `http://www.edc.ncl.ac.uk/assets/hilite_graphics/rhjan07g02.png`

and every individual on which the marker stops at any spin enters the pool of parents for the next generation. Although this approach, statistically, will lead to every individual having the expected number of offspring, the small population sizes typically used in GAs makes it possible for unlikely spins of the roulette wheel to introduce severe biases in the evolutionary process by selecting not-so-fit individuals. This problem is refered to with the term "spread" (referring to the range of possible actual values given an expected value).

An alternative sampling method developed to cope with the above drawbacks is the "stochastic universal sampling" (SUS) [23]. Where fitness-proportionate selection by means of roulette wheel chooses several solutions from the population by repeated random sampling, SUS uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals. To do so, the imaginary roulette wheel does not use a single marker, rather N of them, equally spaced.

As pointed out by Hancock [148], notwithstanding which sampling method is used fitness-proportionate selection suffers from "premature convergence" due

to scaling problems. Premature convergence [183, 314] is a phenomenon taking place when an evolutionary algorithm gets stuck in a point of local optima of the search space, without being able to reach the gloabl optima, *i.e.* the best solution that the algorithm could potentially find (we will get back on this point in a following subsection of this paragraph). This problem can be partially mitigated in different ways. For example adopting scaling methods that map "raw" fitness values to expected values. A classical example of scaling method is the "sigma-scaling" [121, 137] which helps to keep the selection pressure (*i.e.* the degree to which highly fit individuals are allowed many offspring) high and pretty much constant over time. This goal is achieved calculating individual' expected value as a function of several parameters, specifically its fitness, the average population fitness and the standard deviation. Another way to address this issue is by using methods that vary the selection pressure according to the evolutionary dynamics. An example of such an approach is the "Boltzmann selection", which allows the less fit individuals to reproduce at a similar rate to the one assigned to fitter individuals during the early stages of the evolution in order to mantain a certain level of variation within the population until a certain stage is reached;

- *rank selection*: although from some points of view it might somewhat be considered just another variation of the fitness-proportionate selection created to prevent premature convergence, rank selection [22] works in a slightly different way. In rank selection there is no need for scaling, as the individuals are ranked based on their fitness values. The expected value for each individual is calculated according to its rank. Making the absolute fitness information disappear, ranking helps in reducing the selection pressure when the fitness variance is high;

- *tournament selection*: both fitness-proportionate and rank selection methods are quite expensive in computational terms, as they require passing several times through the entire population performing different kinds of computations. From this point of view, tournament selection [139] is a significantly

more efficient solution. According to this selection scheme two individuals are chosen at random from the population and a random value r, uniformly distributed between 0 and 1, is calculated. Depending on whether this value is above or below an arbitrary threshold fixed by the experimenter, one of the two individuals is selected to be a parent. After the instance of the "tournament" the two individuals selected are brought back into the original population to be, potentially, selected again;

- *steady-state selection*: the selection schemes we have seen so far operate within the context of algorithms that tend to recreate entirely new populations at every generation. When steady-state selection [380] is used, only a few individuals are selected. These are the least fit individuals that are going to be replaced. This procedure provides several advantages in specific domains, as for example when evolving rule-based or classifier systems (see [133]).

Elitism [87], a simple tweak in the evolutionary algorithms that forces it to preserve unmodified the fittest individual at any given generation, is a commonly adopted modification to the selection operators described above.

An empirical comparison between alternative selection schemes is the subject of the extensive work carried out by Hancock [148].

Interestingly enough, in the scientific literature there is disagreement concerning whether to consider the selection a "genetic operator" or not. In this thesis we have decided to consider selection and elitism to be two separate operators, and "genetic operators" those (listed in next subparagraph) that actively produce modifications on the individuals' genomes.

Genetic operators

One of the most significant innovations introduced by Holland on top of the work on evolutionary computation (evolutionary strategies) previously carried out by Rechenberg [309] consists in using both multi-individual populations and genetic operators (as the crossover we will see next) working on multiple individuals at once. Genetic operators operate when moving from one generation to the subsequent one.

They work modifying the genome of certain individuals mimicking phenomena that occur in nature (random mutation, genetic recombination, etc.).

The genetic operators used by Holland can be summarised in the following list:

- *mutation*: for one or more genes of the chromosome its/their corresponding allele/s is/are switched to the alternative value;
- *inversion/variation*: the order of a contiguous section of the original chromosome is reversed;
- *crossover*: mimicking genetic material recombination, crossover creates a new genome that inherits from both parents. In essence, subparts of the two starting chromosomes are exchanged around one or multiple "cut-points" (giving rise to "single-point" and "multiple-point" crossover respectively).

As can be easily seen, mutation and inversion/variation operate on single individuals, while crossover requires the genomes coming from two parents. Figure 2.17 shows mutation, inversion/variation and single-point crossover in action.

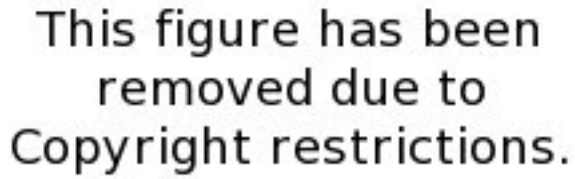


Figure 2.17: Graphical examples of how mutation, inversion/variation, and single-point crossover genetic operators work on binary genomes. Source: `http://bioinformatics.istge.it/bcd/Curric/ProtEn/111.html`

Every genetic operator is typically associated with a certain probability of being used during reproduction. In the rest of this thesis we will refer to these proba-

bilities as: p_c for the crossover operator; p_m for the mutation operator; p_i for the inversion/variation operator.

Since the binary one is the most frequently kind of encoding used, most of the genetic operators elaborated over the last decades have been designed to cope with such genomes. On the other hand, despite having been introduced by Holland in his seminal work, the inversion/variation operator finds little (if any) usage in today's GAs. Although the operators that have been described herein have been assumed to be dealing with binary genomes, some of these have been applied to genomes encoded in alternative ways. An example consists in the crossover operator used by Montana and David [262], which works on real-valued genomes.

Parameter tuning

A crucial decision to make in implementing a genetic algorithm consists in how to set the values for the various parameters, such as the size of the population and the rates associated to the genetic operators employed. This task is everything but easy, as the interaction between the various parameters is nonlinear, meaning that they cannot be optimised one at a time. Furthermore, a GA relying on a certain set of parameters might work well in a certain scenario, but quite badly when applied to a different domain instead. This often means that there is little use in looking at the scientific literature in search of inspiration. Literature that, according to Mitchell, does not provide conclusive results on what are the best parameters to use anyway [257].

There has been nonetheless several attempts to identify a universal set of parameters valid for most of the GA applications. One of the first research in this direction is the work done by De Jong [87], who tested different sets of parameters against several benchmark scenarios. His results suggested that generally optimal parameters are: population sizes (N) included between 50 and 100 individuals, single-point crossover applied with $p_c = 0.6$, mutation rate $p_m = 0.0001$ per bit. These settings were widely adopted and used by the GA community for years. Then, more than a decade later, Grefenstette [146] carried out further studies based on a "meta-GA."

As genetic algorithms can be seen as function optimisators, the scientist's idea was to use a GA to optimise the parameters of a different genetic algorithm. The results he obtained advise for the use of smaller population sizes, elitism, and higher operator rates: $N = 30$, $p_c = 0.95$, and $p_m = 0.01$. Schaffer *et al.* [335] ended up with very similar results after having spent over a year of CPU time carefully testing a wide range of parameter's combinations on different problems.

An algorithmic view

All in all, the description of a simple/generalised GA has been provided in a schematic form by Mitchell [257], and can be seen translated in pseudo-code in Algorithm 2.

Algorithm 2 Basic functioning of a GA

```
create a randomly generated population of n l-bit chromosomes;
repeat
  for each chromosome do
    calculate fitness value f(x);
  end for
  select a pair of parent chromosomes from the current population (probability
  of selection being an increasing function of their fitness);
  with probability p_c cross over the pair at a randomly chosen point (chosen with
  uniform probability) to form two offspring;
  mutate the offspring at each locus with probability p_m;
  replace the two parents with the offspring;
  if n/2 ≠ 0 then
    discard at random one of the new population members;
  end if
  replace the current population with the new one;
until a certain number of generations have been generated;
return the chromosome with the highest fitness value;
```

Fitness landscape and search space

We enter now into a brief discussion on the concepts of "fitness landscape", first introduced by Wright back in 1936 [398], and "search space."

The idea behind fitness landscapes consists in the possibility of graphically representing a set of genotypes/phenotypes and their corresponding likelihood of reproducing (fitness values) in the same plot (an example is shown in Figure 2.18).

Fitness landscapes can be portrayed as N-dimensional graphs (where "N" corresponds to the number of genes for the candidate solution), in which both valleys

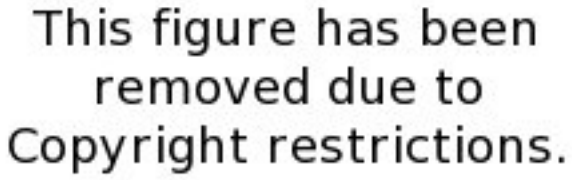


Figure 2.18: Example of a fitness landscape for a two-gene genome. In this case the fitness value is represented on the vertical axis, while the other two axes identify the values the two genes can assume. This landscape is characterised by three peaks and large valley areas. Source: `http://ieatbugsforbreakfast.wordpress.com/2011/03/04/fitness-functions/`

(regions from which most paths lead uphill, *i.e.* towards points with higher fitness) and peaks (points from which all paths are downhill) are present. Their importance in evolutionary computation is in that they provide a useful metaphor to understanding what happens when evolutionary optimisation techniques, as genetic algorithms, are at work. As exemplarily summarised by Mitchell [257]:

> "*According to Wright's formulation, evolution causes populations to move along landscapes in particular ways, and "adaptation" can be seen as the movement towards local peaks.*"

An evolving population typically climbs uphill in the fitness landscape, through progressive modifications of the genomes of their members, until a peak is found. Such peaks can be either "local optima" (*i.e.* points that constitute peaks for a limited region of the space, but that are not the highest peak in the entire landscape) or "global optima" (the highest peak across the landscape).

Genetic operators determine the direction and the magnitude that an evolutionary process takes across the fitness landscape. A typical evolutionary run performed on a multi-individual population starts with the individuals randomly scattered

along the fitness space. Because of the effect played by the selection operator, those located into the valleys tend to disappear, while those closer to the peaks reproduce. The genetic modifications intervening during reproduction give rise to new individuals, deployed into positions close to the ones in which their parents were. Parameters such as the mutation rate (*i.e.* the frequency according to which random mutations in the genome occur) and the crossover rate (*i.e.* the same as the mutation rate, but applied to the crossover operator) influence the distance in the fitness space between the offspring and the parents (an interesting discussion about the role played by the crossover operator can be found in [258]). A trade-off is generally required. The evolutionary process can reach a local optima point reasonably fast if high application rates are used for the genetic operators, but at the same time it will be extremely unlikely to reach a global optimum or ending up in a stable state in which the genomes of the entire population converge to a similar one. On the other hand, using rates that are too low can make the evolutionary process extremely time consuming and still does not guarantee the identification of a global optimum[23].

The wider the variety of variables constituting a genome, the more the dimensions that a fitness landscape can assume, quickly becoming unrepresentable from a graphical perspective. Nonetheless, they preserve their explicative content as metaphors of what happens during computer-based evolution.

Fitness functions

With regard to the role played by the fitness function, the structure of the fitness formula (also "objective function"), which strongly depends on the kind of task under examination, is used to shape the fitness landscape. Continuing with the metaphor we are using, we may argue that what we want to achieve when designing a fitness function is a "regular" landscape (intended as the less irregular possible), a kind of environment which is easy to explore for the population, lacking in local points of minima/optima and having a single easily-identifiable optimum. Unfortunately, for problems of significant complexity, it is more or less impossible to predict in

[23]Genetic Algorithms are optimisation methods that do not guarantee to reach points of global optimum, though they guarantee instead that a (local) optimum will be identified.

advance what kind of landscape a certain fitness function would create[24]. Most of the time the experimenter has to go through a trial and error procedure, which is a highly time consuming approach and makes it hard to compare the results coming from different fitness functions. Nonetheless, certain guidelines can be followed. In the context of autonomous robotics, for example, Nolfi and Floreano [280] have proposed the "fitness space" as an objective framework for describing and comparing alternative fitness functions. The fitness space is defined by three "dimensions": 1) functional-behavioural, which determines whether the fitness is based on the control system or on the robot behaviour; 2) explicit-implicit, indicating the number of external variables the fitness function uses; and 3) external-internal, which states to what extent the fitness function's variables are accessible to the control system. A graphical representation can be seen in Figure 2.19.

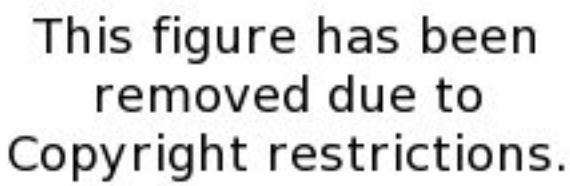


Figure 2.19: Graphical representation of the fitness space. Source: [280]

Lima et al. [215] have analysed the process of designing fitness functions in cost evaluation-based problems. With regard to the subject of this thesis, a recent review specifically focused on fitness functions used in Evolutionary Robotics can be found

[24]As Nolfi and Floreano [280] have interestingly highlighted, "*In artificial evolution the fitness function is used to evaluate the performance of individuals and to select the best ones. The result of an evolutionary run depends very much on the form of this function. Fitness functions for autonomous robots usually include variables and constraints that rate the performance with respect to the expected behaviour, but these variables and constraints are difficult to choose because the behaviour evolved by the robot is not fully known in advance. Actually, the degree of knowledge of an expected behaviour is inversely proportional to the appeal of using artificial evolution.*"

in [276].

2.4.6 Evolutionary Programming (EP)

The Evolutionary Programming (EP) field [118] originated with the pioneering work by Fogel, Owens, and Walsh [120]. The original authors' motivation for evolutionary programming centred on generating an alternative approach to artificial intelligence[25]. Rather than emulating/simulating intelligent beings in terms of neurophysiological structure or particular behaviours exhibited, the idea was to use the natural evolution metaphor as a model to generate artificial organisms of increasing "intellect" over time. From this point of view it is important to consider what Fogel and colleagues meant for "intelligence" and "intelligent behaviour". Their definition of intelligence concerns "*the ability of an organism to achieve goals in a range of environments.*" Consequently, intelligent behaviour is defined as something "*requiring the ability to predict future environmental occurrences coupled with a translation of those predictions into suitable responses.*" [119]

Genetic Programming (GP)

Genetic Programming (GP) [197] is a subcategory of the EP field which is devoted to the evolution of computer programs. Simple instructions, written in source code for a specific computer programming language, are combined together in agreement with a certain set of rules. The resulting programs are evaluated in relation to the execution of a given task, then modified and mixed together, mimicking evolutionary metaphors.

A more formal definition is provided by Koza and Poli [197], according to whom genetic programming is:

> "*[a] domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations.*"

[25] A detailed comparison between evolutionary programming and evolution strategies can be found in [21].

In GP, programs are expressed as "syntax trees" rather than as lines of code. A tree includes "nodes" (also "points") and "links". The nodes indicate the instructions to be executed, while the links indicate the arguments to be provided to each instruction. For this reason, nodes within a tree can be called "functions", while the tree's leaves can be defined as "terminals[26]".

Genetic programming trees can be seen either in a graphical form (see the example in Figure 2.20) or expressed in "prefix notation", with functions always preceding their arguments. An example of such a syntax, also used by some computer programming language as LISP, can be seen in the following example. Supposing there is a function called $max()$ that accepts two input arguments and determines which one is the larger. In a "traditional" infix-notation a call to this function would look like $max(x * x, x + 3 * y)$. In prefix-notation the same function would be expressed ad $(max(*xx)(+x(*3y)))$ instead.

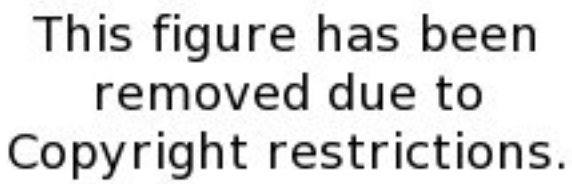


Figure 2.20: Example of a genetic programming tree. Source: `http://en.wikipedia.org/wiki/Genetic_programming`

The designer of the basic version of a GP algorithm must identify a set of terminals and primitive functions for each branch of the program to be evolved (the search space), a fitness function to be used in order to evaluate the performance of any given program (which can be seen as a way to implicitly specify the desired

[26]In more advanced forms of GP, programs can be decomposed in several subroutines. In this case the tree representation consists in a set of trees (one for each component/subroutine), grouped together under a special node called "root." Each subroutine gets the name of "branch" instead.

goal), and a set of conditions for determining how to run and when to stop the evolutionary process. Both the terminals/sets of primitives and the fitness function are highly task-dependent. Nonetheless, the identification of the first two elements is usually a straightforward process, since terminals and primitives are dictated by the characteristics of the task under examination (*e.g.* if the goal is to get genetic programming to automatically program a robot to mop the floor of an empty room, then the designer has to specify the behavioural possibilities of the robots and the possible readings coming from its sensors).

In Genetic Programming the evolutionary process runs in a very similar way to GAs. It starts with a random population of programs for which the individual fitness values are computed. The fittest individuals are selected (on a probabilistic basis) for reproduction and modified through the intervention of genetic operators. As for GAs, the most commonly used operators are mutation and crossover (although applied in a very different way, since in GP these operators must work with trees rather than strings of characters). Architecture-altering operations are also commonly used. A following generation is so created and the process reiterates itself until a termination condition is satisfied.

Classical examples of this approach can be found in Koza's work [195, 196], in which the scientist and his research group presented the results of the evolution of several LISP [356] programs aimed to tackle a wide range of tasks.

Focusing on the domain of evolutionary computation, for "incremental evolution" - as stated by Barlow in [27] - we refer to:

> "*[...] the process of evolving a population on a simple problem and then using the resulting evolved population as a seed to evolve a solution to a related problem of greater complexity.*"

2.5 Incremental evolution

So far we have introduced the field of Evolutionary Robotics and provided an overview of its two main components, *i.e.* neural networks and evolutionary algorithms. According to the "standard" approach to ER, a population is evolved

based on a specific fitness function which evaluates the individuals based on how good they are with respect to the "main" goal they have to achieve. Sometimes, in particular when the final goal is complex, evolution algorithms can fail in evolving the proper solution and get stuck in points of local optima of the fitness landscape. Incremental evolution is an alternative approach to the traditional "direct evolution" methodology which aims to simplify the evolution of complex behaviours by dictating a "path" the evolutionary algorithm should follow.

The inspiration for an incremental approach in artificial evolution clearly derives from biological evolution. Animal species (and this is particularly true for humans) have acquired over the time the ability to perform extremely complex tasks. These abilities were not plucked out of the ether at a certain step along the evolutionary path, rather they have been progressively built up on top of simpler behaviours used as prerequisites. If it is true that these simple behaviours have been sometimes quite evident in their manifestations (*e.g.* in order to learn how to run, the humans have gone through a complex series of sequential steps involving standing on their legs, walking, etc.) however this has not always been the case. Sometimes, in fact, the underlying capabilities needed as prerequisites for the development of more complex behaviours remained "silent" over the time, *i.e.* not expressed in form of explicit behaviours before abruptly appearing. This is what has been demonstrated by Gould introducing the concept of "punctuated equilibrium" [142, 143], a phenomenon that later on has been identified among many other areas outside the biological evolution domain, such as the introduction of government policies [29], and the diffusion of technological innovations [221]. Depending on the fitness function used and the specifications of the problem tackled, both continuous evolution and punctuated equilibrium dynamics (see for example the classic work by Lindgren [218]) can be seen as the result of computer-simulated evolutionary processes. Incremental evolution, by definition, recreates punctuated equilibrium-like dynamics, even though the same sort of phenomena can emerge from direct evolution, especially when complex multi-parameter fitness functions are used.

2.5.1 Incremental evolution in autonomous robotics

The idea of using an incremental approach to evolution is very well known in the autonomous robotics and evolutionary computation fields. Gomez and Miikkulainen [140] described the advantages of incremental evolution in 1997, mentioning, among others, the possibility offered by this approach for evolving behaviours otherwise not obtainable; as well as the better generalisation capabilities exhibited by controllers designed following this paradigm. But the origin of incremental evolution can be dated back even further. Rodney Brooks, taking inspiration from his previous work on behavior-based robotics, was among the firsts to propose an incremental approach to be used within the Genetic Programming domain back in 1991 [52]. This should not be at all surprising when taking into account the fact that Brook's subsumption architecture itself [50] could be easily seen as a way to mimic incremental evolution, building increasingly complex behavioural modules on top of simpler ones. Recognition must therefore be given to Inman Harvey and his research group in Sussex for their studies on the SAGA (Species Adaptation GAs) framework [151]. Thanks to the work of Harvey's group - that could be considered the "father" of the incremental approach to evolution in autonomous robotics - a coherent theoretical framework for incremental evolution has been created. This framework has been used by a significant number of researchers all over the world as a basis for their works.

During more recent times, Mouret and Doncieux [265] have attempted to bring order into the field, providing a classification of the possible approaches to incremental evolution in autonomous robotics in four categories: 1) "staged evolution"; 2) "environmental complexification"; 3) "behavioural decomposition"; and 4) "fitness shaping". Staged evolution employs multiple fitness functions that correspond to multiple sub-tasks of increasingly difficulty; the population is initially evolved to perform the simplest task, then the fitness function is modified leading to the solution of the second task and so on. Environmental complexification is similar to staged evolution, but the complexity of the task can be modified continuously operating on certain parameters. Behavioural decomposition (also known as modular

evolution) relies on the decomposition of a neural controller into separate task-based sub-controllers, each of these is evolved independently from the others. An evolutionary algorithm then combines all of these modules into a master neurocontroller. Finally, fitness shaping uses a weighted sum of multiple evaluation criteria in order to create a fitness gradient that evolution tries to follow.

As from the above classification, incremental evolution does not necessarily involve a progressive complexification of the controller architecture, although this is frequently the case especially when a neural network is employed. Utilisation of this approach is abundant in literature. An example of this is the work by Stanley and Miikkulainen [355], in which they present the NEAT (NeuroEvolution of Augmenting Topologies) method, a framework for evolving neural network topologies along with synaptic weights. The results they have obtained applying NEAT on a reinforcement learning task used as benchmark demonstrate how this approach can outperform those based on fixed neural networks topologies. In Stanley's case, the topology of the network changes over time following evolutionary dynamics. But it is also common that the experimenter decides the topology that the "global" controller must have, manually joining a range of sub-modules dedicated to different functions. Togelius [365] (also reviewed in Tomko & Harvey [368]) provides a further classification based on the possible ways in which different neural modules could be incrementally attached to an existing controller. He defines "incremental evolution" the evolution of a one layer network using multiple fitness functions, "modularised evolution" the evolution of multiple layers or multiple networks with a single fitness function, and "layered evolution" the evolution of a multi-layered network using multiple fitness functions, specific for each layer. On this basis, Tomko and Harvey have pointed out that it is also of fundamental importance to consider how new units/modules are connected to the main controller during incremental evolution. Their findings highlight the detrimental effect generated by the use of random large connection weights, rather suggesting the linkage of additional neural modules using connection weights with zero values.

As reviewed by recent research carried out by Petrovic [295, 296], many works

that can be found within the abundant Evolutionary Robotics literature have employed, to different extents, an incremental approach to evolution. The topics differ widely in their subject matter and range from: the control of unmanned aerial vehicles [27] to 6-legged robots [109], passing through artificial vision systems [153] and autonomous learning [371]. Though most of the published works simply justify the reason for using an evolutionary approach as a consequence of not better specified "issues" in evolving the desired behaviour through direct evolution. A quantitative analysis of the advantages coming from the adoption of an incremental approach is rarely provided. One of the few exceptions to this trend consists in the work carried out by Walker [385], who draws an accurate comparison between the performances generated by a direct and an incremental method for a multi-variable symbolic regression problem. Walker interestingly takes into account the full "computational costs" of both approaches, intended as the number of evaluations of the fitness formula required by the two alternatives. The results he collected demonstrate that no significant advantages in terms of full computational costs are guaranteed by the adoption of an incremental approach. Another recently conducted study, leading to similar evidences, is the one carried out by Christensen and Dorigo [69] comparing the performances generated by two popular approaches to incremental evolution (behavioral decomposition and environmental complexity increase) against the results obtained through several non-incremental evolutionary algorithms. According to their results none of the incremental evolutionary strategies perform any better than the non-incremental methodologies. This stream of criticism seems to also have had an effect on a fierce supporter of the incremental approach, namely Inman Harvey. That in his previously mentioned work states how - according to the analysis carried out - incremental evolution seems to outperform direct evolution only under specific conditions.

In conclusion literature presents results supporting both the arguments, with a recent increase in the number of works that look at the phenomena with a sceptical eye. Anyway, the impression is that it is still extremely difficult to provide a definitive answer to the dilemma of whether incremental evolution is "better" or not than

direct evolution.

Chapter 3

Aerial Robotics and Intelligent Control: History and Technologies

Throughout this chapter we will discuss UAVs (Unmanned Aerial Vehicles, often referred to as Unattended or Unassisted) and MAVs (Micro-unmanned Aerial Vehicles). Instead of the unnecessarily complex (at least for the purposes of this thesis) classification proposed by the US Department of Defense [374], herein we will differentiate between the two categories based on the size of the platform considered, but without relying on a strict criterion for distinguishing between MAVs and UAVs. In general, throughout the following pages, any unmanned aircraft below a four-metres wingspan will be considered a MAV, intended as a subcategory of the more general UAVs class. As a remedy to what we have identified as a factor that is lacking in the literature, we will propose a classification of MAVs in three different categories: Small, Mini and Nano.

The first section will give a quick introduction to the field of unmanned flight from an historical perspective, since its beginning up until modern times. This section offers a military-oriented look to the field of autonomous flight. This focus is justified due to the fact that, until very recently, the major drive towards research in unmanned aerial vehicles has come from the armed forces.

The second section focuses on modern UAVs and MAVs. It begins by outlining the tier classification systems for unmanned aerial vehicles employed by the US Air Force and by the US Marine Corps, and then discusses the main reasons that have

led to a wide adoption of unmanned aerial systems by armies all over the world. The history of miniature aerial vehicles is traced (a review of the most prominent examples of MAVs available today is included in the Appendix) and their typical and potential applications, both from a civilian and a military perspective, are analysed.

The third section focuses on aerial robotics from a more technical point of view. The basic fixed-wing MAV aerodynamics is introduced, as well as the principal challenges involved in designing such aircraft working at low Reynolds numbers. The typical characterisation of robotic MAVs is then introduced, before describing what autopilot systems are and how they work.

Many different platforms will be presented throughout this chapter. Some rotorcraft, either 3 or 4-rotor configurations, will be mentioned although the UAVs/MAVs described herein will be for the most part fixed-wing ones. Rigid and non-rigid airships (see for example Zufferey *et al.* [410]) will not be covered in this chapter because - notwithstanding the interesting scientific challenges they involve - they play a relatively small role with regards to today's military and civilian needs. For the same reason, unusual configurations such as flapping-wings aircraft (see for example Deng *et al.* [92], or the recently introduced Festo SmartBird[1]), aerial vehicles with inflatable and rigidisable wings [376], and hybrid models (as those able to both fly, crawl on the ground and swim [249]) will not be taken into account as well, except in a very marginal way.

3.1 Unmanned flight: a brief history

On 17 December 1903, the Wright brothers, Orville and Wilbur, carried out the first successful heavier-than-air flight test in human history using a powered vehicle. Although they were not the first ones to build and fly experimental aircraft, the Wright brothers were the earliest to invent aircraft controls that made fixed-wing powered flight possible. This was due to the inter-linked roll-yaw control system, extensively tested by the Wright brothers on gliders, carefully described in a recent publication by Padfield and Lawrence [284].

[1] `http://www.festo.com/cms/en_corp/11369.htm`

Following the Wright brothers' success, it did not take long for the domain of unmanned flight to emerge. In reviewing the history of the field, Sullivan [359] mentions the Kettering Bug flying bomb, developed by Charles Kettering in 1918, as the first unmanned vehicle flown by the US Army Signal Corps. Kettering's bug was a gyroscope-controlled flying machine that would fall to earth and explode after the propeller turned a preset number of times. All in all similar to an aircraft, this might also be considered the first missile in military history.

But the Kettering Bug was not an original concept. Four years earlier, in June 1914, Lawrence Sperry[2] [85] - together with his assistant/technician Emil Cachin - carried out a public demonstration of an aircraft whose control surfaces were managed by a rudimentary autopilot system, governed in turn by a gyroscope (a "*Sperry gyroscope*") integrated in the fuselage. The gyroscope was "merely" measuring the error (angle of deviation) between the desired (stable) attitude of the aircraft and the current one, making the necessary adjustments via simple mechanical devices. The demonstration took place in France, during the "*Concours de la Securité en Aéroplane*." Sperry and Cachin made their exhibition as impressive as possible, flying several times in front of the reviewing stand, sitting on the wings of the plane, with no one at the cockpit.

Sperry, inventor of this first autopilot system, quickly became extremely popular, appearing on the front pages of the most important newspapers of the time. Furthermore, despite what was erroneously reported by Sullivan, his research was the inspiration behind the creation of the Kettering Bug as well. The first guided bomb in military history was in fact developed by both Sperry and the automotive inventor Charles Kettering, with external advice provided by James Doolittle. William Scheck's essay [336] on the development of the autopilot narrates the full story in details.

Despite the tremendous success achieved from an engineering perspective, the world was not ready for large-scale unmanned flight. Aircraft were not yet considered a method of mass transportation, instead they were viewed as tools of war or entertainment. No obvious benefits were visible in the use of autopilot systems with

[2]Son of Elmer A. Sperry, the inventor of the gyrocompass.

regard to these two areas. Furthermore, the enthusiastic Lawrence Sperry passed away in December 1923. Without his leadership and feeling the economical pressure exercised by the governments selling the aircraft leftover of WWI, the Sperry Gyroscope Company he was governing in conjunction with his father Elmer (that in the meanwhile managed to develop "universal" autopilot systems thanks to the widespread introduction of the Deperdussin system [4]) did not survive. As a result, the research into unmanned flight consequently stagnated for a couple of decades.

Eventually, in the early 1940s, Sperry's innovations acted as starting points for the birth of the missile field. Several prototypes, as the above mentioned Sperry-Kettering Bug, were developed over the years, but was not until the burst of WWII that they became a common asset in the arsenal of the belligerent armies. The research on this area took place in Germany in particular, and brought to the appearance of the "Vergeltungswaffe-1" (also known as V-1), the first missile (the type of which would be referred to today as a "cruise missile") to be employed in wartime. The V-1 (known among the Britons as "doodlebug" or "buzz bomb", because of its noise) was soon replaced by the technologically impressive V-2 (also known as "Aggregat-4", A4). The V-2 was also the first known human artefact to achieve sub-orbital spaceflight.

On the Allies side, the only attempt to use unmanned aircraft carried out during the second world war was the highly unsuccessful Operation Aphrodite [187]. Operation Aphrodite attempted to use manned vehicles (namely B-17s and PB4Ys bombers) as unmanned ones. Stripped of their standard equipment and loaded with several tons of explosive instead, the Allies were planning to use them against fortified Axis' defences, but none of them actually managed to hit their designated target.

3.1.1 The drives for the UAVs in the military

The rest of the story on unmanned flight, since the end of WWII until modern times, has been reviewed by Sullivan [359] from an interesting perspective. Rather than focusing on trends in technical development of pilotless aircraft, Sullivan has

identified four main ideas driving the use of unmanned aerial vehicles in the military. Although recently a strong interest has started to grow within the scientific community also, the military field has always been the most significant source of innovation in pilotless aircraft. Thus, the story of unmanned aerial vehicles is closely linked to military history.

The four drives identified by Sullivan are:

- *force multiplication*: a constant drive in military history responds to the "do more with less" logic. This does not simply mean providing more "power" to smaller groups, rather accomplishing more with that group than what could have been done previously;

- *strategic bombing*: started during the Spanish civil war in 1936-39, the practice of strategic bombing became widely accepted (and used) during the second World War, as becoming a standard tactic during every following conflict;

- *better intelligence, search and reconnaissance*: since the first battle in history was fought, gathering information on enemy troops and fortifications is considered a dangerous but extremely important task. The importance of intelligence further increased over the last few decades, given the nature of modern warfare scenarios that rarely see two or more conventional armies facing each other on an open battlefield;

- *battlefield of the future*: any military planner's work focuses on imagining "the next war." This consideration alone is enough to justify investigations and financial investments in any sort of cutting edge technology that could be used for military purposes.

What follows is a list of some more detailed examples related to Sullivan's points.

Concerning force multiplication, the aforementioned Aphrodite project demonstrated the need for more precise application of force during WWII. The recognition of this need indirectly started a stream of investigations along the direction of cruise missiles, weapons significantly more accurate than air-dropped bombs in hitting enemy targets located in difficult to reach positions. More accurate weapons means

more weapons and this is where the concept of force multiplication comes into play. As Davis said [86]:

> "*[t]he accuracy and invulnerability to enemy countermeasures achieved [by American researchers] effectively multiplies the number of missiles we [the US] have now on stands.*"

In 1946 (unofficially at first) the US Navy started the development of the AIM-9 Sidewinder air-to-air missile, a "heat-homing rocket" according to the words of William B. McLean. The AIM-4 Falcon, designed by the US Air Force, quickly followed. Time was ready for the introduction of AGM (air-to-ground) missiles, with AGM-45 Shrike and AGM-65 Maverick being two of the most prominent examples of the category. The AGM-28 Hound Dog was the first prototype of a cruise missile[3] instead. Worried by the improvements in SAM (surface-to-air) counter air missile technologies exhibited by the Soviet Union (that could have significantly reduced the impact of a nuclear deterrence mainly based on bombers), American scientists started to investigate alternative carriers for their nuclear warheads. The solution that was found, consisting in the usage of cruise missiles, subsequently led to the appearance of intercontinental ballistic missiles (ICBMs).

The second point raised by Sullivan concerns strategic bombing. This kind of military operation can be performed nowadays by several different means, employing both manned or unmanned aircraft bombers, as well as missiles with various degrees of autonomy. Roughly three families of tools suitable for this purpose can be identified, each of them with its own advantages and disadvantages: bombers, cruise missiles and ICBMs. Bombers are the most flexible solution, since they are flown directly by a human pilot, but at the same time they are the slowest and the biggest (in terms of size) amongst these tools, thus constituting a relatively easy target for enemies' air defences. Among the options that have been discussed, ICBMs are the most autonomous instrument, as they fly independently from lift-off to target under their own guidance systems. Their effectiveness is extremely high, but on the other hand their programming requires time and must be done well in advance

[3]We define as cruise missile a "*SSM surface-to-surface guided missile that carries an explosive payload and is propelled, usually by a jet engine, towards a land-based or sea-based target.*"

before the mission they are intended to take part in. Cruise missiles guarantee a mix of flexibility (provided by the fact they can be fired from mobile platforms) and effectiveness (in terms of penetration inside the enemy's airspace,) without excelling in any of these two dimensions. The X-45 UCAV[4], for which the development was started by Boeing in 1998, is a technological attempt to combine the flexibility of a manned aircraft with the penetration and range of an ICBM.

Sullivan then introduces among the drives identified the set of tasks constituted by intelligence, search, and reconnaissance (ISR). Considering that, while not being dangerous[5], collecting intelligence information is an extremely repetitive and boring task, employing automatic systems (as unmanned aircraft) always seemed to be a natural way to go. Firebee Q-2A, developed by Ryan Aeronautical, was a jet-powered, air-launched, remotely piloted and expendable UAV designed to gather information over hostile areas. Operative since 1951, the Firebee is the most widely used UAV family in military history. An example of the Firebee's use is during the Vietnam War. In this conflict, the Firebee aircraft (the model 147 Lighting Bug particularly) flew over 3,400 missions. Its design was improved over time, allowing the aircraft to become increasingly more autonomous in its sorties, not necessarily depending on the C-130 bombers usually used for the deployment. Some of the Firebee models also acquired strike capabilities, as the AQM-34 version, operational since 1976.

This trend has not stopped yet. Aiming to "maintain global awareness" [358], in 1998 the US Air Force took control over the High-Altitude Endurance (HAE) UAV Advanced Concept Technology Demonstration (ACTD) programme[6], resulting in the development of the Northrop Grumman (the new name for the former Ryan Aeronautical company) Global Hawk UAV. The Global Hawk, one of the most advanced UAVs to date, is an extremely interesting aerial platform, since it provides up to 42 hours of endurance and can operate safely in adverse weather conditions. Its most prominent competitor in terms of popularity is certainly the General Atomics

[4] "UCAV" is the acronym for "Unmanned Combat Aerial Vehicle", *i.e.* a UAV with combat capabilities.

[5] Both in terms of human and political costs. See for example the issues that arose during the Cuban Missile Crisis.

[6] `http://www.fas.org/spp/military/docops/defense/actd_mp/HAE.htm`

Predator. Although the two aircraft have demonstrated how they can be jointly used in a successful way, the Global Hawk, thanks to its significantly superior capabilities for intelligence operations, has become the default choice for intelligence operations, while the Predator is quickly evolving into a widely appreciated combat vehicle.

Finally, Sullivan mentions the "battlefield of the future." As we have seen, UAV technologies emerged from the research into missiles carried out during the First World War. At that time, it was thought that "flying bombs" or "aerial torpedoes" would have been main players in the upcoming warfare scenarios. These developments did not result in any practical application until much later, though. But they eventually did. The same applies to unmanned aircraft. Particularly during the Vietnam War, they demonstrated their efficiency in penetrating dense enemy air defences, but it took long before they became accepted as a standard intelligence/-combat tool. Now, thanks to the constant technological march towards smaller components, the research is focused on new classes of miniature UAVs, which is the domain we refer to across this thesis.

3.2 UAVs and MAVs

On the basis of the brief historical background traced in the previous section, we now jump forward to modern times and focus on todays' UAVs and MAVs, as well as their most typical application areas.

3.2.1 Unmanned Aerial Vehicles (UAVs)

According to a widely accepted definition published in the US Department of Defence Dictionary of Military and Associated Terms [374], a UAV can be considered:

> "*A powered aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable [...]*"

Since the above definition might include expendable devices also, many authors have decided to adopt slightly different denotations in order to distinguish between aircraft and missiles. Sullivan [359], for example, uses the acronym UAV to refer

to "*any reusable air vehicle that does not have a pilot on board*," while defining a missile as a "*one-time use vehicle with no pilot on board.*" Beside disputes about the definitions, it is a fact that missiles and unmanned aircraft have several things in common, as their shared development history demonstrates.

The Tier systems

According to the military point of view, future UAV systems are intended to be employed alongside soldiers not as mere tools in their hands, but rather as autonomous systems capable of performing operations without human supervision. Although, at the moment UAVs are integrated with the other components of an armed force. The integration scheme between these various elements is described in terms of a "Tier" system, and it is used by military planners to designate the various individual aircraft elements in an overall usage plan for integrated operations. The Tiers do not refer to specific models of aircraft, rather to roles for which various models are intended. The U.S. Air Force and the U.S. Marine Corps each have their own tier classification system. Unfortunately these two systems are not integrated.

For what concerns the tier system used by the USAF, it relies on three broad categories of Tier I, Tier II or Tier III, with sub-categories such as Tier II Plus and Tier III Minus [403]. Tier I refers to low altitude and long endurance vehicles, Tier II to medium altitude and long endurance (MALE) vehicles, Tier II+ to high altitude and long endurance (HALE) "conventional" UAVs, Tier III- to high altitude and long endurance low-observable aerial systems. Lax and Sutherland [205] have elaborated on this UAVs categorisation also focusing on the general capabilities of control range and speed, thus producing the resume reproduced herein (see Table 3.1, which has been further modified in this place).

The US Marine Corps (USMC) tier system [271] is also divided into three classes, mainly in function of: the kind of missions the UAV can take part in, who should assume the control over it, what the operating radius and the payload supported are (see Table 3.2).

Table 3.1: USAF UAVs tier classification system, after Lax & Sutherland's modifications [205]

Category	Designation	Max Alt	Radius	Speed	Endurance	Examples
Tier I	Low altitude, long endurance	Up to 15,000 ft	Up to 250 km	60-100 kts	5-24 hrs	RQ-2Pioneer, Searcher, GNAT 750
Tier II	Medium altitude, long endurance (MALE)	3,000 ft to 15,000 ft	Up to 900 km	70 kts	Over 24 hrs	MQ-1 Predator, MQ-9 Reaper
Tier II Plus	High altitude, (long) endurance (HAE, or HALE)	Up to 65,000 ft	Up to 5,000 km	350 kts	Up to 42 hrs	Global Hawk
Tier III Minus	High altitude, medium endurance, stealth and low-observable characteristics	45,000-65,000 ft max	Up to 800 km	300 kts	Up to 12 hrs	RQ-3 DarkStar

The advantages of UAVs compared to manned aircraft

From a more general point of view - and still specifically focusing on their usage in the military - UAVs can also be classified in three wider families: reconnaissance, combat, and target (see Figure 3.1, re-elaborated by Lax and Sutherland [205]).

Of course, UAVs are not limited to these three kinds of operations only. Midway between the civilian and military fields we have logistics, which can be intended used in both the domains. Furthermore, additional applications are possible within the civilian domain, as for example those related to scientific research. We will get back to military and civilian applications of unmanned aerial vehicles in Section 3.2.3.

Keeping for the moment our focus on a military perspective, the increasing interest shown by various stakeholders towards autonomous aircraft systems is not only justified by economical considerations, although the use of autonomous aircraft instead of traditional manned airplanes would allow institutions to save significant

Table 3.2: USMC UAVs tier classification system

Category	User	Owning unit	Missions	Radius	Payloads
Tier I	Pilot, Command, Battalion	Battalion	SR	2.7 NM	EO/IR/LL
Tier II	Regiment, Battalion, Marine Expeditionary Unit	Regiment, Marine Expeditionary Unit	STAR, Battle Damage Assessment	27 NM	EO/IR/LRF
Tier III	Marine Expeditionary Unit, Marine Expeditionary Brigate, Division, Marine Expeditionary Force	Wing	ISR, Communication relay, Patrolling/Law Enforcement, Battle Damage Assessment	110-240 NM	EO/IR/LD, Communication relay

amounts of money that would normally be assigned to the training/maintaining a crew of pilots. Several reasons for this trend can also be found in what has been pointed out by Cambone *et al.* [62], *i.e.* that human elements are, generally speaking, the limiting factors in performing certain airborne roles. The authors justify this apparently strong view mentioning three examples of applications belonging to the "*dull, dirty or dangerous*" categories. In all of those it is clear how autonomous systems could perform significantly better than manned setups for a wide variety of tasks, both in terms of reliability and accuracy.

The dull factor does not need careful explanation. As we have been taught by industrial history (and, maybe more effectively, by Charlie Chaplin's film "Modern Times") machines are better than humans in performing repetitive and boring tasks without sacrificing effectiveness over time. In 1999, during NATO's intervention in Kosovo, it was not unusual for crews of manned bombers to perform 30-hour roundtrip missions from Missouri to Serbia. This clearly had a strong impact on the operational proficiency of the pilots, making it difficult for the US to perform all the operations they wanted to do (and that could have been done if no human beings were "inside the loop").

The dirty factor can be easily understood by reporting a story that happened

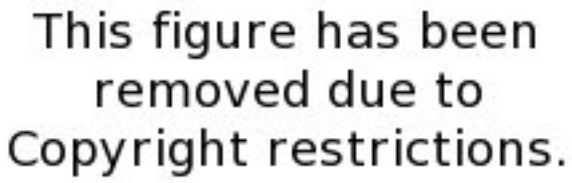


Figure 3.1: Classification of UAVs based on their role. Source: [205]

very recently. During the accident at the Japanese nuclear plant in Fukushima, Global Hawk and U-2 unmanned aircraft were sent over the damaged reactors to monitor the situation[7]. Since one of the first solutions attempted to cool down the damaged reactor (consisting in dropping water from helicopters) failed because of the high levels of radiation threatened the pilots' lives (dirty scenario), one of the alternative hypotheses elaborated involved the usage of unmanned aerial vehicles instead.

Finally, the dangerous factor is related to risks that can be both human and political, but might also be economical. Intelligence operations performed by aircraft have always been extremely risky. As an example, one may just recall that during Vietnam and Israeli-Arab conflicts, the highest loss rate for aircrew and aircraft came from surveillance missions. Political costs are immediately obvious instead when aircrew are captured by the security forces of third-party countries, and are even higher when these aircrew are not involved in "off the books" operations (as it is often the case). To see the possible economical savings offered by the employment of unmanned aircraft it is necessary to look at the entire "life-cycle" that brings an aircraft to the air for a mission. First of all, the crews of manned aircraft must be trained for years before they become operational, with all the associated costs. These

[7] `http://www.nytimes.com/2011/03/19/world/asia/19japan.html?_r=1&pagewanted=all`

costs (despite the fact that the UAV systems commonly used nowadays do not have a 100% autonomy level thus requiring - remote - pilots), are significantly minor for unmanned vehicles. Furthermore the technology behind unmanned aerial vehicles has costs that are not comparable anymore to those required for the development and the maintenance of a "traditional" manned aircraft, and instead are significantly lower. From a mere economical point of view, then, the loss of a manned aircraft and the associated pilots can be considered incomparably more costly than the loss of a UAV. However, there seems nonetheless to exist an interesting paradox between the employment of increasingly cheaper technologies and the fact that most of the modern UAVs are designed to be recoverable (rather than expendable devices as for example the first Firebee flown over Vietnam were). Of course, less expensive designs mean that budget money could be saved and allocated elsewhere, globally improving the financial scenario of the military institution adopting unmanned aerial vehicles. On the other side, the focus on recoverable rather than expendable UAVs risks to deny a proper solution to the "dangerous" factor highlighted by Cambone. If UAVs, whatever the reason might be, can not be considered expendable platforms, then their application range can get very restricted and more or less similar to that typical of manned aircraft. Stated in different words, the goal of not losing human lives (one of the strongest reasons behind UAVs adoption) translates into not losing aircraft, thus limiting the advantages of unmanned systems.

Despite the fact that the current effectiveness of strategic air power is sometimes debatable, military experts agree on the fact that air dominance will still be a crucial factor in the near future. Air forces can not win a war by themselves. The recent Gulf, Kosovo, and Libyan conflicts have clearly shown how air power can make it extremely easy for ground forces to successfully annihilate the enemy, though it is not capable of doing everything on its own. Rather than being a point for undermining the role of air forces, in the eyes of the military leaders, this seems to be an additional reason to predict that air power will not go out of fashion anytime soon. UAVs - attractive because of their capability of keeping combat casualties low and overcoming many human limitations - will definitely be part of future wars.

Ghosh, in a careful analysis on the potential integration of UAV elements into the Indian Air Forces [134], elaborated many interesting additional considerations about military applications of unmanned aerial vehicles. First of all, the researcher stressed how the advantages provided by UAVs are not in terms of reduced loss rates only but - extending the traditional "dull, dirty and dangerous" factors - also involved the major overcoming of human limitations in performing air manoeuvres (as G crunching ones) and affect in a positive way the aircraft designs (not being centred anymore on the safety of the pilot). Despite this, according to Ghosh:

> "*[...] it must be understood that UAVs are not a panacea. Some missions can benefit by the use of UAVs but some others have to be left to manned flights. It is for the air force to determine the correct mix of manned and unmanned aircraft in future battle scenarios.*"

Apart from the considerations above, that so far have lead to a wide adoption of UAVs by all the major military forces all over the world, unmanned aerial systems are subject to a continuous ongoing development. What the air power protagonists are looking for in future UAVs are the following qualities:

- *endurance*: future UAVs must have far greater endurance than manned aircraft, because there is no question of crew fatigue involved. This would allow persistent surveillance over specific territories and continuous deterrence. Since longer shifts imply less aircraft needed, this would lead to a significant reduction of costs for monitoring tasks;

- *wide range of operating abilities*: UAVs should be able to smoothly perform combat tasks, this includes strategic bombings and dogfights. They should also be able to operate over contaminated areas (dirty scenarios) due to the increasing spread of chemical/biological weapons. Furthermore, they could be used, as the need arises, in a provocative role[8];

- *high degree of autonomy*: as they become simpler and easier to operate than manned counterparts due to technological improvements (although this view

[8]This point reminds the operations carried out by Hezbollah in 2004 and 2005, when they invaded the Israeli airspace using a Mirsad-1 UAV. Though this can be seen as a demonstrative rather than merely provocative act, even if Hezbollah claimed that was a retaliation (`http://news.bbc.co.uk/1/hi/world/middle_east/4434505.stm`).

has been challenged in [235]), the costs associated with the training of operating crews are significantly lower for remote controlled UAVs. Further progress on this area is expected (and desirable) in the near future;

- *quick reaction times*: in order to widen the range of operative applications, future UAVs must improve in terms of responsiveness (to reduce their vulnerability and being more proficient in both dogfights and operations behind the enemy lines), in their ability to operate a multitude of weapons, and in the processing of information gathered by multiple onboard and remote sensors.

Modern UAV systems

The UAV systems developed over the last 15 years have been presented in several publications. Among those, one of the first is the review made by Howard & Kaminer [170] in 1995. A more recent, extensive and detailed gallery can be found instead in the "*2005-2030 Unmanned Aircraft Systems Roadmap*" by Cambone *et al.* [62], where the UAVs (or UASs, according to the terminology used by the authors) systems available at the time their research was carried out are carefully described.

3.2.2 Micro-unmanned Aerial Vehicles (MAVs)

Micro-unmanned Aerial/Air Vehicles (MAVs, often defined as MUAVs as well) are in their very essence miniature (in terms of size and weight) aerial platforms. Interestingly enough, their history followed a somewhat different path than the one that has characterised manned aviation.

The origins

In a recent publication, Mueller [267] has outlined the history of micro air vehicles. A history that can be traced back to the early 20th century and more specifically to the development of the first small radio-controlled model airplanes. According to the author, the technology and experience provided by the model airplane community (dating as far back as during the 19th century) served as starting point for the design

of today's MAVs. Moving from the earliest models, rubber-powered or gliders, three important technological innovations progressively made possible the development of powered radio control models: 1) small internal combustion engines (later on replaced by batteries); 2) appropriate sized radio receivers and transmitters; 3) actuators to move the airplane control surfaces.

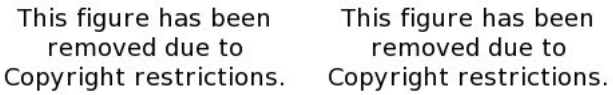
The first reported flight of a RC airplane (a glider with remote rudder control) concerns the exhibition organised by Alfred Lippitsch and Egon Sykora, both from Dresden, Germany, during a competition held in Rohen on May 31, 1936. Later on the same year, Walter and William Good developed a rudder control system and managed to get it working on a gas powered RC model. It took over 20 years, until June 1957, before the first electric powered RC model made its first (recorded) appearance, which happened thanks to the efforts of British Colonel H.J. Taplin. Later on, because of the introduction of nickel-cadmium (Ni-Cd) batteries and small cobalt type electric motors, electric propulsion became the standard for RC aircraft[9] [2]. The United Kingdom, through the work carried out by the newborn RAE (Royal Aircraft Establishment), was also involved in the development of multi-channel proportional radio control, successfully demonstrated in 1952.

Although the improvement of radio-control equipment continued relentlessly, serious interest in the design of small UAVs began in the U.S. during the 1970s, with the NRL (Naval Research Laboratory's) Vehicle Research Section actively involved in demonstrating the feasibility of a non-recoverable MAV for electronic warfare missions. The investigations eventually resulted in the development of the LODED (Long Durance Expendable Decoy) MAV[10], a 1.24m long and with a 1.73m wingspan aerial platform. Despite the disappointing results obtained during the testing of the LODED suggested that further knowledge was required before "useful" MAVs could be developed, this platform proved invaluable as a research tool for identifying the limitations in the technologies then available for unmanned flight and served as a basis for the later development of MAVs such as LAURA [115], SENDER [114], MITE [186], and Dragon Eye [116].

[9] Although it is still possible today to buy off-the-shelf RC models with alternative propeller systems.

[10] `http://www.designation-systems.net/dusrm/app4/loded.html`

In December 1992, in the light of recent innovations in micro-technologies (as micro-mechanical systems and micro-electronic components) DARPA decided that time had come for a second round of serious investigations. In collaboration with RAND Corporation a feasibility study was carried out exploring the possibility of creating a very small, potentially with only $1cm$ wingspan and $1g$ mass, MAV platform [172]. Although such a technological extreme has not been reached yet this event marked the beginning of a flourishing series of MAVs appearing on the scene before the end of the century. Examples of these platforms are the MITE2, characterised by its dual motor design (for a more detailed analysis of MITE2 and its successors see [6]), and the very successful Black Widow [145] manufactured by AeroVironment (see Figure 3.2(a) and 3.2(b)).



(a) (b)

Figure 3.2: (a) NRL MITE2; (b) AeroVironment Black Widow. Sources: (a) `http://www.designation-systems.net/dusrm/app4/mite.html`; (b) `http://www.avinc.com/uas/adc/black_widow/`

The twenty-first century brought many new examples of MAVs to the air. We provide a review of the most prominent examples in appendix A.

Classification

If it is true that distinguishing between a "traditional" aircraft and an unmanned one is often quite a straightforward task, it is significantly more challenging to trace a clear demarcation line between UAVs and MAVs and, within MAVs, between "small", "mini", "micro", "nano", or "palm-sized" vehicles as they are often referred to across the literature. No researchers, as far as the knowledge of the author goes, have provided a clear and easy-to-use classification system yet and most of

the researchers and firms involved in MAVs design seem to adopt their own (flexible and malleable) definitions. This point has also been highlighted by Mueller & DeLaurier [268]:

> "*Although the definition of small UAVs is arbitrary, vehicles with wing spans less than 6 meters and masses less than* $25kg$ *are usually considered to be in this category.*"

The above one is obviously a very generic definition, which could be harmlessly adopted for the purposes of this thesis. Though this would not really be helpful in classifying modern MAVs, as all the examples shown in the next pages, despite the significant differences among them, would be classified as belonging to the same category according to the criterion above.

Herein we propose therefore a classification of MAVs in three families, based upon size (in terms of wingspan) and take-off weight as done by Mueller & DeLaurier, but slightly more detailed and restrictive:

- *Small (S-UAV)*: wingspan less than $4m$, mass less than $26kg$;
- *Mini (M-UAV)*: wingspan less than $1.5m$, mass less than $3.6kg$;
- *Nano (N-UAV)*: palm-sized (wingspan less than $0.4m$), mass less than $0.5kg$.

Although not particularly sophisticated and built around the already existent small UAV models rather than on theoretical basis, the biggest advantage provided by this classification system consists in creating non-ambiguous acronyms, as well as delineating clear boundaries between the different categories. In the rest of this thesis, we will stick to the classification method outlined above, referring to MAVs as the UAVs sub-category including any aircraft capable of autonomous behaviours having a wingspan less than or equal than $4m$ and a mass lesser or equal than $26kg$.

Figure 3.3 graphically displays the boundaries between the above categories (Small, Mini, and Micro UAVs). The circles represent a series of popular Miniature UAV platforms reviewed in the appendix.

Given their handiness, flexibility and the relatively small monetary investment needed to build or acquire platforms belonging to this category, interest in the design

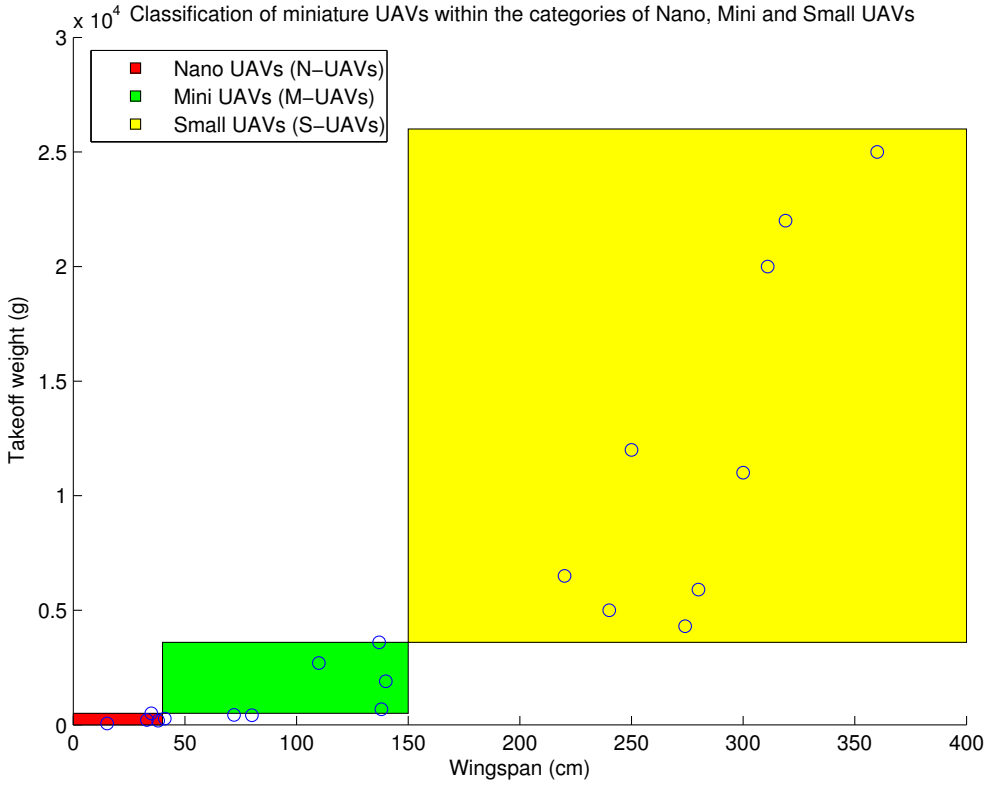


Figure 3.3: Classification of the miniature UAVs reviewed in the appendix, with the three main categories identified highlighted in different colours

and development of small-unmanned air vehicles has increased dramatically in the last twenty-five years [267]. Nowadays MAVs are a quite popular research tool among universities and research centres all over the world, as well as a useful instrument for tasks such as aerial photography in the civilian domain and intelligence gathering in military scenarios.

3.2.3 Applications

Applications of UAVs to military and civilian domains have been reviewed by many authors, as for example in [60, 61, 105, 93]. Since unmanned aerial robots, although not new as a concept, only recently have become an affordable tool from an economic point of view, new applications of this technology are appearing day after day. This section illustrates some of the most significant ones developed so far. At the same time, it examines the role UAVs play in terms of modern warfare needs and how they can also be extremely useful in civilian environments.

Military applications

The most common reasons that have lead to the widespread usage of UAVs and MAVs across armies all over the world have already been covered in previous pages. Here we focus instead on describing some practical applications in order to clarify some of the points introduced above.

Barbara Fletcher, at the Space and Naval Warfare Systems Center Pacific at San Diego (SSC-SD[11]), has elaborated a survey on the role autonomous vehicles can play within modern network-centric battlespaces [110]. The term "net-centric battlespace" is used to identify today's battlespaces, encompassing air, land and sea domains in addition to requiring full sensor coverage and communication across all boundaries. The resulting network of sensors, platforms and communication modes results in a net-centric grid. According to Fletcher, autonomous vehicles are particularly well suited for being applied in the context of C4ISR (Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance) missions. A flow diagram that shows what a typical C4ISR operation consists of can be seen in Figure 3.4.

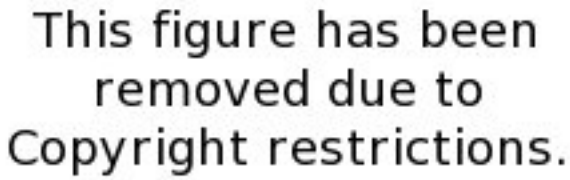


Figure 3.4: Typical action flow for a C4ISR mission. Source: [110]

At the beginning of the cycle is data collection, usually carried out by sensors deployed at the sites of interest, or gathered in alternative ways. Next, the data

[11] `http://www.nosc.mil/robots/`

must be communicated to a central location where it is integrated with other data (data fusion) and interpreted within the context of the situation. Based on such interpretation, decisions are made as to the appropriate actions to be taken. Finally, the action is implemented, often requiring verification, which in turn is provided by sensors, restarting the cycle.

Autonomous vehicles can be used at various stages during this process, and UAVs are no exception. The most natural use of unmanned aerial vehicles involves collecting data. First of all, UAVs can be used both to deploy data sensors over areas of interest. Furthermore, UAVs can work as sensor platforms themselves, thanks to the amount of sensors (cameras, etc.) that they can carry onboard. Other than that, one of the emerging roles for autonomous systems is that of communication relays within the net-centric grid. As with the sensor platform role, use of autonomous vehicles confers multiple advantages. Among these, their extensive reach permits timely communication with remote sites without undue exposure of platforms. Finally, after the data has been collected, communicated to the central nodes, and processed, it is often desirable to be able to act upon it in a timely and appropriate fashion. Autonomous vehicles can be also implementers of the actions elaborated at the end of the cycle. UAVs with strike capabilities (as for example the Predator and the Reaper, to just mention two very popular combat unmanned aircraft) can directly perform a wide array of typical military actions, including firing on a target, intercepting a target, retrieving an object, or neutralising a threat of different kind.

During the last 30 years SSC-SD has developed three large projects involving UAVs as sensor platforms: Airborne Remotely Operated Device [248] (AROD[12], 1982-88), Multipurpose Security and Surveillance Mission Platform [272] (MSSMP[13], 1992-98), and Autonomous UAVs Mission System [270] (AUMS[14], 2002-03).

The AROD project led to the development of two remote-controlled ducted fan Vertical-Take-Off-and-Landing (VTOL) aerial vehicles for short-range aerial surveillance. The first-generation AROD vehicle was electrically powered, with power supplied through a tether from the ground station, and was easily small enough to be

[12] `http://www.nosc.mil/robots/air/arod/arod.html`
[13] `http://www.nosc.mil/robots/air/amgsss/mssmp.html`
[14] `http://www.nosc.mil/robots/air/aums/aums.html`

carried by one person. The second-generation vehicles were much larger and powered by a 26-horsepower, two-stroke gasoline engine, driving a single lifting propeller. Servo driven vanes located at the bottom of AROD controlled vehicle attitude, allowing hover, multi-directional translation, and rotation about its vertical axis. An automatic control system helped maintain vehicle stability. A optic fibre cable provided a communication link to a small Ground Control Unit, with a radio link as backup. A 5 km spool of optical fibre was carried aboard AROD to support a $2km$ round trip or $5km$ one-way mission.

An improvement over the AROD vehicles came few years later with the second project developed at SSC-SD. The Multipurpose Security and Surveillance Mission Platform (MSSMP) system is a distributed network of remote sensors mounted on VTOL mobility platforms plus portable control stations. This sensor package can work either as a portable stand-alone unit or from specifically designed air-mobile platforms.

According to the words of their creators[15], MSSMP was designed to:

> "*[...] provide a rapidly deployable, extended-range surveillance capability for a variety of operations and missions, including: fire control, force protection, tactical security, support to counter-drug and border patrol operations, signal/communications relays, detection and assessment of barriers (i.e. mine fields, tank traps), remote assessment of suspected contaminated areas (i.e. chemical, biological, and nuclear), and even resupply of small quantities of critical items.*"

Finally, the AUMS project was intended for the investigation of technologies for automated launching, landing, refuelling, and rearming of small VTOL air vehicles. The platform used for the experiments was an Allied Aerospace iSTAR UAV, combined with an unmanned ground vehicle used as a transport carrier and to facilitate take-off and landing/recovery operations.

Somewhat related to the second point highlighted by Fletcher (autonomous vehicles as communication relays), UAVs, and particularly small-size models, can also be used to create communication networks in areas struck by natural disaster in order to assist with information exchange among the rescue operators. This possibility

[15] `http://www.nosc.mil/robots/air/amgsss/mssmp.html`

has been investigated by Burdakov and colleagues [56] as well as systematically explored within the SMAVNET project[16] [158, 159], ran at the EPFL in Lausanne, Switzerland.

The third role for UAVs that Fletcher has pointed out concerns autonomous vehicles used as action implementers. The Israeli Air Force was the first military corp that acknowledged the use of an unmanned aerial vehicle (allegedly a IAI Harpy UAV) for a combat operation. That happened in 2006 at the Masna checkpoint in Bekaa valley, during the war against Hezbollah in Lebanon[17]. Many UAV platforms have been upgraded during the last few years to be able to carry warheads and bombs, but their role as implementers is not limited to unloading bombs and missiles on specific targets. As reviewed in a more general way by Ghosh [134] (although discussed in terms of "operational missions"), the range of actions that could be implemented by unmanned (combat) aerial vehicles includes[18]:

- *attacking fixed targets*: UCAVs could be employed to attack high value heavily defended fixed enemy targets, without exposing men to potentially unacceptable losses;

- *attacking moving targets*: missions as interdiction, strategic attacks and close air support typically involve moving targets. UCAVs might be helpful in dealing with these situations as they are capable of loitering for a long time (at high or low altitudes) as they monitor the target in order to predict its movements and identify the right moment in which to attack. Target acquisition using a fixed-wing MAV is a subject investigated by Quigley and colleagues [303];

- *jamming*: when penetrating airspaces protected by radars triggering quick reaction surface-to-air missiles (QRSAMs), it is extremely important to protect the attacking vehicles. Nowadays this is frequently done via "jamming", *i.e.* intentionally emitting radio signals to interfere with the operation of a radar by

[16] `http://lis.epfl.ch/?content=research/projects/SwarmingMAVs/`

[17] `http://www.worldtribune.com/worldtribune/WTARC/2006/me_israel_08_02.html`

[18] Ghosh also included *surveillance/reconnaissance*, and *UAVs as communication nodes* among these missions. They have not been included in the following list as we have already discussed these two topics earlier.

saturating its receiver with noise or false information. This operation can be carried out either by individual fighters/bombers, or by appositely equipped UAVs. Thanks to their endurance, a single UAV might support several strikes;

- *suppression of enemy air defence (SEAD)*: UAVs could also used as receivers in the context of SEAD missions aimed at acquiring air dominance. Loitering over the enemy air space, the unmanned vehicles might be able to pick up emitter data generated by enemy's counter-air defences and pass it to the SEAD network. This data can then be interpreted in order to accurately drive cruise missiles or following bombers missions to neutralise the threat. As the recent NATO intervention in Libya has demonstrated, SEAD operations are frequently the first stage of any large-scale military operation;

- *air-to-air combat*: the dogfight has traditionally been restricted due to human and technological limitations. The human limitation is because pilots can sustain certain levels of "G" force for a limited period of time only. The technological limitation arises because several improvements have been made over the years to stretch this limit as far possible. Without humans piloting the aircraft, UCAVs can be designed to reach degrees of manoeuvrability much higher than those offered by manned aircraft. High "G" and high-speed interceptions would therefore be a definite possibility for UCAVs, while playing both offensive and defensive roles.

Despite the "active" roles that UAVs and UCAVs have played and certainly will be playing in the future, they remain an extraordinary means of information gathering (intelligence). This is not an easy task at all in most of the modern and warmest warfare scenarios, where a regular army typically face insurgent forces not organised in a traditional military way. Valpolini [377] has recently published an interesting survey about the UAV usage in Afghanistan, specifically focusing on the difficulties in collecting (and understanding) intelligence information during an asymmetrical conflict. Maintaining situation awareness during crisis situations and performing early detection of security threats is instead the central topic in the work by Freed

and colleagues [122]. According to their point of view UAVs offer tremendous potential as ISR (Intelligence, Surveillance and Reconnaissance) platforms. With this perspective in mind, the Aeroflightdynamics Directorate (AFDD) of the US Army Research, Development, and Engineering Command, in a joint collaboration with NASA, has developed the Autonomous Rotorcraft Project (ARP). Using a Yamaha RMAX helicopter outfit with a Crossbow IMU, a 900 MHz radio modem, a PC104+ flight computer, a PCI video computer, a sonar, other than differential GPS, vibration and weight-on-wheels sensors. The consortium has investigated topics related to autonomous surveillance such as active/passive obstacle sensing and mapping, route planning around obstacle (*i.e.* obstacle avoidance), and safe landing area determination (relying on an interesting technology based on the JPL Safe Landing Area Determination, SLAD, algorithm [363]).

Monitoring crisis conditions has also been the central research topic for the Navy Research Laboratory (NRL) during the development of the FINDER (Flight Inserted Detector Expendable for Reconnaissance) UAV. The aim of the project, commissioned by the US Defense Threat Reduction Agency (DTRA), was to produce a small aerial platform capable of determining the presence of chemical agents in the air following an attack on a Weapons of Mass Destruction (WMD) facility[19]. The FINDER is not capable of autonomous take-off, thus requiring a UAV to be deployed. Extensive tests have been performed using a specially modified version of the Predator as carrier. The Predator transports two FINDER UAVs on its wing pylons and then releases them to descend to low level and collect air samples. The two "explorers" gather meteorological and chemical data and broadcast it back to the Predator Ground Control Station in order to make it immediately available to the human decision-makers.

So far in this section we have discussed the typical applications in the military field of UAVs. MAVs are also interesting from this perspective, although not so many analyses have been carried out so far on their potential roles. Some general and high-level considerations can be found in [145, 225, 302, 303, 326]. One of the few

[19] `http://www.nrl.navy.mil/research/nrl-review/2003/simulation-computing-modeling/cross/`

studies that specifically deals with the role potentially playable by miniature aerial vehicles from a military perspective (specifically on the roles they might assume in the context of counter-terrorism within the Singapore Armed Forces) is the one published by Chew [67]. The subject will be covered in more detail later on, while describing the potential of the work that has been carried out for this thesis.

Civilian applications

Outside of the military domain, civilian applications of UAVs have touched several different areas.

Since providing timely information on highway traffic conditions for use by a traffic management centre (TMC) is a major function of modern intelligent transportation systems (ITS) this domain has been helped in many different areas by the employment of both UAVs and MAVs. Unmanned aerial vehicles seem to be a natural choice for these sorts of tasks, as satellites available to civilian applications are expensive, have cloud cover restrictions, and cannot provide the temporal and spatial resolution needed for precise traffic and parking analysis. An extensive review of such applications can be found in Anuj Puri's work [300]. Noteworthy amongst them is the ATSS (Airborne Traffic Surveillance System) project[20] [203], ran by the University of Florida in collaboration with the Florida Department of Transportation (FDOT), the Tallahassee Commercial Airport and the University of North Florida Road Weather Information System (RWIS) Research Team. Using an Aerosonde UAV the team intended to collect real-time data about traffic on highways and interurban roads and make this information immediately available to human operators through wireless data transmission.

The Aerosonde UAV was first introduced in 1995 for weather surveillance applications[21]. The plane has a $2.90m$ wingspan, a length of $1.90m$, and weighs slightly more than $13kg$. It can fly continuously for up to 32 hours, with on-board strobe lights that make it visible at night-time. The Aerosonde employs a Sony XC555 colour video camera and a pair of Vaisala RSS901 weather sondes.

[20] `http://www.list.ufl.edu/uav/project.htm`

[21] It is allegedly the first UAV to have flown over the Atlantic Ocean [240].

A proof of concept was elaborated by using two microwave towers managed by the FDOT. The UAV was made to fly over a highway - midway between the two towers - transmitting in real-time to the base stations the video signal recorded by the onboard camera. The two base stations were equipped with video encoder devices that were used to encode the visual stream and transmit it over the FDOT computer network. This information was in turn passed to the State Emergency Operations Center (SEOC), where an automatic system was in charge of identifying the higher quality video and displaying it to the human operators.

Another interesting project is WITAS (Wallenberg Laboratory for Information Technology and Autonomous Systems) [94, 144], coordinated by Linkoping University that ran between 1997 and 2005. The research consortium, in collaboration with Scandicraft Systems, developed the APID Mk III, a 3.63 meter long rotorcraft with a body manufactured using carbon fibre/kevlar sandwich material. The APID Mk III can support a payload of up to $20kg$ (including fuel) and it is endowed with on-board sensors including a radar altimeter, an IR altimeter, a barometer, a compass, and a differential GPS receiver. A 1W radio link is for two-way communication with a ground station. Information from all sensors can be received from the platform and control commands can be sent back.

The core of the project revolves around the vision system installed on the UAV. Digital video cameras are contained in a housing consisting of gyro-stabilised pan-tilt gimbals. Panning, tilt and camera zoom can be controlled from the ground via a separate radio link, or on-board using a specially designed interface. The UAV is intended to navigate autonomously at different altitudes to use its vision system to locate, identify, track and monitor different vehicle types.

Coifman and colleagues [75], at the Ohio State University, have carried out field experiments using a MLB BAT III Mini-UAV[22]. The experimentations involved the monitoring of freeway conditions (for the purpose of observing flows, speeds, densities, off-ramp weaving, turning movements, and vehicle trajectories), intersection movements (with a specific focus on analysing the length of the queues) and network paths (looking for flows, speeds, densities, and vehicle trajectories), and parking lots

[22] `http://spyplanes.com/pdf_new/bat3_brochure.pdf`

(to assess their utilisation).

Another aerial platform for traffic surveillance has been developed at the Georgia Tech Research Institute, in collaboration with the Georgia Department of Transportation and the Federal Highway Administration's Priority Technology Program. Lead by Robert C. Michelson, the program led to the creation of a VTOL traffic surveillance drone capable of relaying live video and two-way audio from the site of traffic incidents, back into the state's Advanced Traffic Management System (ATMS). A military version of this UAV, named Dragon Stalker, was subsequently developed.

The AINS Center for Collaborative Control of Unmanned Vehicles at the University of Berkeley has worked on several topics related to traffic control. Frew *et al.* [126], for example, have developed a computer vision system that enables a UAV to fly autonomously following a road below it in a more accurate way than if only using GPS information. The platform used for the experiments was a modified Sig Rascal R/C plane. Lee *et al.* [208] have focused instead on the development of path-planning strategies for the tracking of a ground vehicle.

The COMETS project[23], coordinated by the Association of Research and Industrial Co-operation of Andalucia (AICIA), aimed to design and implement a distributed control system for cooperative detection and monitoring using heterogeneous UAVs. Both airships and rotary-wing UAVs have been employed for this purpose. Although the project eventually focused mainly on the detection of forest fires as an application of the research carried out by the consortium, among the possible applicative scenarios traffic surveillance was carefully examined. Identified as the most challenging tasks were: to monitor traffic situations; to identify and track individual vehicles; to identify episodic behaviour of both individual and groups of vehicles; to gather data pertaining to road network use and abuse; to provide assistance to emergency services; to serve as a mobile sensory platform with real-time information gathering and processing capabilities.

In addition to those listed above, McCornack *et al.* [238] have elaborated a detailed report about the experiments carried out by the Washington State Transporta-

[23] `http://www.comets-uavs.org/`

tion Center (TRAC), while Harman *et al.* [149], at the Bridgewater State College, have analysed four sub-areas of TDM (Transportation Demand Management) that could be helped by the use of unmanned aerial vehicles.

To some extent comparable to traffic control from a scientific stand point, law enforcement tasks can be performed through UAVs as well. Amidi and colleagues [12], reviewing the work carried out on autonomous helicopters during the 1990s at Carnegie Mellon University, include the following among the goals that can be pursued using aerial robots:

> "*Vision-guided robot helicopters can fly overhead to aid the police in dangerous high-speed chases or criminal search operations. Stationed on top of buildings in urban areas, they can be dispatched in seconds to take off and relay images from trouble spots. This real time imagery is crucial to the tactical assessment of the situation by human experts who dispatch police units to the area.*"

Cristopher Bolkcom examined, for the US Congress, the strengths and limitations of deploying UAVs for a particular kind of law enforcement, namely border surveillance [43]. The same domain has been investigated by Freed and colleagues [122] also comparing the performances of autonomous vs. human control [123].

Other UAV applications within the civilian domain include remote sensing and mapping [105], precision agriculture [163], aerial photography of rangelands [304], photogrammetric recording and documentation of cultural heritage [100], inspection of bridges [247], power utility assets [261] and dams[24], detection of forest [246] and non-forest [193] fires, and oil spill surveillance, detection, and monitoring [8].

A final factor worth mentioning is the role that unmanned aerial vehicles can play in scientific research. Scientific and weather data can be collected via UAVs able to access areas that would be otherwise impossible for humans. The US Geological Survey, for example, has used a small (less than $10kg$) UAV to collect seismic data from the crater on Mount St Helens after the eruption that took place in 2004 [291]. An even more challenging (although unsuccessful) test was carried out by Lin and colleagues [217] in 2000, when they attempted to fly a UAV into typhoon Haiyan in order to collect scientific measurements. The same research group had more luck in

[24] `http://wn.com/Infotron`

2005, when they eventually managed to penetrate a different typhoon (Longwang) using an Aerosonde UAV [216].

3.3 Aerial robotics

With the term "aerial robot" we refer herein to any aerial platform (airplane, helicopter, airships, etc.) capable of executing specific instructions. These instructions can be either provided by a human supervisor, or generated autonomously by the robot. In the latter case, we speak in terms of autonomous control. Autonomous aerial robotics is therefore the science that studies how to design both aerial robots and their associated autonomous controllers.

The focus that will be provided in the next few sections is on fixed-wing MAVs, although many of the topics discussed would fit well if translated to UAVs and "classic" large-sized fixed-wing aircraft. Zufferey *et al.* [411] listed the main characteristics and advantages provided by fixed-wing airplane architectures. According to the authors, this kind of configuration is widespread in robotics due to its simple mechanical design and energetic efficiency when it comes to travelling relatively long distances (a point that was also highlighted by Tennekes [362]). But, side by side with advantages, also come drawbacks and limitations. Quoting Zufferey and colleagues [411]:

> "*The use of no-tail or flying-wing geometries has recently gained a lot of interest in the domain thanks to its mechanical simplicity. However, fixed-wing airplanes dynamics are known to be nonholonomic because their trajectory is mostly defined by the orientation of their main axis. In normal flight regimes, the turn-rate of fixed-wing airplanes is indeed imposed by the inclination around their main axis (i.e. the roll axis) and cannot be changed instantaneously. In addition and contrarily to terrestrial robots, airplanes cannot slow down below a certain velocity known as the stall speed. They are therefore incapable of hovering or moving backwards.*"

As we will see later, these limitations in terms of motion have a strong impact on the design of autonomous controllers for fixed-wing aircraft, especially when the tasks the aerial robots are subject to are not trivial.

3.3.1 Fixed-wing MAVs: basic aerodynamics, design issues, characterisation and control techniques

Fixed-wing aircraft generally rely on three rotation axes, that within the aeronautics field [76] are commonly named as follows: (a) *yaw*, the rotation around the top-down axis, (b) *pitch*, the rotation around the wing-to-wing axis, and (c) *roll*, the rotation around the nose-to-tail axis. Mathematical notation typically uses three symbols to identify these rotations, that are identified as Ψ, θ, and Φ respectively. Figure 3.5 shows the three mentioned rotation axes in relation to an aircraft model.

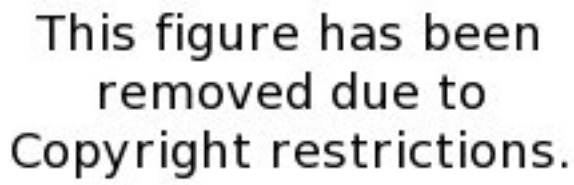


Figure 3.5: Rotation axes for a typical fixed-wing aircraft. Source: [322]

The rotations are controlled through so-called "control surfaces" embedded on the aircraft. The main control surfaces for a fixed-wing aerial vehicle (see Figure 3.6(a)) are generally summarised under the notation "ailerons, elevator, rudder", combining together the names of the three most common control devices used. Specifically, the uses of the three control surfaces are as follows: ailerons control the roll angle, the elevator controls the pitch angle, and the rudder is in charge of the yaw angle.

Figure 3.6 shows where these control surfaces are located in a typical fixed-wing aircraft configuration. It is important to consider how rudder and elevator are generally individual elements, while the ailerons are two (or more) separated items

commanded independently and situated on the two lateral sides of the aircraft[25]. The figure also illustrates the effects generated by the activation of each of these control surfaces on the aircraft. Raising the right aileron and lowering the left one (Figure 3.6(b)) cause the aircraft to roll clockwise (and anti-clockwise when inverting the raising/lowering of the two ailerons, thus the reason for commanding them independently from each other). Raising the elevator (Figure 3.6(c)) makes the airplane pitch up (or down if the elevator is lowered). Finally, moving the rudder (Figure 3.6(d)) left/right causes the aircraft to turn (yaw) left/right in a corresponding way.

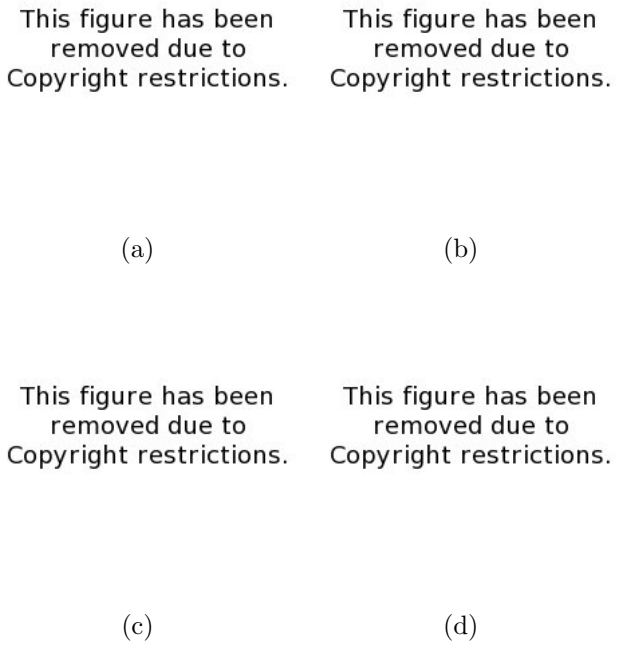


Figure 3.6: Control surfaces for a typical fixed-wing aircraft: (a) birds-eye view of the control surfaces available, plus the throttle; (b) effects generated by the ailerons; (c) effects generated by the elevator; (d) effects generated by the rudder. Source: `http://www.rc-airplane-world.com/rc-airplane-controls.html`

Some authors (*e.g.* Chao [66]) also include the "throttle", the device used to control the motor speed, among the control surfaces. In our opinion the use of the term "control input," rather than "control surface" for the throttle would be more appropriate. Furthermore, it is worth considering how the control surfaces listed

[25]The same sometimes applies to the elevator also. Since, as we will se in the following, the two parts it is made of always behave at the same way, it can be safely considered as a single object anyhow.

above are those "typically" present of a small aircraft, but their presence in that form is not a compulsory requirement to control a fixed-wing aircraft. Ailerons, for example, can often be used as elevators also. When elevators and ailerons are combined together they become "elevons". They look just like elevators but move together when it comes to modify the altitude, as elevators do, and individually, as ailerons do, when the aircraft is intended to steer. In short, one pair of elevons does the job of elevators and ailerons. This solution, despite providing a lesser manoeuvrability, is required in specific fixed-wing aircraft configurations, as the mono-wing ones (see Figure 3.7).

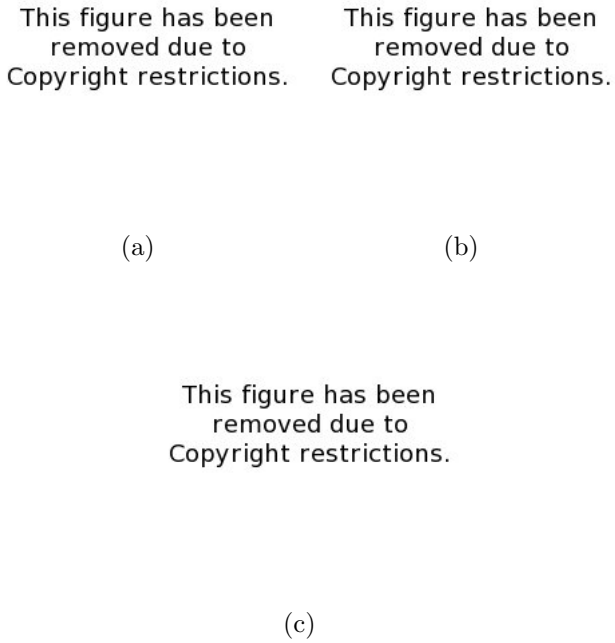


Figure 3.7: Three types of wings commonly used for the design of MAVs: (a) delta wing; (b) swept wing (forward swept or sweptback); (c) straight wing. Source: `http://quest.nasa.gov/aero/planetary/atmospheric/lift1.html`

Although more commonly found in large aircraft, additional control surfaces are the flaps, located on the trailing edge of each wing, between the aileron and fuselage. Flaps are used to generate more lift at slower flying speeds, as well as in order to slow down the airplane when approaching the landing stage. Like elevators, flaps on both wings are operated at the same way (they drop exactly the same amount at the same time). Again, hybrid setups are possible. "Flaperons", for example,

are control surfaces that mix the actions of ailerons with flaps. In other words, one pair of control surfaces along the trailing edge of the wing takes on the job of aileron control and flap control, when needed. On the flip side, "spoilerons" are the opposite version of flaperons, operated by the control surface moving upwards as opposed to flaps that drop down.

The main design issues for miniature flying platforms

The concept and design requirements of miniature flying platforms have been investigated in several publications, most notably by Michelson [250, 251, 252], Mueller [266], and Wu [400, 402].

One of the most important things that the designers of MAV systems have to keep in mind is a very important threshold. Conventional fixed-wing aerodynamics has in fact proven to work well as long as the platform controlled is over 15cm in size, thus staying away from the so-called "low Reynolds number[26] regime" [266, 299].

The main issues arising when dealing with miniature platforms over the 15cm boundary are generally thought to be related to the high density requirements for the energy source, and to the extreme miniaturisation needed for the electro/mechanical components. Michelson [252], investigating this domain in more detail, identified a wider family of aspects of crucial importance related to MAV design: aerodynamics, structure and materials, flight control, morphology, and energy storage/propulsion system. In detail we can see what these aspects refer to:

- *aerodynamics*: working in a low Reynolds number regime implies that aerodynamic characteristics such as the lift-to-drag ratio change dramatically compared to those observable at "standard" Reynolds numbers. The classic aerodynamic analysis methods then break down, often making it compulsory for the designer of MAV systems to test their platforms empirically in order to get an understanding of their actual behaviour. Furthermore, the impact on small flying vehicles by the environmental agents is significant. Watkins and

[26] Reynolds number (Re) is a dimensionless number that relates inertial forces of an object such an airfoil, to viscous forces in a fluid (air).

Vino have carefully analysed the characteristics of the typical turbulent wind environment that birds, insects and MAVs have to face [386];

- *structure and materials*: MAVs require materials that are strong and lightweight at the same time. This is generally not an easy requirement to satisfy. But since the strength of materials does not necessarily scale proportionally to variations in terms of size, materials otherwise unsuitable for aircraft use at "large" scale can become useful at reduced scales instead (as it is the case, for example, for ABS plastic). Another crucial aspect involves the method of control surface actuation. Actuators must be able to move with enough deflection to effect a change in the flow over a control surface while at the same time having sufficient force to work under all flight conditions. Several actuation materials have some of the above characteristics, but often not all of them and furthermore they are not necessarily compatible with the low voltages or currents provided by the (limited) onboard energy source;

- *flight control*: stability and control of MAVs performing in outdoor environments is a highly relevant topic since there is typically not enough power, mass, or control surface area to fight the extremes of the environment. Fixed-wing MAVs suffer particularly in face of roll perturbations, thus roll stability augmentation is often required as well when a pilot on the ground remotely operates them. Developing a system able to maintain the desired attitude while manoeuvring through environmental perturbations is thus the main concern for MAVs designers. On top of this "low-level" system, more advanced trajectory planners can be designed for navigation purposes;

- *morphology*: when it comes to morphology, MAVs usually fall in one of these three categories: 1) fixed-wing, 2) rotary-wing, or 3) flapping-wing configurations. The choice among these configurations largely depends on the purpose the platform is built for. Fixed-wing MAVs can achieve relatively high speeds, but since their design forces them to fly fast at any time, indoor or confined operation is impractical. Rotary-wing MAVs have instead the ability of fly-

ing slowly and also hovering, but the drawbacks consist in their low flight efficiency and duration. While rotor-wing configurations can be used more successfully than fixed-wing ones in indoor environment, the designer still has to protect the MAV from possible rotor strikes. Flapping-wing designs, although not widespread yet because of significant technical limitations, can be seen as a solution that incorporate the benefits of both fixed and rotary-wind configurations, avoiding their most significant drawbacks [249]. A comparative analysis between flapping and fixed-wing configurations, also focusing on the aerodynamics effects generated by the components required to implement these designs on low Reynolds numbers, has been performed by Viieru and colleagues [383]. Alternative design strategies could involve the implementation of flexible wings rather than rigid ones [175]. A comparison between rigid and flexible wing based MAVs can be found in De Luca *et al.* [90];

- *energy storage and propulsion systems*: one of the main current issues in MAVs consists in their limited autonomy. This is something that we see in biology too. Flying insects and birds are constantly challenged by the need for food. Hummingbirds, for example, can ingest nearly three times their body mass in nectar per day, due to their high-energy requirements and to their small bodies not allowing them to store large amounts of food. The same problem applies to MAVs. To complicate things further, current technologies can not guarantee the same efficiency of biological systems and often (as it is the case for batteries) the components used to store energy weigh as much when they are full than when they are empty. Many alternative solutions have been proposed over the years, such as the employment of solar power [315], but this does not seem to be feasible in relation to small MAVs. A recent and promising approach is the so called "energy harvesting" [13, 81, 88]. For what concerns the propulsion systems, the most popular propulsion method for MAVs by far has been the brushless electric motor operating from high energy density batteries or fuel cells. Many drawbacks are nonetheless associated to this technology, leading to the investigation of more energy efficient alternatives, as

micro MEMS gas turbine systems [103] and chemical propulsion systems [249].

Leaving apart considerations strictly related with the $15cm$ threshold, looking at the "micro" aerial vehicles that have been developed over the last few years we can see some common traits among them. Most notably, the distribution of the weight generally follows the 50/20/20/10 rule. This means that 50% of the weight comes from energy storage devices, 20% from the propulsion system, 20% from the airframe, and the remaining 10% (only) constitutes the payload. This highlights the strong impact that propulsion and energy storage systems have on the current MAVs designs. On this basis, Hermans and Decuypere [162] have added to Michelson's list the so called "sensor problem". In essence, the sensor problem stresses the fact that the payload must be extremely light-weight in order to be carried onboard by a MAV. Limited size often means limited functionality, and this is certainly the case when it comes to avionics equipment. High-resolution imaging sensors and appropriate data storage devices available for purchase at the moment, for example, are still too big to be employed on small MAVs, making it practically impossible for them to perform certain kinds of operations.

All in all, the design of small (over 15cm in size) aerial platforms does not present insurmountable issues anyway, as testified by the number of platforms that have been developed. The main problems typically come from the testing phase, rather than the design stage. Williams and Harris [394] discussed in details on the non-obvious problem about how to find the appropriate level of "man-in-the-loop" control during flight-testing. Modelling the developed platform can be a challenging task as well, as demonstrated by the work carried out by Taha and colleagues [361], specifically aimed to collect the most detailed and effective flight data generated in real time by their MAV. Finally, issues related to the usability of human-MAV control interfaces (ground station software) must be considered [127]. As mentioned by Quigley, Goodrich and Beard [302] it is important to provide the end user with an appropriate mix of flexibility and ease of use. Recently, on this subject, studies investigating the employment of multimodal interfaces, with the possible involvements of PDAs/smartphones and tablet computers also, have been carried out [234].

Control techniques

At the beginning of this chapter we have used the term "autopilot" several times. However the definition of this word has not been clarified yet. MAVs can generally rely on two alternative control modes: remote control (also "radio control", RC), and autopilot control. Remote control does not need extensive explanations, as it simply consists of a RC receiver installed on the aircraft and a transmitter operated by a pilot on the ground (see an example of a standard off-the-shelf transmitter in Figure 3.8). The onboard receiver decodes the signal received from the transmitter and operates accordingly the servomotors that in turn drive the aircraft. Transmitters typically work according to a proportional principle[27] and each feature (*e.g.* ailerons control, throttle control, etc.) is conveyed through a specific communication channel.

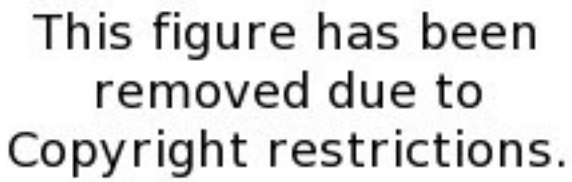


Figure 3.8: Example of a multi-channel transmitter for RC aircraft. Source: `http://www.hooked-on-rc-airplanes.com/rc-airplane-controls.html`

Remote controlled aircraft are, in their essence, 100% human-driven, the only difference in comparison with a "traditional" manned aircraft being in the absence of the pilot onboard. A certain degree of autonomy is obtained when traditional piloting and radio-control techniques are replaced by the use of onboard control

[27]With the term "proportional" the literature typically refers to the fact that a movement performed on the sticks of the remote generates a corresponding (proportional) modification on the configuration of the control surface operated.

architectures, *i.e.* autopilot systems. Ollero and colleagues [283] have reviewed those that are the most commonly used in the field of autonomous aerial robotics[28].

Before entering into details about autopilots, it might be useful to look at the characterisation "problem", *i.e.* the set of variables an autopilot system has (or can have) access to in order to elaborate the proper control strategy.

3.3.2 Characterisation and attitude determination

The characterisation process refers to the identification (and measurement) of the most relevant variables that must be taken into account in order to instruct an autonomous controller to make the controlled robot to perform a specific task. According to Chao and colleagues [66], the most important state variables for a UAV, thus those required for its characterisation, include the following ones:

- p_n: inertial (north) position;
- p_e: inertial (east) position;
- h: altitude;
- u: body frame velocity measured along the body x axis;
- v: body frame velocity measured along the body y axis;
- w: body frame velocity measured along the body z axis;
- Φ: roll angle;
- θ: pitch angle;
- Ψ: yaw angle;
- p: roll rate measured along the body x axis;
- q: pitch rate measured along the body y axis;
- r: yaw rate measured along the body z axis.

[28] Although their survey is mostly focused on helicopters rather than on fixed-wing aircraft.

It is easily surmised that the interactions between these variables are non-linear, as well as non-linear are the effects generated by wind and environment turbulences more in general to the motion dynamics of an aircraft. This high level of non-linearity is the main reason why designing autopilot systems has always be considered a significant engineering effort [227].

Among the variables described above, the absolute roll, pitch, and yaw angles are together referred to as the "attitude", which is the absolute orientation of the aircraft (*i.e.* the relative orientation having the main Earth axis as reference). Attitude is generally represented (although in a restricted manner) on manned aircraft and computer simulators through an Attitude Indicator (AI), also known as "artificial horizon" (AH) or "gyro horizon". The AH is a gyroscope-based device that indicates pitch (fore and aft tilt) and roll (often "bank") angles in an immediately understandable graphical way (see Figure 3.9).

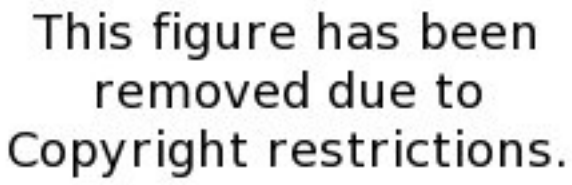


Figure 3.9: An Artificial Horizon (AH) device. Source: `http://en.wikipedia.org/wiki/Attitude_indicator`

The structure of such navigation aid[29] typically comprises of three elements: 1) "miniature wings" (horizontal lines with a dot between them representing the actual wings and nose of the aircraft); 2) a central horizon bar separating the two halves of the display (with the top half usually blue in colour to represent sky and the bottom

[29]It should be noted that Western and Eastern countries have adopted, over the years, different design principles.

half usually dark to represent earth); 3) degree marks representing the bank angle (running along the rim of the dial; on a typical indicator, the first 3 marks on both sides of the centre mark are 10 degrees apart, the next is 60 degrees and the mark in the middle of the dial is 90 degrees).

Artificial horizons are compulsory elements for flights in the so called "instrument meteorological conditions", *i.e.* flights that take place in adverse weather conditions therefore requiring pilots to fly primarily by looking at their instruments, rather than by outside visual references [375]. Many authors have investigated methodologies to determine/estimate the attitude of an aircraft in absence (or failure) of the instrumental reference tools typically used. Cohen and colleagues [74], for example, have carried out studies comparing the performance of GPS-based systems against Inertial Measurement Units. Gebre-Egziabher *et al.* [132] have studied a gyro-free quaternion-based attitude determination system instead.

3.3.3 Autopilots and autonomous control

According to McLean [242] and how reported by Ollero et al [283], the control of fixed-wing aircraft can be considered at different levels. Low-level control is called "stability augmentation" in the airplane control domain; its role consists in managing perturbations and improving the dynamic response of the aircraft when the pilot (or a higher level controller/guidance system) provides commands. On the top of the control hierarchy is the flight path, or trajectory planning. This is a high-level abstraction that ignores the fundamental problems of flying, focusing on merely "piloting" an aircraft through specific paths, waypoints, etc.

To reinforce this point, Michelson wrote [252]:

> "*For autonomous flight, it is common to separate the flight control problem into an inner loop that controls attitude and an outer loop that controls the translational trajectory of the vehicle.*"

Both low and high-level control (*i.e.* inner and outer loops) are possible on MAVs through so-called "autopilot systems" (or autopilots). Autopilots are microelectromechanical systems (MEMS) controlled via software and physically interacting

with the control surfaces of the aircraft. They typically comprise of two parts, a state observer and a controller (thus the name of "state observer-based controllers"). The former provides to collect all the data required for the generation, by the controller, of the manoeuvres that are then implemented mechanically. The most commonly used state observers are micro inertial guidance system that include gyroscopes, acceleration and magnetic (compass-like) sensors. The controller generally consists of an electronic micro-controller, *i.e.* a miniaturised computer embedded on a single integrated circuit.

In terms of control theory, autopilots for MAVs can be seen as closed-loop controllers. Closed-loop controllers, as opposed to open-loop ones (that operate only using the current state of the system and a model of it), rely on feedbacks generated by the dynamical system they govern. The mechanism is somewhat similar to the gradient descent method we have seen for neural network training algorithms. More specifically, the output y of the system at time t is fed back through a sensor measurement F to the reference value $r(t)$ (the reference value is the desired output). The controller C then takes the error (difference) e between the reference and the output to change the inputs to the system under control, thus reacting to the feedback received.

The autopilot of an autonomous aircraft generally answers to two basic needs: state estimation and control inputs generation (based on the reference path and the current state of the system). Figure 3.10 graphically illustrates the functional structure of an autopilot system and the relationships between the various blocks constituting it.

Several techniques can be used for autopilot design. PID (Proportional − Integral − Derivative) based [1] autopilots are possibly the most common ones, but various alternatives, based for example on fuzzy logic [206, 207] or neural networks [289] can be found. A good overview of the main issues involved in the autonomous control of robotics aircraft (namely stabilisation, localisation/navigation, and obstacle avoidance), with a description of the correspondent approaches to their resolution proposed among the literature, can be found in the work by Zufferey

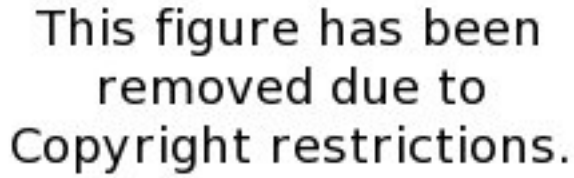


Figure 3.10: Functional structure of an autopilot system. Source: [66]

and colleagues [411].

In addition to the basic control mechanisms described above, an autopilot needs a few additional elements in order to be functioning on an aerial robot. A typical off-the-shelf autopilot for MAVs comprises of an Inertial Navigation System (INS) [28] and an onboard processor[30] to be used both as state estimator (collecting, processing and filtering the information gathered by the sensors) and flight controller. The GPS receiver, also included in the package, is frequently used. Figure 3.11 shows the structure of a typical flight control system for a MAV capable of autonomous flight. When the plane can be controlled either manually or autonomously, the autopilot also requires a communication link with the ground station in order to switch between the different control modes possible (*i.e.* from RC controlled to autonomous and vice versa). This point highlights how, from a software point of view, a complete autopilot system is made of two different parts: the controller software running on the MAV and the one running on the computer(s) used as ground station(s). Figure 3.11 also illustrates the role played within the closed-loop controller by the device dedicated to receiving broadcasts from GPS satellites, information used as part of both the state estimation and, according to the application, to direct

[30]Frequent choices are PC/104 (`http://www.pc104.org/`), Crossbow's Stargate (`http://platformx.sourceforge.net/home.html`), and Gumstix boards (`http://www.gumstix.com/store/catalog/index.php?cPath=27`).

the commands issued by the controller.

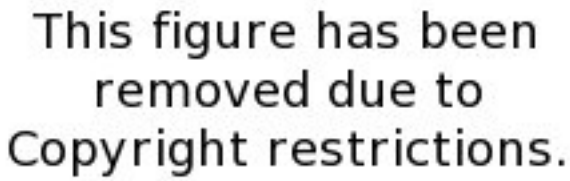


Figure 3.11: A typical flight control system for MAVs. Source: [66]

Thanks to the surveys recently published by Chao and colleagues [66] and Alvis *et al.* [11], we can see reviewed in Appendix B the most common autopilot systems available on the market. The topics of the integration and calibration of such autopilots into physical experimental platforms are not covered within this thesis, but for further reading one could refer to [351] and, although it focuses more on sensors, to [7, 349]. An example of the complete development and integration of an autopilot system on a MAV platform is in [370].

Chapter 4

Distributed Control for Collective Behaviour in MAV Teams: Methodologies, Challenges, Ethical Considerations and Safety Issues

This chapter aims to link the introductory part of this thesis with the experimental component.

In chapter 2 we have introduced the field of Evolutionary Robotics, as well as the two main subfields that constitute it, that are Neural Networks and Evolutionary Algorithms (mainly Genetic Algorithms). Chapter 3 has instead focused on the domain of aerial robotics, with a particular emphasis on fixed-wing Micro Unmanned Aerial Vehicles. What we will describe here is the approach we have proposed to combine the two fields together, having as a target distributed control for collective behaviour in teams of MAVs. Side by side with it, an alternative approach, which does not rely on ER but rather on Reynolds-style flocking algorithms, is also introduced.

In order to present our experimental work, this chapter begins by providing

an overview on the state-of-art computational intelligence approaches to collective behaviour applied to teams of unmanned aerial vehicles. The two approaches we have developed are then introduced and the most significant associated challenges are discussed.

Finally, a plan of the experiments that will be presented in the following chapters is included.

4.1 Intelligent autonomous controllers for collective aerial robots: the scientific literature

Notwithstanding all of the considerations made in the previous chapter, UAVs used nowadays in real applications are still far from being fully autonomous systems. Many of these aircraft can rely on their own guidance systems via which they are able fly without human intervention. The limitation of these systems consists in the fact that, in terms of capabilities, they are not too dissimilar to the automatic pilots used within the civilian aviation domain. They simply provide a means of keeping the aircraft following a given route once they are already in flight. When it comes to aerial vehicles of significant sizes, takeoff and landing, as well as any other non pre-planned manoeuvres, remain operations that have to be physically carried out by human operators. Thus the role of these systems is mainly in relieving the human component from the completion of the most monotonous parts of the missions (although, as mentioned in Section 3.1, sometimes autonomous stabilisation could be enough of a reason for justifying the adoption of an autopilot system for certain kinds of tasks). For smaller UAVs (MAVs and similar), the degree of autonomous control can be slightly higher, mostly because extremely high precision in operations like takeoff and landing is often not a requirement. However, this is rarely reached for most of the aerial robots we can see in action at the moment.

The scientific effort aimed to make this kind of systems (both UAVs and MAVs) fully autonomous is slowed down by both technological issues and ethical considerations. The technology is currently the main limit, since the research in autonomous

aerial robotics can be still considered to be in its infancy. The first practical applications have only appeared during the very last few years, and still have significant room for improvement. Nonetheless, ethical discussions have been constantly arising since then. Of course, these discussions are boosted by the developments that UAVs have been subjected to, which have transformed the early unmanned vehicles, mainly used for intelligence operations, to powerful and "active" war instruments able to fire missiles and engage in a wide range of missions. Furthermore, even if this might well be just a speculation of the author, it is safe to assume that the pilots themselves (even if they are not lobbying against introducing these systems too early because it would make them superfluous for many practical applications) are not pushing for an early introduction of the latest technologies.

During the last few decades the field of autonomous mobile robotics has been tackled from several directions. Several applications of autonomous controllers based on neural networks have been produced for many kinds of vehicles, from wheeled robots [366, 367] to ships [407] and submarines [404, 381, 277, 176]. At the same time, the application of these same design principles to flying robots has not yet been thoroughly investigated. With the only notable exceptions being the systems developed by Buskey *et al.* [57, 58, 59], De Nardi and Holland [89], and Hauert, Leven, Zufferey & Floreano [111, 156, 159, 160], it seems that current approaches to the development of autonomous controllers for aircraft mainly rely on techniques other than neural networks. The design of such controllers via evolutionary paradigms is even less frequently considered. Among the more popular methodologies employed are Behaviour-based robotics [95], Genetic Programming [27, 313], evolution-based path planning [305], modeling field theory [91, 294], and graph search methods [301]. The most significant exception to this trend, as it has been developed around both neural networks and evolutionary methodologies, can be identified in the above mentioned work by De Nardi within the Ultraswarm project[1]. Under the expert supervision of Professor Owen Holland, De Nardi has designed a helicopter capable of indoor autonomous navigation through a series of way-points disseminated in the environment. The helicopter (which flies due to two counter-rotating rotors) is con-

[1] `http://gridswarms.essex.ac.uk/index.html`

trolled by a feed-forward neural network evolved via a genetic algorithm, composed of four separate modules, closely resembling a classical PD controller. The four parts respectively control the longitudinal motion, the lateral motion, the collective (*i.e.* the main rotor), and the yaw. The network input simply consists of a subset of the helicopter state, the vector distance to the next waypoint and the deviation from the reference heading.

There are few publications on collective behaviour in aerial robotics, regardless of the methodology. The field gets even more restricted when it comes to fixed-wing aircraft configurations. Moreover, most of this work consists of theoretical researches based on mathematical or computer models, with little or no experimentations carried out on real robotics platforms. This is not surprising at all considering the issues related to acquiring and carrying out experimentations on aerial robots, that remain significantly stronger than those associated to the more traditional ground robotics field, where experiments can be carried out inside easily accessible and modifiable laboratory environments. Among these few publications, we will use the relevant contribution by Richards and colleagues [313] in order to classify the current approaches to autonomous cooperative UAVs control into four different categories, defined as follows:

- *behaviour-based control systems*: behaviour-based control systems use a network of interacting high-level behaviours to perform a task. Cooperation is achieved through the local interactions between UAVs performing the job;

- *deliberative approach*: focused on developing a specific flight path for each UAV to follow. Such flight paths are rigid and they cannot be altered even in the event that new information is discovered. In other words, the entire scenario is assumed to be fixed and already known in all of its aspects;

- *adaptive replanning approach*: in order to achieve some degree of flexibility, deliberative systems have been improved by incorporating an element of adaptive replanning. As for the deliberative approach, also in adaptive replanning the main role is played by a centralised controller, which generates a specific

flight path for each UAV to follow, based on the currently available information. The UAVs move according to the flight paths received, but they are also able to gather sensorial information from the environment and communicate it to the controller as it becomes available. As the central controller receives new information, it may generate updated flight paths that are in turn broadcasted back to the UAVs;

- *reactive strategies*: rather than generating specific flight paths that require live updates, this approach aims to generate a so-called "reactive strategy" for every UAV to adhere to. This kind of strategy can be thought of as a single decision tree that controls the aircraft for the life of the mission. The decision tree determines changes in the UAV's heading, based on immediate low-level information collected from its sensors.

To shed some light on the general approaches introduced, we briefly introduce in the following pages a few works falling in each of the above categories.

For what concerns behaviour-based systems (see Section 2.1.3), a classic example of autonomous group coordination (in that case among ground-based robots with the purpose of navigation across hazardous environments) comes from the work by Balch & Arkin [24]. The literature abounds in terms of research in behaviour-based robotics, and much of it (*e.g.* [232, 298]) involves collective behaviour. Only a small part, though, focuses on aerial robotics. One of the most relevant exceptions to this trend consists in the work by Schlect *et al.* [338], in which a team of UAVs cooperatively conduct a parallel sweeping search of a geographic area for specific targets (implementing a sort of ASAS, Autonomous Search and Attack System).

The deliberative approach has proven to be highly successful in the civilian domain, but it is often considered too simplistic from a military perspective and, more generally, for tasks of non-trivial complexity. The main assumption underlying this approach consists of having a complete knowledge about the environment and all the relevant variables available to a central controller. It is easy to identify the two main drawbacks of this methodology. First of all, just in very few real-life scenarios (the control tower of an airport is one of these few exceptions) all

the information required is always available and completely reliable. Second, the presence of a central controller in charge of a multitude of vehicles introduces an important criticality into the system, which must count on one or more backup systems to guarantee an acceptable degree of reliability. The approach is nonetheless interesting from a scientific point of view and many remarkable implementations of it can be found in the literature, as those proposed by Abvlasky *et al.* [3] (who stress on the "decomposition problem"), and Sinsley *et al.* [350]. Using a modified version of the SIG Kadet Senior RC aircraft[2], equipped with a Piccolo autopilot (see Appendix B), Sinsley and colleagues have implemented an intelligent controller architecture for collaborative control of multiple MAVs. Collaborative capabilities include formation flight, search of an area, and cooperative investigation of a target. Their work is the best possible example of the deliberative approach in its essence. The cooperation in their system is achieved thanks to a central controller that develops individual flight paths for each UAV to follow.

A prototypical example of adaptive replanning can instead be seen in the control system developed by How *et al.* [169], aimed at controlling a fleet composed of eight ARF 60 MAVs[3]. The coordination algorithm developed (see Figure 4.1) relies on Mixed-Integer Linear Programming (MILP), in combination with a decomposition approach, based on an (improved) heuristic called "petal algorithm", which makes it possible to use such a system in real-time. A number of experiments have been performed by How and colleagues using the control framework they designed, including an extremely simple two-vehicle formation flight with autonomous rendezvous (despite how How and colleagues define "formation flight", this just consists of having two MAVs independently following the same flight path).

The UAV manager concept elaborated by Rathinam, Zennaro, Mark, & Sengupta [306] represents an additional instance of adaptive replanning. They propose an adaptive replanning strategy for teams of UAVs that have to localise enemy targets within a warfare environment. Their approach relies on the presence of a "manager system" interacting with a resource allocator that can redistribute the

[2] `http://www.sigmfg.com/IndexText/SIGRC58ARFB.html`
[3] `http://www.duanesplanes.com.au/product.php?productid=1349`

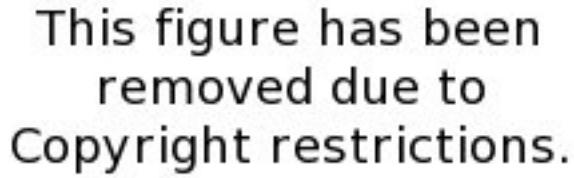


Figure 4.1: Block diagram describing the control system implemented by How *et al.*. Source: [169]

task load among the available resources in case some of them fail (*e.g.* the case of a UAV being destroyed by enemy forces). Cooperative search strategies are the main focus of Vincent & Rubin's work [384] as well. In their case, the aircraft belonging to the team fly in a specific configuration that allows them to optimise their integrated sensing capabilities. Despite the fact that the adaptive replanning approach looks promising, it shares many of the issues generally associated with deliberative strategies. For example to decide when replanning is required, and the amount of time needed to calculate and broadcast the new flight paths to the various UAVs are two non-trivial elements to be taken into account. Scherer *et al.* [337] have recently identified a possible solution using two separate but interacting controllers that respectively act on a global and on a local level ("plan globally and react locally"). Even in this case, a good level of knowledge about the environment is still required.

Vidal *et al.* [382], within the Berkeley's BEAR project [4], have proposed an alternative implementation of the adaptive replanning approach in order to coordinate the behaviours of a hybrid group of robots (both terrestrial and aerial) engaged in a pursuit-evasion game.

Beard *et al.* [31] have introduced an innovative approach to decentralised cooperative surveillance using fixed-wing MAV teams. The way they have decided

[4] `http://robotics.eecs.berkeley.edu/bear/publications.html`

to go falls midway between the deliberative and the adaptive replanning methodologies. The approach they have designed consists of four steps: 1) the definition of the cooperation constraints and objectives; 2) the definition of a coordination variable, intended as the minimal amount of information needed to be exchanged by the aircraft to achieve cooperation; 3) the design of a centralised cooperation strategy; and 4) the use of consensus schemes to transform the centralised strategy into a decentralised algorithm. The interesting part consists in the fact that every individual aircraft elaborates a solution to the cooperation problem and then applies the consensus scheme (point 4) with the other MAVs in order to identify the one that should be adopted by the group.

Furthermore it is interesting to look at the very accurate way in which Beard and his colleagues have defined the concept of group cooperative behaviour [31]:

> "*Group cooperative behaviour implies that individuals in the group share a common objective and act according to the mutual interest of the group. Effective cooperation often requires that individuals coordinate their actions. Coordination can take many forms ranging from staying out of each others' way to directly assisting another individual. In general, group cooperation is facilitated by coordinating the actions of individuals.*"

For what concerns reactive strategies, a classic example can be seen in the work done by Moore [263], who proposes an interesting methodology for solving the Missile Countermeasures Optimisation (MCO) problem under conditions of uncertainty. The MCO problem consists in optimising the manoeuvres of an aircraft to evade incoming Surface-launched Anti-aircraft Missiles (SAMs). Implementing a solution based on genetic programming, Moore designed a system that can take into account uncertainty related to the incoming missile, both in term of type (weight, current speed, expected trajectory, etc.) and current state. Sometimes, the implementation of reactive strategies is aided by the employment of evolutionary methodologies. That is what has been done for example by Richards *et al.* [313], tackling the problem of having a team of UAVs exploring a geographical area in a cooperative way. In their research, it is a formally defined decision tree, evolved through genetic programming methodologies which controls the various aircraft to determine how they should

handle events that could potentially take place. A very similar approach has been followed by Barlow & Oh [26]. They relied on multi-objective genetic programming to design UAV controllers able to autonomously locate and then circle around a radar site. Barlow and colleagues have also tested the evolved controllers on real robots (though they were wheeled robots) in order to prove the robustness of the developed systems.

From a more general point of view it is interesting to look at the contribution by Gancet *et al.* [128], who have proposed a classification system in 5 levels concerning UAVs' decisional autonomy (see Figure 4.2). In their work, they have presented a decision architecture and the associated algorithms for coordinating collective behaviours in multi-MAV systems. The architecture elaborated enables different schemes of "decisioning distribution" in the system, depending on the available decision making capabilities of the single aircraft forming the group and on the operational constraints related to the tasks to achieve.

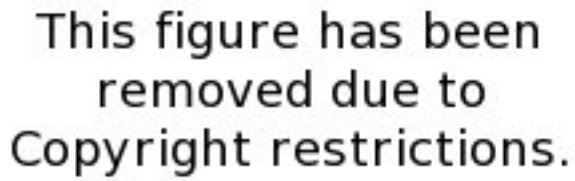


Figure 4.2: Possible levels of decisional autonomy for MAVs involved in cooperative tasks (D: distributed; C: centralised). Source: [128]

What is particularly interesting in the work of Gancet and colleagues is that they employ a hybrid team of MAVs. In one of the tasks studied, for example, they have used at the same time one airship and two helicopters to perform fire monitoring over a certain geographical area.

All in all, to conclude this section and before introducing the experimental part of this thesis, we can state again our starting point. Our belief is that alternative methodologies can be successfully utilised to implement a reactive approach to the distributed control of aerial robots as an alternative to those described herein. One of the two approaches proposed by the author, the one on which most of the effort was spent, relies on Evolutionary Robotics techniques, *i.e.* using neural networks evolved through genetic algorithms in order to design autonomous controllers implementing the desired behaviours. It is worth noting that, though conceptually different, both GP and ER approaches share the compulsory need for computer simulations [78] to be used for designing the controllers. No explicit and pre-planned strategies at a team-level will be developed, since all the aircraft will simply react to the sensorial input they can gather from the environment. The cooperation will emerge spontaneously by merely tuning the rules governing the individual behaviours, and the characteristics and constraints of the task. The second approach proposed is instead based upon flocking algorithms.

4.2 Design methodologies

In this study we propose two different approaches to the distributed control of groups of unmanned aerial vehicles.

The first one relies on a combination of Multi-Agent System (MAS) [389, 397] and Evolutionary Robotics methodologies. The main difference between the product of the procedure we are following and a "standard" reactive strategy approach as described in Richards *et al.* [313] mainly consists in employing a neural network controller instead of a formal decision tree. In both cases the controllers are subjected to an evolutionary process and therefore the use of computer simulators for the training phase is compulsory. The basic principle we have adopted is to some extent similar to the ones proposed by Buskey *et al.* [58, 57] and De Nardi *et al.* [89] for the autonomous control of unmanned helicopters. The controllers we use are in fact neural networks whose outputs affect the heading of the controlled aircraft and its flight direction. However our approach introduces three elements of novelty. The

first is that we aim to study the (simplified) dynamics of fixed-wing aerial vehicles rather than helicopters. Even when employing streamlined simulation models, as those described in this thesis, helicopters are much more flexible in their ability to adjust their movements during the flight when compared to fixed-wing aircraft. If, for example, an unexpected obstacle arises, a helicopter could easily hover overhead, perform a 180 degrees yaw and then look for a different path to follow. When it comes to fixed-wing aircraft, this kind of behaviour is not possible. Therefore the on-line adjustments to the current route need to be extremely accurate (the motion constraints of fixed-wing aircraft are analysed in more detail in Section 4.3.1). The only work where neural networks are applied to the control of aerial vehicles that are not helicopters or airships is the one by Hauert and colleagues [159]. Furthermore, another novelty consists in our decision to implement a basic obstacle avoidance mechanism, which represents an additional challenge to the neural controller. Traditionally, obstacle avoidance behaviour has not been taken into account in studies regarding UAV path planning. As pointed out by Rathbun *et al.* [305], this is mainly due to the fact that UAVs have usually been restricted to operating in areas that do not contain any other vehicles that are not under the control of a supervising authority. Rathbun's work, in which an evolution-based path-planner has to deal with movable and non-accurately estimated obstacles, constitutes one of the few meaningful exceptions to this trend. Finally, the controller we use is made of a single neural network and not of different modules joined together, each of which dedicated to managing different sub-tasks, as in [58, 57, 89]. The entire controller acts therefore as a single entity. Therefore the task of identifying a favourable decomposition of the controller into different dedicated modules is left to the evolutionary process.

The second approach is based upon flocking algorithms. No evolutionary processes and/or neural networks are involved anymore, but simple hand-designed controllers, adhering to the principles described in Sections 4.2.2 and 7.2.3 are used instead.

Both the approaches have required, at a certain stage, the use of a computer simulator. All of the simulators that will be presented in the next chapters have

been designed in accordance with a principle defined as "*incremental geometric flight*". Incremental geometric flight is the topic covered in the next section.

4.2.1 Incremental geometric flight

The term "geometric flight" refers to a computer simulated model of flight[5] in which the involved agents are three-dimensional objects - thus capable of performing rotations around their X, Y, and Z axes - characterised by a "forward direction". The forward direction is arbitrarily chosen by the designer and corresponds to the positive section of one agent's axis. Once the forward direction has been determined, the agent is free to rotate around the remaining two axe, while constantly moving along its heading (which, of course, changes over time because of the effect played by the rotations around the other two axe). The movements happen in discrete time rather than continuously, *i.e.* the model is based on a decomposition of time in small intervals[6]. At any time step the agent moves by a certain distance along its heading direction: for this reason the geometric flight model assumes the adjective "incremental". The distance travelled during each time step depends on the speed at which the agent is moving. Incremental geometric flight allows for conservation of momentum, *i.e.* the agents are generally free to modify their speed, but they are bounded to the maximum accelerations/decelerations they can perform in the time unit, as well as a minimum and a maximum speed they can keep.

Craig Reynolds was amongst the first to formalise the concept of incremental geometric flight. The quotes from Reynolds listed below [311] demonstrate this:

> "*Geometric flight is based on incremental translations along the object's "forward direction", its local positive Z axis. These translations are intermixed with steering-rotations about the local X and Y axes (pitch and yaw), which realign the global orientation of the local Z axis. In real flight, turning and moving happen continuously and simultaneously. Incremental geometric flight is a discrete approximation of this; small linear motions model a continuous curved path.*

[5] Although it could be argued that the same model could apply as well to other domains, *e.g.* underwater schooling, as demonstrated by Lobb [220].

[6] Again, the decision about how long a time step should be is left to the designer. When incremental geometric flight is used to generate animations for movies or similar, the time step length is usually associated with one animation frame.

[...] Geometric flight models conservation of momentum. An object in flight tends to stay in flight. There is a simple model of viscous speed damping, so even if the boid continually accelerates in one direction, it will not exceed a certain maximum speed. A minimum speed can also be specified [but defaults to zero]. A maximum acceleration, expressed as a fraction of the maximum speed, is used to truncate over-anxious requests for acceleration, hence providing for smooth changes of speed and heading. This is a simple model of a creature with a finite amount of available energy."

The simulation models developed by the author and described in this thesis all rely on the incremental geometric flight approach. Sometimes the simulated agents will be constrained in terms of some of the rotations they can perform (*e.g.* in the 2D simulator presented in chapter 5 the agents can only rotate around one axis), whereas at other times they will have access to a slightly wider manoeuvres repository (*e.g.* in some of the configurations tested using the 3D simulator described in chapter 6 the agents can rotate around any of their three axe). In any case, the basis of their motion will always be an incremental geometric flight model.

4.2.2 Flocking

An alternative way to achieve collective behaviour amongst groups of robots consists in relying on the so-called "flocking algorithms". The most prominent example of an algorithm pertaining to this category is the one designed by Craig Reynolds, described in detail in section 7.2.3.

The literature provides only a few theoretical works to use as starting point for research in this direction, pioneered by the research carried out by McLain [241] in the late 1990s. Among these works, Crowther and Rivier [79] have proposed an implementation of the classic Reynolds-like flocking model on UAVs. Their methodology produced promising results on computer simulations but has not been subjected to testing in reality. An alternative approach, although apparently difficult to be implemented on real robots, is the one described by Lawrence *et al.* [204], which according to the authors should scale up to swarm sizes as large as comprised by 147 elements. The reader must have noticed that, notwithstanding the claims made by various researchers, a demonstration of proper autonomous flocking behaviour in-

volving physical MAVs is still lacking. For this reason part of our efforts are focused in that specific direction.

The term "flocking" should not be confused with "swarming", a mistake made for example by Corner and Lamont in [77]. Swarming tends to refer to the field of Swarm Robotics [369, 30], which has some peculiar characteristics (as for example emergence and sitgmergy). Some work on the coordination of multiple UAV vehicles has been carried out using swarming principle, as it has been the case of Bamberger [25] and Pamphile [287]. Somewhat different is the work done by Melhuish and Welsby [244], which implemented what they defined as "secondary swarming" on a group of airships.

Terminology misuses can cause confusion in regard to both terms. This is for example the case of Allred *et al.* [9], who have developed an airborne Wireless Sensor Network (WSN) - called SensorFlock - for the purposes of monitoring wildlife and ecological systems. The peculiarity of this WSN is that its sensors are entirely MAVs. Unfortunately their work does not involve any kind of "flocking behaviour", as the project name would otherwise suggest.

In all cases, the simulations we describe here do not share all the principles of swarm systems, thus we will not define them as Swarm Robotics models. The first two models simply rely on independent individual behaviours that link together to make it possible to achieve higher-level tasks. The third model implements instead a proper flocking behaviour as intended by Reynolds.

Apart from being a way to achieve collective behaviour via distributed control, flocking is also interesting because it makes possible the study of formation flight. This is interesting for a number of reasons. When it comes to MAVs, the most compelling of these reasons are surely autonomy and computational capability. The small size characterising micro-unmanned aerial vehicles, and the consequently restricted payload they can carry onboard, generally means that the batteries they use cannot guarantee a long autonomy (it is rare to find MAVs exceeding 40 minutes of continuous flight) and that the onboard information processing devices (*i.e.* computer boards) are limited in terms of computational power. Flying in forma-

tion would allow the aircraft to save energy during the flight due to the reduced air resistance the MAVs will encounter, and to share among the individual members complex computational tasks, that could in this way be performed in the required amount of time (or, tasks impossible for a single MAV would become possible for a group of them).

4.3 The main challenges

Each of the two approaches we propose for the implementation of collective behaviour comes with its own challenges. For what concerns the ER models, apart from the general and well known issues generally associated with Evolutionary Robotics (already reviewed for example by Inman Harvey [152] in 1993 and for the most part still valid today), additional complications are present, mainly in terms of motion restrictions and computation time required for the evolutionary process. The issue of having to deal with non-omnidirectional robots characterised by a limited turn rate is the main factor affecting the flocking-based approach as well. Both design methodologies elaborated have then to face the so-called "reality gap" problem.

All of these issues, plus a few more, are described in the following sub-sections.

4.3.1 Fixed-wing aircraft: motion constraints

Fixed-wing MAV configurations are interesting because of their simple underlying mechanical design and the associated high speed and energetic efficiency provided [411]. However, they present some drawbacks when compared to rotary-wings or to other more recent (and still quite unusual) flapping-wing aerial vehicles.

The most compelling drawback is related to their highly constrained motion dynamics. Unlike ground-based robots or helicopter-like aircraft, they need to maintain their flight velocity above a certain threshold in order to prevent a stall. Slowing down the global speed of the robots can therefore only be done by having the robots turn, thus making it impossible to implement "stop-and-wait path deconfliction algorithms" (as defined by Beard and colleagues [31]). The rate at which these robots turn is also capped (robots can not turn on the spot) depending on the dynamics of

the platform. These properties lead to more complex robot trajectories than what would typically be observed with other platforms as shown in Figure 4.3. Taken from a collective behaviour perspective, such motion dynamics/constraints could lead to robots rapidly changing their position relative to one another which can disturb inter-robot interactions and communication [104, 156].

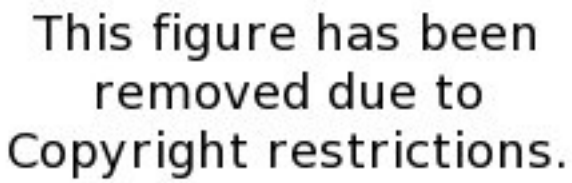


Figure 4.3: Motion constraints for a fixed-wing aircraft. Source: [156]

Additional issues, though more relevant to experiments involving physical robots rather than simulation, are those associated with takeoff and landing procedures. While helicopters or other forms of multi-rotor air vehicles can take off or land by simply moving along a vertical axis, fixed-wing aircraft traditionally require a significant amount of space on a runway to reach the speed needed to take off. The landing phase also consists of a slow and progressive reduction of both the flying altitude and the loiter speed, until the aircraft gets to the ground, thus requiring a wide and clear area available. The situation is slightly better when it comes to MAVs. Many of them can be in fact hand-launched or may integrate VTOL functionalities, but as we can see in Appendix A this is not the case for all of them. The same applies to landing/recovery procedures.

One of the main challenges in this thesis is therefore to develop controllers that should will be able to cope with the typical motion constraints of fixed-wing aerial robots by acting mainly on their turn rate while maintaining a constant forward

speed.

4.3.2 Obstacle avoidance

The issue of obstacle avoidance is a well-known area of studies in mobile robotics. Khatib, in his pioneering work published in 1985 [188], was amongst the first researchers to demonstrate how this problem, that was traditionally thought to be a high level planning problem, could be instead tackled and solved in a distributed way.

Behaviour-based and Evolutionary Robotics are two approaches that can deal very well with obstacle avoidance performed in real-time. They do not require any higher-level knowledge of the environment other than the limited perception provided by a set of sensors embedded on the vehicles controlled. Sensors can be as simple as ultra-sonic ones, as demonstrated for the first time in 1988 by Borenstein & Koren [47], which have become the preferred method (together with infrared sensors) for implementing obstacle avoidance behaviour in mobile robotics platforms.

4.3.3 Target tracking

The problem of tracking and approaching a moving target is also fairly common across the scientific literature. Traditional approaches to autonomous navigation usually require the robot to elaborate a prediction of the target motion based either on real-time observations or on already available knowledge [42]. Based on this prediction, the robot can then modify its action plan accordingly.

A common instrument employed in control theory is the Kalman filter (or Linear Quadratic Estimation, LQE) [184], a recursive algorithm which uses a series of measurements collected over time in order to produce estimates of unknown variables. This algorithm allows to estimate the past, present, and future states of a certain system, even when the precise nature of the modeled system is unknown and the observations are noisy [391]. The Kalman filter has been successfully used in autonomous robotics for tasks as different as state estimation for robotic helicopters [275] and robot localisation inside noisy environments [275].

The main drawback of analytical methods as the Kalman filter consists in the fact that extracting the information required and then translating it into coherent actions is generally a non-trivial task. It is true that this approach, although generally hard to be implemented, can be effective and lead to good results when carefully put into effect. Nonetheless, in presence of a target that does not move according to any specific pattern, the task increases dramatically in terms of complexity, making it difficult for a human designer to reach a working solution using traditional control theory methodologies.

Various authors - such as Ablavski [3] and Bertuccelli [36] - have proposed interesting ways to overcome this issue, respectively focusing on individual and collective behaviours, but the problem remains a serious one and a definitive solution to it seems unlikely to be achieved in the near future.

4.3.4 Collective behaviour, distributed control, and cooperation based upon implicit communication

Collective behaviour involves cooperation amongst individuals belonging to the same group having to perform a certain task together. The choice we have made, which focuses this work on collective behaviours rather than on the problem of controlling an individual unmanned aircraft, has been dictated by a pure scientific interest. Even if it is true that many interesting issues can be faced during the design of a "low-level" controller (essentially an autopilot system) for a single aerial vehicle (*e.g.* autonomous takeoff and landing are two areas where many different and potentially very interesting approaches could be tested), the room for "intelligent" approaches is small. Or, at least, it is relatively small when compared to the extremely wide range of possibilities available to researchers willing to explore the domain of collective behaviours. Good examples of the complexity involved in designing effective cooperative strategies for teams composed of many unmanned vehicles can be seen in the work (although focused on wheeled robots) of Hussain, Montana, & Vidaver [174], as well as that of Gaudiano, Bonabeau, & Shargel [130].

According to our perspective, when we look at cooperation we do this from a

complex systems point of view. As the computer models we have developed can be easily interpreted as sophisticated multi-agent systems, we want the cooperation amongst MAVs not to be explicitly designed but, rather, to emerge as a high-level property from the low-level interactions between the individual aircraft and the environment, replicating in this way dynamics typically observed in multi-agent systems.

Moreover, distributed control, intended as the implementation of collective behaviour within a group without using a central controller, is generally considered a notably interesting problem from both technological and scientific perspectives [279]. The advantages provided by distributed control systems over centralised ones are mainly related to the greater degree of robustness (intended as fault tolerance) they offer, as technical or communication problems affecting one of the individual elements controlled does not affect the functioning of the entire system. As a drawback, distributed controllers are generally more challenging to design than centralised systems. These points have been highlighted by Wu [399], who has argued that distributed control is generally preferable over centralised control since its non-critical reliance on any specific element can in turn guarantee increased reliability, safety and speed of response to the entire system.

Coordination and distributed control, however, can not exist without communication. If the individual members of a group have to perform a certain task together and no centralised controllers are available in order to instruct every robot with regard to what it is supposed to do, the individuals must necessarily have a means to exchange information amongst themselves. Research by Kube [198] demonstrated how explicit communication is not a compulsory requirement in collective robotics. This does not contradict our previous point. The point highlighted by Kube is simply that communication does not necessarily need to be explicit (thus intentional), but it can be implicit (*i.e.* non-intentional) as well. An implicit communication exchange between two agents takes place for example when one of them gathers information by observing the behaviour of the other agent, without the latter necessarily willing to communicate anything by mean of its actions. This is the approach

that we will follow herein.

4.3.5 Computation time issues

The Evolutionary Robotics approach is particularly expensive in computational terms for what concerns the evolution of the controllers. According to the complexity of the software simulator used and of the behaviour to be evolved, a single evolutionary run could easily last for a significant amount of time (*i.e.* hours or days depending on several variables, as for example the computational power available, the programming language used to write the evolutionary algorithm, etc.). Furthermore, evolutionary algorithms are heavily affected by elements of randomness, thus requiring the evolution process to be repeated several times in order to generate reliable results. Powerful computers are therefore required, especially when the modelling of the various components simulated has been done with a rich level of detail. This is often the case for much of the research in ER carried out nowadays, which frequently relies on complex real-physics based software simulators. Amongst the first experiments involving physics-based simulators are those carried out by Bongard *et al.* [45] and by Reil & Husbands [310], both focused on the development of a control system for the locomotion of a bipedal robot.

In more recent times, thanks to the general improvement in personal computer technologies that has made computation power more accessible, this approach has gained even more popularity, extending to areas that could have not considered approachable by Evolutionary Robotics just a few years ago. An example can be seen in the Mars Rover simulator developed by Peniak and colleagues [293], an Evolutionary Robotics model based upon a complex physics-based simulator where an accurate replica of the Curiosity rover[7] has to accomplish autonomous navigation and obstacle avoidance relying either on the readings coming from a set of ultra-sonic sensors or on active vision mechanisms.

This computational/time issue, which is the most compelling reason for ER researchers to use computer simulators rather than evolve behaviours straight on

[7] `http://en.wikipedia.org/wiki/Mars_Science_Laboratory`

physical robots (as this would make the process even slower), has been tackled in the work presented herein in three complementary ways: 1) reducing to the minimum level possible the complexity of the simulator software used; 2) testing several variations of the evolutionary algorithms in order to find the one offering the best tradeoff between accuracy of the behaviours evolved and computational complexity; 3) introducing multi-threading programming methodologies to develop simulators running on multi-processor computers. Details about the computer facility used for our simulations are available in Appendix C.

4.3.6 The reality gap

When moving from computer simulations to real robots it is hardly surprising to discover that the performances of the developed (or evolved) controllers drop significantly. This issue is generally referred to as the "reality gap" and it is particularly relevant in Evolutionary Robotics. As what takes place inside a computer simulation is typically much more precise and accurate than what happens in reality, controllers designed with a computer "in mind" can frequently get lost as soon as they are tested on the real world. This is due in part to the fact that physical hardware can fail in many different respects and that sensor readings might often be inaccurate. The main reason is that the real world is extremely more unstructured and irregular than that which any computer simulation might be able to replicate. A controller evolved in simulation could react when tested in reality as if being put into an environment it had never experienced before (a problem which, we should stress, is common among every design technique and not peculiar of ER).

Husbands and Harvey, two of the pioneers of Evolutionary Robotics, demonstrated that they were well aware of this problem as far back as 1992 [173]. The solution they proposed is still the most widely employed today. It simply consists in making the evolutionary process in simulation less optimal, through the addition of artificial noise to the sensor readings. Effort must be put into generalisation as well. Generalisation can be achieved during the design of a controller by ensuring that it is exposed to the most different conditions possible, in such a way that it will not

"overfit", *i.e.* to only perform well in the environment it was trained for. In this way the controller will not evolve exploiting behaviours due to possible mistakes made by the programmer of the simulator or particular (not necessarily existing outside of the simulator) environment characteristics.

Extensive research on the reality gap issue, specifically focusing on Evolutionary Robotics, has been carried out by Jakobi *et al.* [180], having as aim the measurement of the proper amount of noise to be used in the simulations. Investigations on the role of noise have also been performed by Miglino *et al.* [253], who introduced in their simulations what they have defined as a "conservative" form of noise.

Zagal and colleagues [405] have elaborated a different approach instead, consisting in the robot to keep evolving/adapting its own autonomous controller once transferred to the real world.

Later on, the aforementioned Jakobi theorised "the minimal simulation approach" [179], dictating that an Evolutionary Robotics simulation must only model the aspects that are relevant for the controller development, making the strong assumption that the designer is able to identify those.

Zufferey [410] has discussed in detail the problem of reality gap in the more specific area of aerial robotics instead, focusing an indoor airships.

4.4 Ethical considerations on the military employment of lethal autonomous robots

As we have seen in the previous chapter, the aerial autonomous robotics domain is characterised by a strong military footprint. Most of the latest developments in the filed come from military research and the battlefields are the arenas in which the technological innovations are put at test. The US, who have been employing several thousands of military robots during operations Enduring Freedom and Iraqi Freedom [345], have launched several plans (see for example the Future Combat Systems[8] running from 2003 to 2009, the Unmanned Systems Roadmap 2007-2032 [70],

[8] `http://en.wikipedia.org/wiki/Future_Combat_Systems`

and the Unmanned Aircraft Systems Roadmap 2005-2030 [62]) aiming to increase the proportion of autonomous robots within the American military apparatus. According to their plans, one third of the US operational ground combat vehicles will be unmanned by 2015 [345]. No exact figures have been provided for UAVs, but it is not utopia to believe that these numbers could be even higher for unmanned aircraft.

At the current stage the widely employed Predator and Reaper UAVs, as well as the counterparts employed by other militaries, are only semi-autonomous robots, as they rarely fly in complete autonomy and they still require a man-in-the-loop to decide when to perform a potentially lethal operation (*e.g.* to fire a missile). The legal and ethical implications for the men in control of these UAVs are the same as for piloting an aircraft or calling in the coordinates for a traditional air strike. However, one the new goals of the military research now consists in getting rid of the man-in-the-loop and letting the military robots, either aerial, underwater or terrestrial, to acquire and fire their targets autonomously [46, 346, 348]. Apart from the technological challenges that this plan implies, there are several ethical considerations that have not only to be taken into account, but also to be promptly addressed.

In the next pages we are not going to discuss the morality of the research in autonomous robotics applied to the military world. Anyone may have his opinion on the topic, which I personally respect and which I do not intend to affect in any way. Rather, we will analyse the ethical implications of having autonomous robots on the battlefield in the light of the modern laws of war. Keeping in mind that, as suggested by Arkin [16], when properly functioning, robots can be even "more ethical" than humans, as they are unlikely to imitate the countless war atrocities committed by human soldiers during history.

4.4.1 The laws of war and the ethics of modern conflicts

Modern armed conflicts adhere to a rigorous set of rules concerning both the conditions under which wars can be started, and how, once begun, they must be con-

ducted. This body of law is generally referred to as the "laws of war."

Modern laws of war take inspiration from the "Just War theory", a doctrine of military ethics whose roots date back to 2,000 years ago[9]. The main goal of this theory consists in defining the criteria for a war to be considered "just", thus started, and those according to which carry it out. In its very essence, Just War holds that a violent conflict ought to meet philosophical, religious or political criteria, reflecting the footprint left over the years by several Christian philosophers[10].

Just War theory consists of two main principles: *jus ad bellum*, and *jus in bello*[11] [68]. The criteria belonging to the *jus ad bellum* category define the right to wage war:

- *just cause*: innocent human lives must be in imminent danger and intervention must be a mean to protect these. The reason for going to war can not be solely in recapturing things or punishing people who have misbehaved;

- *comparative justice*: the injustice suffered by one of the parties involved in a conflict must be significantly higher that suffered by the other(s);

- *competent authority*: a genuine war must be paired with genuine justice. Thus a just war can only be initiated by a political authority within a political system that allows distinctions of justice;

- *right intention*: force must only be used for the purpose of correcting a suffered wrong, without any material/economical implication;

- *probability of success*: the use of weapons must not be advocated in futile causes or where disproportionate measures would be required to achieve success;

- *last resort*: force is the last resort. It must only be used when every peaceful and viable alternative have been seriously tried and exhausted, or are clearly not practical;

[9] Cicero's "De Officiis" discussed "just war" in 44BC.
[10] Amongst the various contributors, Thomas Aquinas played a crucial role.
[11] Although some theorists have recently proposed an additional third category, *jus post bellum*.

- *macro-proportionality*: the anticipated benefits of waging a war must be proportionate to its expected evils or harms.

For what concerns *jus in bello* instead, its principles dictate how combatants are expected to act once war has begun:

- *discriminability*: the acts of war must be directed towards enemy combatants only, and not towards non-combatants (*e.g.* civilians);

- *proportionality*: an attack can only be launched against military objectives in the knowledge that the incidental civilian injuries would not be clearly excessive in relation to the estimated military advantage;

- *military necessity*: the governing principle of a just war must be the one of minimum force. An attack must be targeted to a military objective and intended to help in the military defeat of the enemy. The harm caused to civilians and to their properties must be proportional and not excessive in relation to the direct military advantage anticipated;

- *fair treatment of prisoners of war*: any solider, either captured or surrendered, no longer poses a threat. Therefore he must not be tortured or mistreated in any way;

- *no means malum in se*: combatants must not use weapons or other methods of warfare considered as evil (*e.g.* mass rape, forcing soldiers to fight against their own side, or using weapons whose effect cannot be limited by time and space).

In modern times, two widely adhered international treaties have implemented the principles of Just War. The first is the Hague convention, signed in 1899 and further extended in 1907; the second is the 1949 Geneva convention.

4.4.2 The experts' point of view on autonomous robotics

Why the above parentheses about how modern wars are regulated? Because this is the context in which several scientists fear autonomous robots might not be able to

cope once humans are taken out of the control loop.

Amongst all the roboticists, philosophers and war strategists that have studied the potential impact of autonomous military systems a prominent role has been played by British scientist Noel Sharkey. Since a few years ago, Sharkey is involved in a fierce campaign aimed to convince policymakers around the world that today's robots are nowhere near to fulfill their expectations. In his work *''Weapons of indiscriminate lethality*" published in 2009 [347], Sharkey specifically addresses two elements of the *jus in bello* principles that he believes are still out of reach to modern robotics systems: discriminability and proportionality.

Concerning discriminability, Geneva convention suggests the use of "common sense" in discriminating between civilians and combatants. An additional protocol ratified in 1977 specifies that one who is not a combatant must be classified as a civilian. How to instil "common sense" in an artificial system is one of the most challenging issues faced by modern AI [236]. How could an artificial system autonomously classify between combatants and civilians? The task is not easy by any means. Is anyone wearing a uniform a combatant? Surely not. But even if that would be the case, what classifies a certain garment as a uniform? Should we state instead that anyone carrying a weapon is a combatant? Not necessarily, as anything could be considered a weapon depending both on the context and on the intentions of who holds it. In other words, to apply human-like "common sense" is a hugely complex task, which requires either a wide amount of information available and the ability to process this information in the light of a wider environmental context. Even assuming to have access to extremely reliable robot sensor systems, so sophisticated as to be capable of extracting any useful piece of information from the environment (something that current technology does not allow yet), matched with algorithms that can use this information together with previously learned knowledge to classify in real-time between civilians and combatants, the discriminability problem would still not be solved completely. On one hand the friendly fire issue remains a concern. How to discriminate between an ally and an enemy soldier and act accordingly? Some authors, as for example Garfinkel [129], have suggested to

equip every soldier with RFID tags, thus making the recognition task as simple as possible. This solution has nonetheless a number of drawbacks. For example an enemy unit might get rid of his RFID tag, or, even worse, he can produce a fake one pretending to be an ally rather than an enemy[12]. On the other side the legality of the combatant in front of the robot has to be taken into account. An enemy soldier may be wearing a uniform, carrying a weapon and having the proper RFID tag with him, but his intention could be that of surrendering rather than fighting. How can a robot understand that without a proper theory of mind embedded in its circuits? Some authors, as for example Canning [64], have found a shortcut for this problem proposing a working principle for military robots which can be summarised in the sentence "let machines target other machines only". Sharkey, more radically, proposed to ban the military use of autonomous robots until they can pass a sort of "innocent discrimination test" [347].

The second potential element of troubles identified by Sharkey is the principle of proportionality, which requires that the anticipated loss of life and damage to property incidental to attacks must not be excessive in relation to the concrete and direct military advantage expected to be gained. In other words, the "force" to be used during a military action must be "proportionate", *i.e.* neither excessive nor insufficient, to the advantages that can be achieved. How to calculate the right amount of force to apply in a certain operation? Unfortunately there is a lot of uncertainty on how to make such calculation. Military officials are specifically trained for years for this purpose. The difficulty involved in this operation is partly due to the fact that the entire process relies on a extremely wide array of factors, such that it has never been possible to capture all of them in an algorithm (so to be implemented on a computer). Furthermore the military decision-makers, in performing their calculations, must also take into account the possibility for at least some of the intelligence they have at disposal to be inaccurate (as it is has be proven to generally be the case [37]).

Alongside discriminability and proportionality there is nonetheless another very important factor to consider when thinking about the introduction of military robots

[12]Similar topics are covered by Richard Clarke and Robert Knake in their book "*Cyber War*" [72].

inside warfare environments. This factor, which has been extensively studied by Sparrow [352], is responsibility. Who has to be considered responsible in case something goes wrong? If, for example, a robot such as the SWORDS[13] decides to exterminate the civilian population of a village? Or, simpler and much more likely, if it fires a single bullet which misses its designated target and ends up injuring an unfortunate ally soldier? Again we are facing a tough scenario. The entire chain that brings a robot to the battlefield is a long one (as it includes manufacturers, programmers, designers, etc.) and errors can take place at any stage. Even a well projected robot might suddenly behave unexpectedly because of some unavoidable hardware failure [345]. Modern militaries rely on rigorous procedures to determine who is responsible for any sort of adverse event that could potentially occur during a conflict. But machines have never been considered to be anything other than tools, thus being exempt from any attribution of responsibility. Autonomous robots require the military theorists to develop new responsibility attribution procedures. As Sharkey ironically put it [347]:

> "*Who is to be held responsible for the lethal mishaps of a robot? Certainly not the machine itself. There is no way to punish a robot. We could just switch it off but it would not care anymore about that than my washing machine would care. Imagine telling your washing machine that if it does not remove stains properly you will break its door off. Would you expect that to have any impact?*"

Although the author agrees with several of the issues raised by Sharkey, he is also convinced that the British scientist is somehow too pessimistic in his views. It is certainly true that robotics is a growing but not yet mature area of studies. It is true as well that today's robots are not capable of performing tasks that government decision-makers believe are at their reach instead. Are these solid enough reasons for entirely banning robots from the warfare scenarios? Probably not. Nonetheless they surely can serve as useful warnings that every person working in the field should take into proper consideration. There are no reasons, in the author's opinion, for

[13] `http://en.wikipedia.org/wiki/Foster-Miller_TALON`

halting the research on such robotics systems and the associated field tests, as long as military planners do not expect to see robots smoothly performing extremely sophisticated operations in the war field as those depicted in Hollywood movies.

Furthermore a few flaws can be found in Sharkey's reasoning. First of the scientist seems to always refer in his publications to AI systems based on explicit knowledge representation, thus implicitely restricting the entire Artificial Intelligence arena to the symbolic approaches only. Sharkey plainly seems to be unaware (although, as an expert on the field, he surely is not) of the several design methodologies for intelligent systems developed in the last decades that do not rely either on explicit representations of knowledge, on formal decision-trees, or on rule-based systems[14]. The work we are presenting in this thesis constitutes a perfect example in this sense. We will see autonomous controllers for unmanned aerial vehicles based on evolved neural networks that, by definition, can perform complex tasks without the need for any formal representation of knowledge. Second, in pointing out the limitations of modern robots, Sharkey (especially in [346]) likes to think of military autonomous systems dealing with irregular insurgents. It is certainly true that a clearly identifiable post-Cold War trend is the one towards asymmetric warfare. As the continuous advances in military technologies tremendously widen the gap between the war capabilities of different nations (and the militarily most advanced countries prefer to fight each other over diplomatic channels rather than on the field), fewer and fewer countries are prone to wage war to each other. Much more common is the case in which a regular army has to face insurgents rather than another conventional army, as recently happened in Afghanistan during operation Enduring Freedom. At the same time the existence of this trend does not imply that the research in military equipment for "conventional" wars has to be stopped. Political equilibrium, as history demonstrates, can change suddenly. Of fundamental importance, for the military forces of every country, is to be ready and well equipped in case the unexpected happens. Autonomous robots, as we have extensively discussed in previous sections, can constitute a very strong asset in any military force.

[14]One of the reasons for discussing about these methodologies only might consists in the fact that it is easier to produce a safety case using purely deterministic and formally defined methods.

And, even if military robots can arguably do their best in a "regular" war, this fact does not prevent them to be potentially very useful in different warfare environments as well. In particular when both ongoing and future research will have released their outcome.

4.5 Safety issues

The previous paragraph has dealt with the ethical issues related to the employment of autonomous unmanned aerial vehicles (and robots more in general) in war environments. There is nonetheless an additional area that, as reviewed in chapter 3, is likely in the next few years to constitute a good share of the overall MAV usage. This broad domain is law enforcement and includes several activities as for example crowd control, accident investigation, search and rescue, covert surveillance, etc.[15].

Using autonomous aerial vehicles for law enforcement tasks often implies having them to fly over non-warzone environments (*i.e.* where safety issues have to be taken into serious account), as the most crowded quarters of a traditional modern city, thus endangering the safety of people present in those areas. On the technical side there are obviously a lot of issues that have to be dealt with before having MAVs flying comfortably in such environments. Autonomous control is not a straightforward task when related to urban areas, to an extent which does not only depend on the topology of the territory, but also on the kind of robotics platform used (helicopters, as we have discussed already, are by far more manageable than fixed-wing MAVs, but also much noisier in comparison, which could be an issue for particular kinds of task). However, what we are interested in discussing here is rather what are the requirements that should be enforced for MAVs to safely operate within urban environments.

At the current stage there is a lack of ruling on the topic. Several researches have been published focusing on large-size UAVs (see for example Dalamagkidis *et al.* [82] dealing with the risks of unmanned aircraft ground impacts, or Loh *et al.* [222] on

[15]The private company Aeryon Labs proposes an interesting analysis on its website focusing on the most compelling reasons for a police force to employ MAVs (`http://www.aeryon.com/applications/whitepapers/224-whitepaperpolice.html`)

UAS in the civil airspace), but these do not account for the different hazards that small MAVs could generate (although small in terms of size and weight, even small aircraft could pose a significant threat for people exposed to them, mainly because of the damage potentially provoked by moving mechanical parts such as the propellers or the rotors). Nonetheless typical regulations concerning RC aircraft (or "model aircraft" as they are often referred to in lawmakers' jargon) exist already and can be used as a basis for further work in the direction of creating a proper legal framework for operating autonomous MAVs. In the UK, the prototypical example of a country with modern model flying regulations, the authority in charge of the aerial space is the Civilian Aviation Authority (CAA), which provides somewhat loose guidelines for the use of MAVs in public places. In its publication titled "*Model Aircraft: A Guide to Safe Flying*"[16], focused on "small unmanned aircraft" (according to their definition, any unmanned aircraft having a mass of no more than $20kg$), the CAA determines the conditions under which such an aircraft can be flown. Amongst these, there are two points warranting a closer look (article 166):

> *(2) The person in charge of a small unmanned aircraft may only fly the aircraft if reasonably satisfied that the flight can safely be made.*
>
> *(3) The person in charge of a small unmanned aircraft must maintain direct, unaided visual contact with the aircraft sufficient to monitor its flight path in relation to other aircraft, persons, vehicles, vessels and structures for the purpose of avoiding collisions.*

As we can see there is a wide degree of subjectiveness put in the hands of the person flying the model, as it is up to him to determine whether "he is reasonably satisfied that the flight can safely be made." Once this condition is met, the only additional requirements simply consist in maintaining direct visual contact with the aircraft throughout the entire flight.

It is important to consider that a further distinction is applied by CAA between small unmanned aircraft having more or less than a $7kg$ mass respectively. Those

[16] `http://www.caa.co.uk/default.aspx?catid=1416&pageid=8153`

falling in the latter category do not require any authorisation from the civilian authority.

Article 167 imposes additional requirements for (not better defined in terms of size) "small unmanned surveillance aircraft":

> *(1) The person in charge of a small unmanned surveillance aircraft must not fly the aircraft in any of the circumstances described in paragraph (2) except in accordance with a permission issued by the CAA.*
>
> *(2) The circumstances referred to in paragraph (1) are: (a) over or within 150 metres of any congested area; (b) over or within 150 metres of an organised open-air assembly of more than 1,000 persons; (c) within 50 metres of any vessel, vehicle or structure which is not under the control of the person in charge of the aircraft; or (d) subject to paragraphs (3) and (4), within 50 metres of any person.*
>
> *(3) Subject to paragraph (4), during take-off or landing, a small unmanned surveillance aircraft must not be flown within 30 metres of any person.*
>
> *(4) Paragraphs (2)(d) and (3) do not apply to the person in charge of the small unmanned surveillance aircraft or a person under the control of the person in charge of the aircraft.*
>
> *(5) In this article a small unmanned surveillance aircraft means a small unmanned aircraft which is equipped to undertake any form of surveillance or data acquisition.*

Of course these conditions can not be all satisfied during law enforcement operations. Flying at a safe distance from people would make MAVs absolutely useless for operations as crowd control or intelligence gathering more in general. The only solution we can see would consists in allowing security forces to operate their autonomous aircraft on the basis on much more permissive laws. In order to do so safely MAVs need serious improvements in their control systems compared to today's standard, not only for what concerns obstacle avoidance, but also take-off and

landing. The last two are indeed the most critical stages of every flight, and those requiring as well the more room to be performed effectively (particularly for aircraft with fixed-wing configurations). Furthermore, a common practice amongst experimenters in autonomous aerial robotics consists in having the possibility of switching at any time from autonomous to manual control via radio link (having also a backup radio control system in case of failure of the main one). This capability should be present in MAVs used within urban environments as well, as the risk of a malfunctioning is always a significant one. This implies that an expert pilot of RC aircraft has to be available in the neighbourhood of the area where MAVs are employed (not necessarily in the proportion of 1-to-1), at least as long as the technology does not become so accurate that this presence would be unnecessary.

4.6 Plan for the experiments

The main characteristics of the two design methodologies we have decided to adopt (Evolutionary Robotics and flocking algorithms) have been introduced earlier on and are summarised in the bullet points below.

- Evolutionary Robotics approach: a multi-agent system involving teams of fixed-wing MAVs dealing with several variants of "search and hit" tasks is employed to implement a reactive strategy approach. The autonomous controllers evolved are simple neural networks that also have to be capable to generate an obstacle-avoidance behaviour in the robots;

- flocking algorithms: a computer simulator, replicating dynamics based upon incremental geometric flight as defined by Reynolds, is used to design controllers for MAVs obeying to the flocking rules of: 1) collision avoidance; 2) velocity matching; and 3) flock centering.

Now time has come to describe what the experimental part of this work consists of. For the purposes of this thesis, three different computer models have been developed.

The first one, described in chapter 5, is a two-dimensional Evolutionary Robotics simulator which focuses on individual and cooperative navigation tasks from a high-level perspective. The main goal of the experiments carried out with this simulator is to investigate whether the motion of fixed-wing aircraft (including all of the constraints we have discussed so far) can be successfully modelled through traditional Evolutionary Robotics methodologies or not. Several experimental setups are tested, all of them requiring a group of MAVs to autonomously fly towards a certain target area deployed in a random position within the environment. The first set of experiments acts as benchmark, in order to evaluate the maximum level of performance the evolved controllers can achieve in this scenario. At the same time, investigations are carried out to identify the proper encoding for the input information to the neural network controllers. Then the obstacles are introduced into the environment and a corresponding obstacle avoidance ability is evolved for the MAVs. In addition, a further challenge is added to the model introducing a target capable of moving away from an approaching aircraft. Finally, an experimental setup requiring coordinated behaviour (*i.e.* two aircraft reaching the target, either static or moving, at the same time) achieved through implicit communication is developed.

The second computer model, covered in chapter 6, is again dedicated to the Evolutionary Robotics approach. The scenario is now more complicated than the previous one. Despite the fact that no obstacles are deployed into the reference environment, the three-dimensional nature of the new simulator makes the control task significantly more challenging than before. Apart from just performing yaw manoeuvres, the MAVs can now pitch and roll also. With this computer model we want to understand how far we can stretch the model discussed in chapter 5. Thus the same experiments carried out in the 2D simulator are performed.

Finally, the last simulator developed can be seen at work in chapter 7. This computer model allows for experimentations on flocking, implementing among others a customised version of the original Reynolds' algorithm. The flocking algorithm is tested in simulation and the results obtained are reported. A slightly modified version of it is also evaluated on physical robots, namely a fleet of *swinglet* MAVs which

we were able to use thanks to a collaboration with the Laboratory of Intelligent Systems of the EPFL. The chapter also provides a detailed look into the experiments carried out on waypoint navigation and leader-following behaviour, both in simulation and on the real robots. The principal aim of these investigations is to verify to what extent the controllers we have developed can be smoothly applied to physical MAVs.

4.6.1 Defining "success"

When dealing with Evolutionary Robotics it is common, at the end of the evolutionary process, to obtain populations of controllers that amongst them perform in different ways. For simple tasks, the evolutionary algorithm (assuming, of course, a proper design in terms of fitness function chosen and parameters associated to the evolutionary algorithm) can generally lead to a quite uniform distribution of "skills" amongst the individuals belonging to the last generation. In more complex scenarios instead, it is not unusual to find just a few individuals successfully coping with the task while the others fail, at various levels, in doing that. This is not a problem, anyway, as most of the time we can discard all the controllers we do not like and just extract the best one out of the bunch.

For this reason, across the various Evolutionary Robotics experiments presented in the next two chapters we will mainly focus on one single statistic, which is the best success rate, *i.e.* the success rate scored by the best individual in the population at any generation across the various tests it is subject to. Ideally we would like to obtain a 100% success rate in every scenario (and, as we will see, most of the time we managed to reach this value), but considering several factors (how our simulations are affected by elements of randomness, how the controllers were tested only a few times each in order to obtain an evolutionary process fast thus sacrificing accuracy, etc.) for the purposes of the research described in this thesis we have decided to arbitrarily set the threshold to 90%. Although far from the ideal 100%, the value we have chosen looks good in comparison to what can be found in the literature. Just to give the reader an idea, in the work on UAV automated forced landing presented by

Eng and colleagues [102] in 2007 (one of the few that poses a great deal of attention on the concept of success rate, although not defining any threshold to discriminate between the outcome of their experiments), the score they achieved was just 52%.

Things are slightly different for chapter 7 instead. The work presented in there can be seen as a proof-of-concept, which is used to demonstrate how controllers like the ones we designed via the evolutionary approach can be applied, smoothly and successfully, to a real MAV. Therefore in that chapter there will be no measures of success/failure employed.

4.6.2 On the comparison with alternative design methodologies

What we are interested in demonstrating throughout this thesis is mainly how the Evolutionary Robotics approach, combined with elements typically found in other fields (as for example Multi-Agent Systems), can be considered a serious alternative to traditional design methodologies when it comes to the development of autonomous controllers for flying robots. Thus we do not intend to demonstrate herein a potential superiority of our approach in comparison to those arisen from decades of studies in control theory, but simply to point out that the latest technological developments have brought a new competitor into the field. A competitor which not necessarily performs better than human designers when the task the controller has to deal with can be solved analytically but, at the same time, one that has the potential for coping properly with tasks that are too complex for traditional methodologies. This the reason why in the next chapters the Evolutionary Robotics controllers presented will not be compared with alternative control systems (*e.g.* neural networks trained via back-propagation, or analytical control theory models). Furthermore, a strict benchmark comparison with alternative methods would not be possible given the dynamic and variable trajectories followed by the simulated MAVs to reach a dynamic, moving target. Back-propagation, for example, would require the fixed, ad-hoc definition of the trajectories to reach the target, and these trajectories depend on other agents' and target's behaviours. One might argue that an analytical

model could be employed to generate a pseudo-teaching input (the optimum trajectory to follow under certain conditions), but there would be a clear issue concerning generalisation. A neural network controller trained in this way would likely be able to follow a pre-determined flight path, thus loosing the typical flexibility offered by such controllers (while, on the other hand, no obvious advantages are visible).

Chapter 5

Simulation Experiments in Urban Layouts

Within this chapter we describe the first of the three computer models we have developed for the experimental part of the research presented in this this thesis.

The software simulator upon which this chapter is centred on implements a simple two-dimensional multi-agent model which focuses on high-level MAV navigation, thus ignoring all the aspects related to aerodynamics and flight stabilisation that are assumed to be taken care of by an autopilot system.

The approach we have implemented falls into the category of reactive strategies (see section 4.1), and relies on a mixture of local and global information that the MAVs use in order to continuously modify and adapt their behaviour attempting to satisfy the requirement of the mission they are involved in.

5.1 Software simulator

As we have mentioned in previous chapters our approach, relying on Evolutionary Robotics, requires the use of a computer simulator for the evolution of the autonomous controllers. The need for a computer simulator in the context of Evolutionary Robotics should not be interpreted as a dogmatic view. It has in fact been challenged both at the early stages of the field (see for example the pioneering researches of Urzelain & Floreano [373], and Watson *et al.* [387, 388]), and in recent

times thanks to the work by Eiben and colleagues [99] focused on online onboard evolution. Online evolution can be considered a valid candidate for the design of autonomous controllers for land-based robots, but for obvious reasons it can not be easily applied to the aerial robotics domain. While a wheeled robot moving inside an arena is free to make any sort of "behavioural mistake" and, most notably, it can generally stop at any time waiting for the controller to elaborate the next movement (or for the controller to evolve in response to changes into the scenario), things are not as easy when the reference robot is an aircraft. Inside an arena, a wrong movement made by a small wheeled robot as an e-puck or a Khepera will just end up most of the time in the vehicle harmlessly touching the rails that delimit the boundaries of the environment. A mistake made by an aircraft flying over an open environment could potentially be much more dangerous, both for the robot itself (who could crash resulting in a serious hardware damage), and for the safety of other agents possibly sharing the same environment with the robot[1]. Obviously, depending on the configuration of the MAV robot used, it might frequently be the case that the robot can not even stop at mid-air without incurring in a stall, making even more problematic the adoption of an online evolution approach and requiring a much more accurate planning of every single movement.

The structure of the software simulator developed is quite simple from a "theoretical" point of view, as it consists in a computational model implementing a two-dimensional multi-agent system which runs in discrete time steps.

From a software perspective instead, the simulator consists in a C++ application, relying on the Qt libraries[2] for the graphics part and in the Neural Network Framework (NNFW[3]) for the management of neural networks-related aspects. As all the instruments used, from the C++ compiler to the external libraries, are open-source and multi-platform, the application can be freely distributed[4]. Furthermore

[1]It is true that, due to safety reasons, experiments in aerial robotics are typically carried out within isolated areas to ensure that they do not put human beings at risk. Although this does not automatically exclude the risks that human operators are subject to.

[2]`http://qt.nokia.com/products/`

[3]`http://laral.istc.cnr.it/laral++/nnfw/`

[4]The source code can be downloaded from this web page: `http://www.tech.plym.ac.uk/soc/research/ABC/plymav/Communication_and_Distributed_Control_in_Multi-Agent_Systems/Downloads.html`

Figure 5.1: Screenshot of the 2D simulator in a scenario which includes obstacles

the sources can be compiled and executed on all the most popular operating systems available on the market[5].

A screenshot of the application, running on Mac OS X, is presented in Figure 5.1. The main window shows the simulation environment, including in this case building/obstacles (marked in red, see section 5.4.2), and highlights an "enclosed area" in the centre (represented in yellow). From a technical point of view the simulated environment has been implemented as a *QGraphicsScene*[6] on which operates a *QGraphicsView*[7] widget. On the right-hand side of the application window there are various elements the user can interact with. Looking closely, on the top right corner we can find a *QTabWidget* element[8] containing three tabs inside (see Figure 5.2):

- *Evolution parameters*: this tab contains all the parameters related to the configuration of the neural network controller (although the user can only modify

[5]Successful tests have been carried out on Microsoft Windows XP, Ubuntu Linux 10.04 and 11.04, Mac OS X 10.5.x, 10.6.x., and 10.7.x

[6]`http://qt-project.org/doc/qt-4.8/qgraphicsscene.html`

[7]`http://qt-project.org/doc/qt-4.8/qgraphicsview.html`

[8]`http://qt-project.org/doc/qt-4.8/qtabwidget.html`

the number of neurons in the hidden layer; different versions of the simulator have been compiled to cope with alternative neural network topologies) and of the evolutionary algorithm. Concerning the latter, the user can select: the number of evolutionary seeds (or "runs") to perform; the number of generation to evolve during each seed; the number of swarms/teams to use (*i.e.* the population size); the number of tests to be performed on each team when evaluating its fitness. Furthermore, the user can decide whether to use or not the mutation operator. In case he desires to use it, he can select both the associated probability p_m (see section 2.4.5) and the magnitude of the modifications to be operated (the software automatically divides by 2 the value x inserted by the user, and uses $[-\frac{x}{2}; \frac{x}{2}]$ as the range for the mutation operator);

- *Evolution (live)*: when an evolutionary process is started the user can monitor how it is progressing from here. This tab shows: the number of the current evolutionary run; the number of the generation currently evolved; the number of the MAV team under examination; the number of the current test; the number of MAVs still in "operative" state and the number of tests concluded successfully so far for the current team;

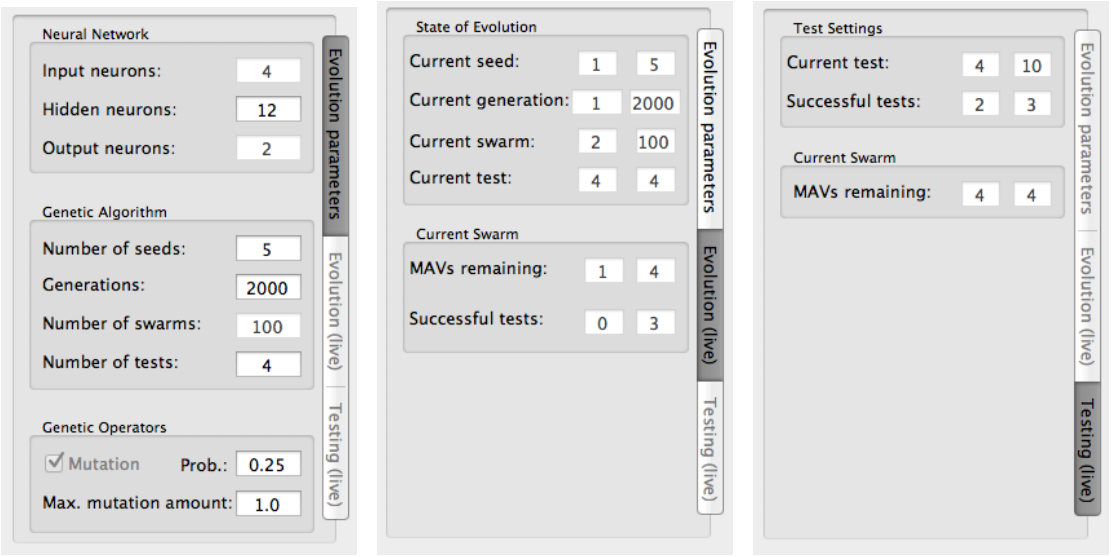
- *Testing (live)*: the testing tab gets activated when the user loads an already evolved controller from the memory and puts it under "examination". This tab simply shows the number of the current test, how many tests so far have achieved success and how many MAVs are still operative in the current test.

On bottom of the QtTabWidget there is a series of checkboxes (implemented using Qt's *QCheckBox*[9] class) that the user can use to modify the level of details displayed by the simulator and, in turn, the speed of the evolutionary process.

- *Graphics view*: this checkbox is used to enable/disable the graphics in the simulator. When disabled the main application window turns white;

- *View background*: active only when "graphics view" is enabled, this checkbox allows to display/hide the background picture on the main application window;

[9] `http://qt-project.org/doc/qt-4.8/qcheckbox.html`



(a) (b) (c)

Figure 5.2: Screenshots of the three tabs included in the QtTabWidget element of the application main window: 5.2(a) Evolution parameters, 5.2(b) Evolution (live), 5.2(c) Testing (live)

- *View statistics*: this checkbox is used to display/hide the statistics in the "Evolution (live)" tab;
- *Manual speed control*: enables/disables the combo box which is used to control the speed of the simulator;
- *Pause the simulation*: puts the simulator on pause.

These settings affect the speed of the evolutionary process as they can minimise the time the CPU has to dedicate to tasks that are not of fundamental importance for the evolution. Additional "tricks", different among them depending on the operating system used, also provide several advantages in terms of speed. Both on Windows and Linux, for example, minimising the application window dramatically reduces the time needed to evolve the required number of generations. On Mac OS X the simple minimisation of the application window has a limited effect on the speed of the evolutionary process; hiding it instead ($cmd + h$ shortcut) leads to a massive performance boost.

The simulator can also be slowed down by using the "manual speed control" combo box (a *QComboBox*[10] item), which is set by default on the maximum speed

[10] `http://qt-project.org/doc/qt-4.8/qcombobox.html`

possible ("Normal", index 0). Alternative options are "Slow" (index 1), "Very slow" (index 2), and "Extremely slow" (index 3). To reduce the simulation speed the software uses at every time step a *for* loop in which simple mathematical operations are computed $index * 10,000,000$ times.

An additional feature of the simulator consists in the zoom slider (technically a *QSlider*[11] object). When the graphics is enabled the user can zoom in and out the simulation environment using that slide bar.

Finally, in the bottom right corner there are the control buttons (*QPushButton*[12]). Using these buttons the user can either start/stop a new evolutionary process, or load and test an already evolved controller from the memory. The user can stop an evolutionary process at any time and, in case he does it before the evolution has reached the last generation scheduled, he is offered the possibility of saving the individuals belonging to the current generation in order to restart the evolutionary process later on without loosing the progress achieved. The individuals are saved into dedicated XML files using the NNFW's *saveXML*[13] method, which produces files as in the extract which follows:

```
<!DOCTYPE nnfw-xml>
<nnfw version="1.1">
<neuralnet>

<cluster numNeurons="12" type="BiasedCluster" name="outL">
<accumulate>false</accumulate>
<inputs>-0.200014 -0.106269 -0.279251 0.195201 0.946954
0.0989379 -0.387646 -0.399544 0.286963 -0.625932 -0.613954
0.350006</inputs>
<outputs>-0.166817 0.0377938 0.0972221 -0.15462 0.807336
0.246236 -0.3278 -0.275734 -0.166989 -0.570402 -0.590832
0.220164</outputs>
```

[11] http://qt-project.org/doc/qt-4.8/qslider.html
[12] http://qt-project.org/doc/qt-4.8/QPushButton.html
[13] http://laral.istc.cnr.it/laral++/nnfw/api/namespacennfw.html

```
<outfunction type="ScaledSigmoidFunction">
<lambda>1</lambda>
<min>-1</min>
<max>1</max>
</outfunction>
<biases>-0.569158 -0.0649201 1.3824 0.922145 0.497208
0.270856 -0.494826 -0.594868 -0.146624 -0.311014 1.13855
0.104204</biases>
</cluster>

<linker from="inL" type="DotLinker" to="outL" name="link">
<weights>-0.943444 -0.297059 -0.813026 -0.763109
0.843994 0.212695 0.822901 0.336799 -0.581674
0.787711 -0.551788 0.604551</weights>
</linker>

<inputs>inL</inputs>
<outputs>outL</outputs>
<order>inL link outL</order>

</neuralnet>
</nnfw>
```

To understand the above extract it must be taken into account how NNFW implements neural network. This library decomposes any neural network in sets of *clusters* and *linkers*. Clusters are groups of neurons that share the same transfer function. Linkers are the connection weights linking two or more clusters together. In the example published we can see the definition of a cluster, of a linker, and the setup of the proper update order for the entire network (*i.e.* the direction of the data flow).

5.1.1 Common features of all simulation setups

The simulator introduced in the previous section has been used to implement several different experimental setups. In this paragraph we describe the main characteristics they share while we will analyse in details the individual differences when introducing the different scenarios later on.

The computer model has been built with distributed control and group behaviour in mind. This is reflected in the employment, in every experimental scenario, of MAV teams (with a fixed group size of four) where each individual is endowed with its own controller, which is identical to the ones used by its teammates.

The domain where the simulations take place is a rectangular area with size $710x760$ pixels (px) which is a 2D representation of the Canary Wharf financial district in London. MAVs are modelled as green squares with a side length of $2px$.

The fact that the flight behaviour we are simulating is a fixed-wing airplane-like motion adds the constraint for the MAVs of always being in movement. The speed is assumed to be constant: during each time step all of the simulated aircraft move, sequentially, of a certain amount of pixels along their heading.

The task the MAVs have to perform resembles a classical "search and hit" scenario. At the beginning of a test, an "enemy" target (represented as a red square with a side length of $1px$) is deployed in a certain position inside the environment. Starting from the four corners of the environment and facing its centre, the MAVs have to find their way to the target. To conclude the task successfully, one of the aircraft needs to perform a certain operation (represented as the activation of a Boolean output unit of its controller) when it is close enough to the target ($2.48px$ or less), in which case the latter would be considered "hit".

A test ends when the target has been hit or no MAV within the team currently tested is operative anymore. A MAV will stop flying (thus becoming non-operative) if it activates its "end-operation" Boolean unit (which can therefore only be activated once per test by each MAV), if it attempts to cross the environment boundaries, if it collides with a teammate (in which case both the MAVs would become non-operative), or if it runs out of energy. The decision to limit the number of times

the aircraft can activate their "end-operation" neurons to just one has been made in order to ensure the task requires a high degree of precision.

The behaviour of the MAVs is governed by an embedded autonomous controller, implemented as a neural network (see section 5.2.) This type of controller processes the information available to generate a yaw manoeuvre to be executed by the controlled aircraft. The information is related to the position of the target, and is received in form of relative polar coordinates by the MAVs, *i.e.* as distance and steering angle required to align with the target. In other words, the robots are not capable of what is called Automatic Target Acquisition (ATR). Because of this they do not need to execute such an intensive computational task (even if the job could be effectively tackled cooperatively, as demonstrated by Dasgupta [84]) and can dedicate all the available computational resources to alternative tasks. Our assumption is based on the presence of a satellite system that constantly monitors the target and broadcasts real-time information about its position to all the team members. In this way the MAVs, equipped with a GPS receiver that allows them to compute their position and an absolute heading, can easily calculate their distance from the target matching the two data sources gathered. A simple compass can also allow the MAVs to determine the relative direction of the target. It must be noted that this hypothesis has been made in order to simplify the development of the model and it is not to be considered in contrast with the idea of a reactive non-centralised strategy. As we have discussed in previous chapters, one of the advantages of reactive strategies over those based on deliberative or replanning approaches is based upon the fact that they implement a distributed control system. As a result of this the MAVs are not dependent upon a central element. Many alternative hypotheses could be postulated, thus eliminating the need of a system with an "bird's eye view" over the entire scenario. For example, the system we have developed would work in the same way if we imagine that no GPS systems are available, but the MAVs perceive instead a radio signal emitted by the target or by a transmitter placed nearby the target.

At the end of every generation the simulator saves a series of statistics into

distinct files:

- *Alive*: average number of MAVs per team alive at the end of a test;
- *Collided*: average number of MAVs collided with each other during a test;
- *Completion attempts*: average number of MAVs activating their end-operation unit during a test;
- *Crashed*: average number of MAVs crashed against a building during a test;
- *Energy remained*: average amount of energy left to the MAVs when a test ends;
- *Fitness (average)*: average fitness value for the entire population;
- *Fitness (maximum)*: best fitness value across the entire population;
- *Out of bounds*: average number of MAVs that attempted to exit the boundaries of the environment during a test;
- *Out of energy*: average number of MAVs running out of energy during a test;
- *Success rate (overall)*: overall percentage of tests concluded successfully by the individuals of a given generation;
- *Success rate (maximum)*: percentage of tests concluded successfully by the best team in a given generation;
- *Target distance (average)*: average distance between the target and the MAV which was the closest when it activated its end-operation unit during a test (average value for the entire population);
- *Target distance (minimum)*: average distance between the target and the MAV which was the closest when it activated its end-operation unit during a test (minimum value for the entire population).

A script automatically generated by the simulator imports all the statistics into the Matlab®[14] environment, ready for plotting and further analysis.

[14] `http://www.mathworks.co.uk/products/matlab/`

5.2 Neural network controllers

The autonomous controllers have been implemented as neural networks (as the one depicted in Figure 5.3), for the most part feed-forward ones.

The networks are fully connected, meaning that all the neurons in one layer have synaptic connections to all the neurons in the following layer. Since the network is a made up of thee layers (or two layers according to the interpretation provided by other authors) this refers to the connections from the input to the hidden layer, and to those that connect the hidden with the output layer.

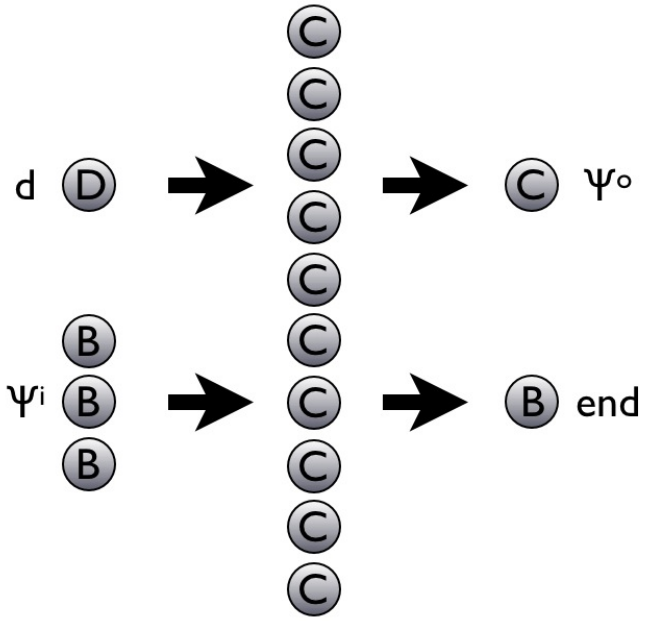


Figure 5.3: Example topology of a typical NN controller. In this case using four input neurons (one for encoding in a discretised way the MAV-target distance, three for the MAV-target angle), 10 units in the hidden layer, and two output neurons (yaw and "end-operation" respectively) [D: discrete, C: continuous, B: Boolean]

It is worth highlighting how all the neurons employed in this network just use summation as aggregation function. It means that the activation function $f(x)$ for each neuron can be formalised according to Equation 5.1, where b_0 represents the bias associated to the output neuron, n is the number of neurons that contribute to the activation of the given one, x_i is the activation value of the i-th neuron, and w_i is the weight associated to the connection between the i-th neuron and the output one. $g()$ is generic indicator of a further transformation operated on the sum of all the inputs, which depends on the specific neuron under examination.

$$f(x) = g(b_0 + \sum_{i=0}^{n} x_i w_i) \tag{5.1}$$

The input neurons are fully connected to the neural network hidden layer, made of a certain number of continuous neurons (which differs in the various setups elaborated) activated through a log-sigmoid function (slope 1.0) of which the output values are in the range $[-1.0; 1.0]$ (see Equation 2.14).

The neural network output layer consists of just two neurons, receiving incoming connections from the neurons belonging to the hidden layer only. One output unit is continuous and gets activated through the same log-sigmoid function as the neurons of the hidden layer. This neuron is in charge of controlling the flight of the MAV, generating yaw manoeuvres to be executed by the embedded autopilot system. The value generated is scaled according to the estimated maximum turn-rate of the aircraft simulated. For example, if we assume the turn-rate of the MAV being $+/-20^\circ$ in the time unit, the value generated by this output neuron, originally in the $[-1.0; 1.0]$ range, must be translated into the range $[-20.0^\circ; 20.0^\circ]$. The other output unit is a Boolean neuron (activated through a step function with a 0 threshold) which, when it turns to 1, causes the MAV to activate the "end-operation" procedure.

As stated before, the input arrives both from sensors embedded on the aircraft and by an overall monitoring system. The next section introduces the issue of the encoding, which will be further analysed when describing the first set of simulations carried out.

5.2.1 Encoding of the input information

The various experimental setups involve different types of information available to the aircraft in order to process its flight manoeuvres. This means that different neural network structures will be used as well. This is because of the different information provided to the network, partly because of the different levels of computational capabilities required to the controller to compute the data received.

As neural networks work at their best when the input used reflects certain char-

acteristics [280], a key aspect of making a model such as the one we have developed successfully relies upon using the proper type of encoding. As we will see later, the first set of Simulations (labelled "A") is specifically aimed to identify the right encoding system.

5.3 Evolutionary algorithm

The evolution towards a controller able to perform the desired task is made possible by the use of a genetic algorithm (see section 2.4.5) integrated into the computer simulator developed.

An initial population of 100 controllers is created with randomly assigned connection weights and biases, uniformly distributed within the [-1.0; 1.0] range[15]. Each controller works at a team-level, *i.e.*, it is assigned to 4 MAVs that together form a team. The genome consists of a vector of real values directly encoding connection weights and biases values, so no binary genomes (as it is usual instead in GA literature) are employed. Every controller is tested a certain number of times with the target deployed in different positions and the MAVs starting with varying positions/headings.

At the end of each generation (*i.e.* when all 100 controllers/teams have been tested) the 20 individuals that have performed the best scores according to the specific fitness function used are selected for reproduction. Each of the selected controllers generates 5 copies of its genome, on which the mutation operator is then applied. Each gene of the copied genome is modified, with probability 0.25, by a amount within the $[-0.5; 0.5]$ uniform probability distribution. The only exception is for the best individual of the current generation, which generates a copy of its genome without any modifications (according to a procedure called "elitism" [87]). The resulting 100 individuals will constitute the new population of the next generation. The evolutionary process lasts for a fixed number of generations and it is repeated several times with the results coming from all runs averaged together in order to

[15]The proper setting of the values that connection weights and biases can assume at the beginning of the evolutionary process is critical in order to avoid the so-called "bootstrap issue" [264].

obtain more robust data about the generated trends.

5.4 Experiments

This section illustrates the four main scenarios that have been tested, as well as the details of each experimental setup. The four scenarios are of increased complexity in the sense that they require increasingly sophisticated controllers for the MAVs to succeed. In the first one a static target is deployed inside an obstacle-free environment and one of four MAVs has to navigate to it and perform a certain task once got there (generalised as the activation of a "end-operation unit"). The second setup is similar to the previous one, except for the presence of obstacles that the MAVs can detect and have to avoid. In the third scenario the target is capable of moving away from an approaching MAV, which has in turn to chase it. In the fourth and final experimental setup two MAVs are required to approach the target (which can be either static or capable of moving) at the same time in order for the test to succeed.

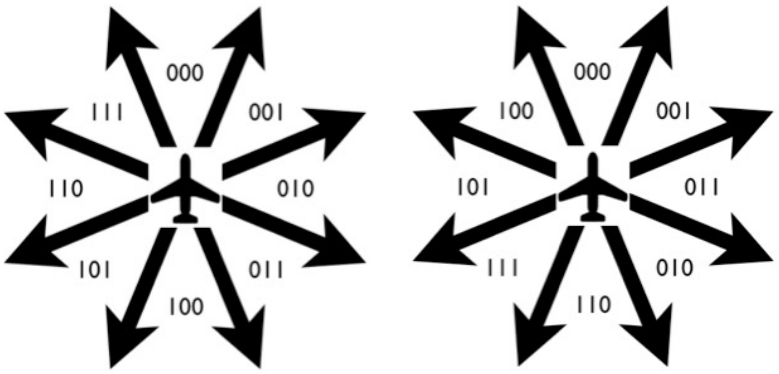
An additional set of simulations, that do not introduce a new experimental setup but just some modifications on the functioning of the evolutionary algorithm, will be described towards the end of the chapter

5.4.1 Basic scenario (simulations A)

The first set of experiments carried out, reported in Ruini *et al.* [319], aims to identify the most appropriate encoding for the sensorial input. Eight different simulations (A1-A8) have been elaborated, testing various encodings and the related neural architectures. The reference environment is the one described at the beginning of this chapter and no obstacles are present.

In simulation A1 the distance between the MAV and the target, which is measured in pixels by the simulator, is discretised according to the conversion criterion outlined in Table 5.1, which is valid for simulations A1, A2, A3, and A4.

As the table shows, and somehow counterintuitively, short distances are associated to high activation values and vice-versa. This solution is often employed in Evolutionary Robotics (see for example [224] or the works cited in [280]) in order to



(a) (b)

Figure 5.4: Discrete encodings of the MAV-target angle (assuming the aircraft facing North): (a) using a standard Boolean representation of the sub-spaces; (b) using Gray code instead

Table 5.1: Discretised encoding of the MAV-target distance

Distance (px)	Discretised value
$1,040 \geq d > 900$	0.0
$900 \geq d > 800$	0.1
$800 \geq d > 700$	0.2
$700 \geq d > 600$	0.3
$600 \geq d > 500$	0.4
$500 \geq d > 400$	0.5
$400 \geq d > 300$	0.6
$300 \geq d > 200$	0.7
$200 \geq d > 100$	0.8
$100 \geq d > 2$	0.9
$2 \geq d \geq 0$	1.0

make more likely the evolution of certain behaviours. A common example concerns obstacle-avoidance behaviours. In that case the designer generally wants the neural network to generate a motor response which is stronger as closer an obstacle is to the robot. Obtaining that is easier if being close to an obstacle feeds a "high" value into the neural network controller and the power of the motor response positively correlates with high network outputs. In this case things are not much different as we want the MAVs to activate a specific Boolean unit (*i.e.* to bring its activation value to 1) the closest as possible to the target. The very same principle is also at work with many robot sensors. The infra-red sensors installed on the Khepera and e-puck robots, for example, emit beams of infra-red light and measure the quantity

of reflected light, which is higher the closer the robot is to an obstacle.

The space surrounding each MAV is divided into 8 sub-fields depending on its current heading (see Figure 5.4). The first one includes all the angles equal to or greater than 347.5° and lower than 22.5°; the second one comprises all the values between 22.5° (included) and 67.5° (excluded), and so on. These sub-fields are numbered progressively according to a Boolean encoding, as shown in Figure 5.4(a). At any time, the direction in which the target lays can be matched with one of these sub-spaces. The three-digit value representing that portion of the space is fed into the neural controller via an equal number of neurons.

In simulation A2 the encoding of the angle is the same as in simulation A1, but how it varies is the way in which the sub-fields are numbered, not according to a Boolean one anymore, but to a Gray code-based [136] instead (see Figure 5.4(b)).

In simulation A3 the angle is encoded through two continuous values in the $[-1; 1]$ range, respectively corresponding to its sine and cosine (a redundant approach successfully employed in many researches, as for example in [34] and [378]).

In simulation A4, the same angle is represented instead by means of a single neuron that assumes the raw original value included between 0° and 360°, after its normalisation into the $[0; 1]$ continuous range (0° remains 0, 360° becomes 1).

Simulations A5, A6, A7, and A8 are "copies" of Simulations A1, A2, A3, and A4 respectively, but they encode the MAV-target distance not in a discretised way, rather normalising it into the $[0; 1]$ range (where 0 corresponds to the maximum distance possible, which is $1,040$, and 1 to the minimum one).

Table 5.2 summaries the distinctive characteristics of simulations A1-A8.

The evolutionary algorithm runs for 500 generations. The teams/controllers are tested four times each, with the individual MAVs starting every test with a storage level amounting to $5,000$ energy units (EU). When flying their energy consumption amounts to 3EU per time step, which allows them to move $2.14px$ along their heading, the turn-rate per time step is $[-10°; 10°]$ instead. At the beginning of each test the target is deployed in a different position across the environment, while the MAVs use fixed starting positions but have slightly different initial headings from

Table 5.2: Summary of the main characteristics of simulations A

Simulation	MAV-target distance encoding	MAV-target angle encoding
A1	Discrete	3-digits (Boolean)
A2	Discrete	3-digits (Gray code)
A3	Discrete	Continuous
A4	Discrete	Sine/cosine
A5	Continuous	3-digits (Boolean)
A6	Continuous	3-digits (Gray code)
A7	Continuous	Continuous
A8	Continuous	Sine/cosine

test to test. Table 5.3) shows all the details about the initial setup of the various tests.

Table 5.3: Simulations A: initial deployment of MAVs and target (0° is considered the heading for a MAV facing North, then the angle is measured clockwise)

Agent	X coord.	Y coord.	heading (α)
Target	$[0; 710]$	$[0; 760]$	N/A
MAV_0	20	20	$135° \pm 10°$
MAV_1	690	20	$225° \pm 10°$
MAV_2	20	740	$45° \pm 10°$
MAV_3	690	740	$315° \pm 10°$

Five evolutionary runs (*i.e.* reinstatements of the evolutionary process with a different seed provided to the random number generator, RNG), each of them 500 generations long[16], have been run and the results obtained were averaged together.

The fitness function that has been used to evaluate and compare the controllers used by the different teams is shown in Equation 5.2, where: α is the average distance (measured in pixels) between the target and the team member who has activated its "end-operation" neuron the closest to it in any test (that has been calculated based upon the four tests performed); β is the average amount of energy retained by the MAV (also calculated based upon the four tests); σ is the number of tests concluded successfully by the given team; ϵ is the total number of MAVs still operative after

[16]In order to determine the number of generations to evolve we have not relied on any theoretical notion. Rather we have preferred to stick to an empiric approach, *i.e.* to carry out some preliminary analysis, looking at how long it takes to the average and maximum fitness to reach a stable state, and then using this (or a slightly higher) number of generations.

the four tests have finished (maximum 12, as at least one of the MAVs must become non-operative for a test to be considered successful.)

$$f = -\alpha + \frac{\beta}{50} + 50\sigma + 5\epsilon \tag{5.2}$$

Amongst all of these parameters, the two with the highest relative importance at the beginning of the evolutionary process are α and σ. The fitness function used generally returns negative values for the individuals of the first generations, as they tend either to do not activate their "end-operation" unit (which causes α, in case no MAVs activate their neuron, to be set on the maximum value possible given the environment size, *i.e.* $1,040$) or to do so when far from the target (making α assuming values higher than 0, not compensated by the positive parameters of the formula). Once selective pressure makes the controllers capable of approaching the target effectively, the σ parameter kicks in, advantaging those MAVs activating their end-operation unit when close enough to the target (*i.e.* within a $2.48px$ distance). Finally, when most of the controllers can successfully reach the target and perform the required operation at the proper distance, an additional source of selective pressure is given by parameters β and ϵ, whose relative importance increases favouring teams that can perform the task in the most effective way possible (*i.e.* consuming the least amount of energy possible and with no unnecessary "losses").

Results

We present in detail the results obtained by the second of the simulations that were run - A2 - just to give an example of how the statistics listed in paragraph 5.1.1 are collected and how they should be interpreted.

Figure 5.5 shows the average and maximum fitness values registered across the generations[17]. Both lines follow a similar growth pattern, although with different steepness. The average fitness increases dramatically over the first 50 generations,

[17]The attentive reader should notice that the two lines plotted in the graph do not consist of 500 data points each. To make the graphs more clear we have decided to only plot 50 data-points in total: the first one corresponds to the average of the first 50 data-points, the second one to the average of the $[51st; 100th]$ data-points interval, and so on. This approach will be followed across the entire thesis.

because of the quick evolution from controllers behaving randomly to neural networks approximately performing the task. Then the growth slows down until it stabilises when the 300th generation is reached. Looking at the second parameter, *i.e.* the fitness scored by the best individual in the population at a given generation, it can be seen how the best possible value in this experimental setup is reached in less than 50 generations and then maintained until the end of the evolutionary process thanks to the elitism operator. Of course, such a quick evolutionary dynamics means that the search space created by the fitness function we decided to use is fairly simple for the genetic algorithm to explore.

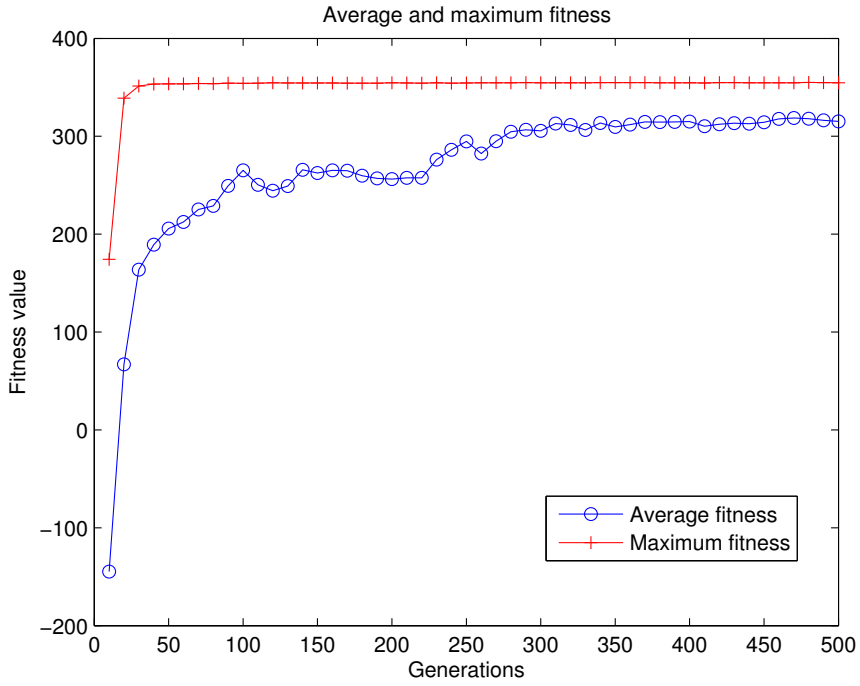


Figure 5.5: Simulation A2: average and best fitness (average of 5 evolutionary runs)

Figure 5.6 illustrates the overall percentage of tests concluded successfully at each generation. The shape of this curve resembles the one we have just seen in Figure 5.5 for the average fitness. This is easily understandable, as they are essentially two different ways of looking at the same phenomenon (which is not true for the maximum fitness statistics, as it tells us instead whether a team managed to complete all the tests successfully or not). At any generation a total of 400 tests is carried out, thus the final value obtained, about 95%, roughly corresponds to 380 tests successfully executed. There are no theoretical constraints preventing

this value from reaching the 100% threshold. This did not happen in this scenario most likely due to the high mutation rate used ($[-0.5; 0.5]$ on 25% of the connection weights/biases), which often introduces disruptive modifications in the controllers. The results have nonetheless been considered satisfying enough, considering that an aggregate 95% values indicates that most of the evolved controllers easily perform the required task 100% of the time. Reducing the mutation rate would most likely increase the amount of time required for the evolutionary algorithm to identify a solution, without providing at the end of the day any benefit (as even a single well working controller would be enough for our purposes).

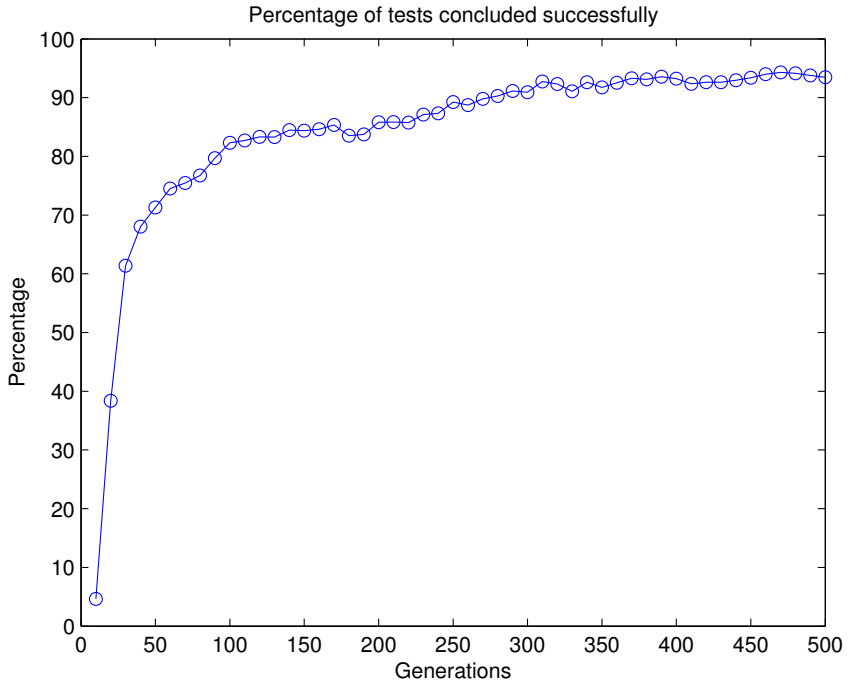


Figure 5.6: Simulation A2: percentage of successful tests (average of 5 evolutionary runs)

Figure 5.7 focuses on the moment in which MAVs activate their end-operation unit, showing the average and minimum distance from the target. As it can be seen, at the beginning of the evolution most of the controllers do not yet know how to use the end-operation unit, consequently they activate it more or less randomly. But thanks to the fitness function used, which rewards the careful use of that unit, the average distance decreases quite quickly, converging to the minimum one.

Figure 5.8 shows the amount of energy saved by the "average best" MAV at every generation. This statistic is collected by measuring the energy left to the first MAV

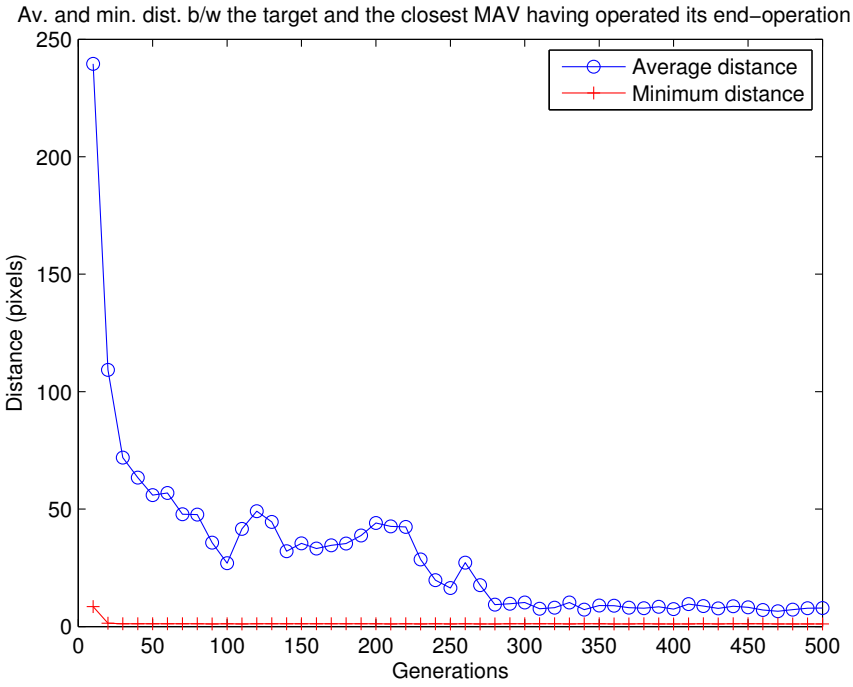


Figure 5.7: Simulation A2: average and minimum distance between the target and the MAV having operated the end-operation unit closest to it (average of 5 evolutionary runs)

to activate its end-operation neuron when close enough to the target during each test. Then the average value is calculated across the 100 teams. As expected, the plot shows a continuous increase in the performance of the controllers, that gradually use less energy to complete the tasks they are assigned. At the beginning of a test, the target is deployed in a random position in the $([0; 710]; [0; 760])$ coordinates range. On average the target will therefore be at the centre of the environment (in the point of coordinates $(355, 380)$), $488.36px$ far from the MAVs at the beginning of a test. The amount of $4,575EU$ left, shown by the graph, means that the best MAV in a team evolved for 500 generations uses on average 425 energy units to reach the target. Considering that $1EU$ allows a MAV to fly for $0.93px$, $395.25px$ is the distance traveled by the best MAV on average. It might look counter-intuitive that this value is lower than $488.36px$, but this can be easily explained considering that the statistics in Figure 5.8 take into account the best individuals at any test. Being the best during a given test, knowing that the controllers that all the MAVs within a team use are identical to each other, is often due to the luck of finding themselves close to the target when the test begins.

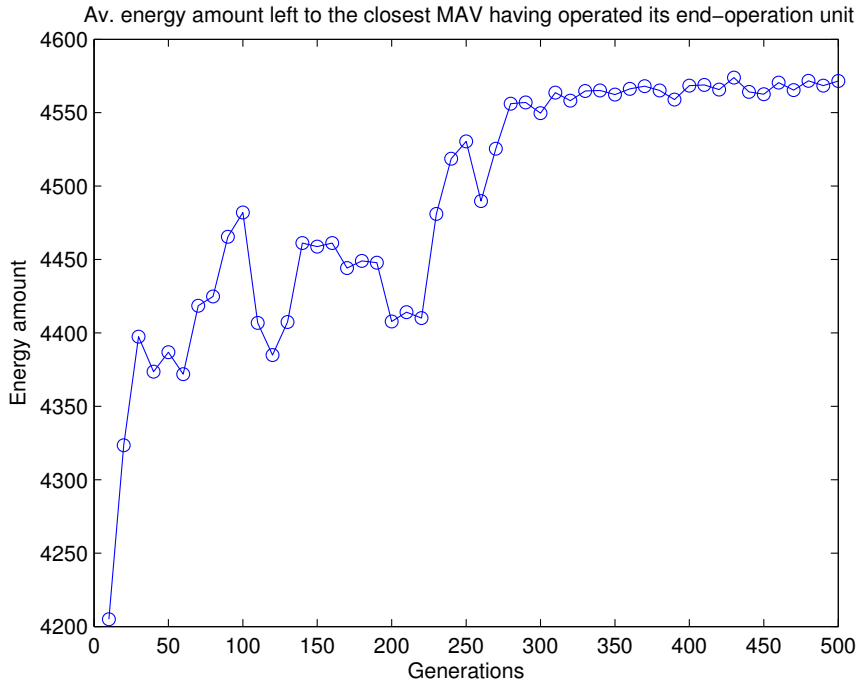


Figure 5.8: Simulation A2: amount of energy left to the MAV activating its end-operation output unit closest to the target (average of 5 evolutionary runs)

Finally, Figure 5.9 shows what is the state of the MAVs belonging to a team at the end of a "typical" test (which is the average of the 4 tests each team performs). The "ideal" situation would consist of having one single MAV having activated its end-operation unit and the other three still operative. The values shown in the plot differ slightly from this theoretical optimum. First of all, we can see that about 1.5 MAVs activate their end-operation neuron during the test. That means that, on average, two MAVs are required to have guarantee of success. Consequently the amount of aircraft still operative when the mission has been successful is lower than three. In this case it amounts to about 2.3. The remaining 0.3 is constituted by MAVs becoming non-operative for different reasons, mainly exiting the environment boundaries (which they can not perceive), or running out of energy. Furthermore, as the MAVs are not equipped with any sensors to detect teammates, a small percentage of collisions (which would mainly depend on the size of the reference environment and it is relatively small in our scenario) is natural.

We can now perform a comparison between the eight experimental setups analysed in order to identify the best encoding to be used for this model. Among all of those listed in the previous section, three of the most significant statistics are the av-

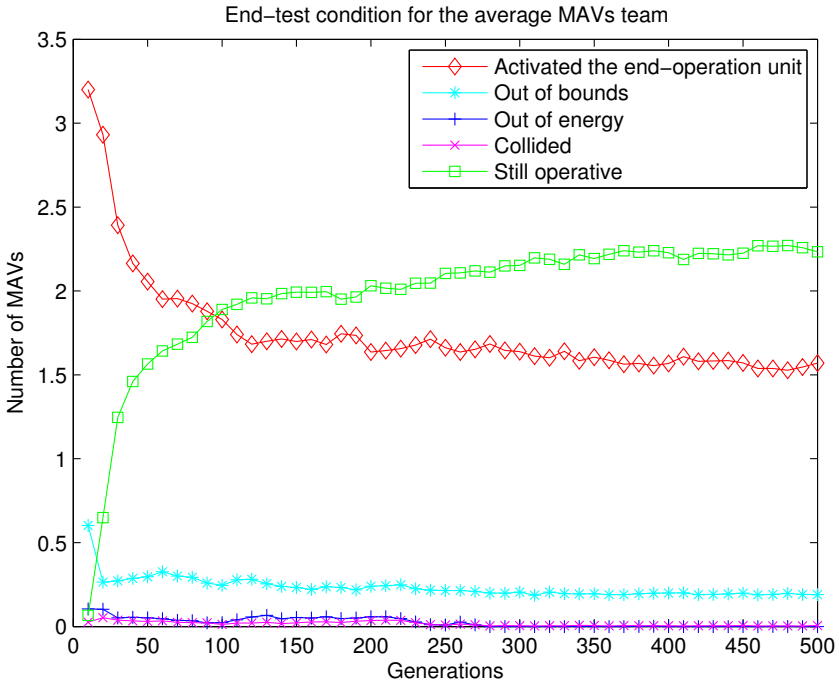


Figure 5.9: Simulation A2: condition of the MAVs at the end of the "average test" (average of 5 evolutionary runs)

erage and maximum fitness, as well as the percentage of tests concluded successfully. Our analysis therefore focuses on a comparison of those.

Figure 5.10 contrasts the average fitness values obtained in the different scenarios. The same colour, but with different markers, has been used to plot the curves related to simulations that share the same angle encoding but use a different system for the distance (A1 and A5, A2 and A6, A3 and A7, A4 and A8). Looking at the graph it is possible to see that Simulations A2 and A6 perform definitely better than all the others, scoring at least 100 points more than their closest competitors. Looking more generally, a trend can be identified according to which architectures A1, A2, A3, and A4 seem to perform better (or at least equal) than A5, A6, A7, and A8 correspondents. This suggests that the discretised encoding of the MAV-target distance makes the task easier to the neural controller. If the two kinds of encoding score similar results with regard to A1-A5 and A3-A7, the advantages are more evident contrasting A2 with A6, and A4 with A8. The comparison between A4 and A8 also highlights two very different evolutionary patterns for which the author does not have a proper explanation to offer.

The graph about the maximum fitness (Figure 5.11) provides less informative

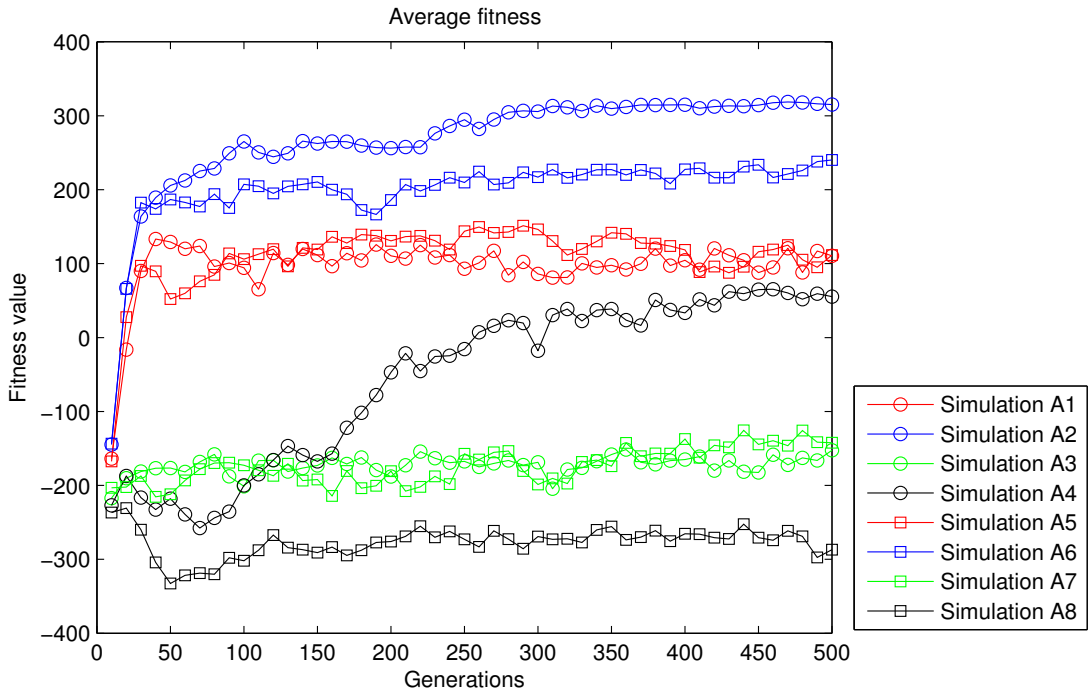


Figure 5.10: Simulations A: comparison for the average fitness (average of 5 evolutionary runs)

contents than the one we have just analysed, as four different architectures (A1, A2, A5, and A6) equally manage to reach the maximum level of performance (about 350 points). Neural controller A3, A4, A7, and A8 seem instead to struggle, registering significantly lower values.

Figure 5.12 confirms the findings we have been discussing so far, suggesting that architecture A2 and A6 are those overall performing in the best way, with an average success rate (percentage of tests concluded successfully) approximately equal or greater than 90%. Decent results are obtained by A1 and A5 as well (about 75% of success rate at the end of the evolution) while A4 does not reach the 60% threshold. The remaining three neural architectures, namely A3, A7, and A8 score pretty marginal results.

Table 5.4 presents in a numerical form the same data presented in Figures 5.10, 5.11, and 5.12, focusing on the values at the end of the evolutionary process only (average over the last 10 generations).

On the basis of these results (the best overall success rate achieved and the highest average fitness, combined with a general trend suggesting the validity of

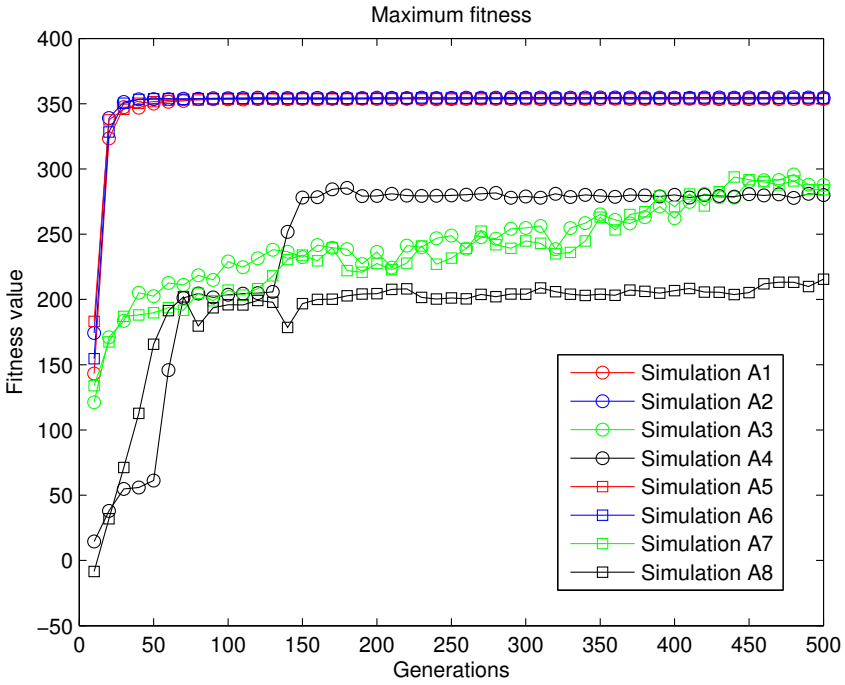


Figure 5.11: Simulations A: comparison for the best fitness (average of 5 evolutionary runs)

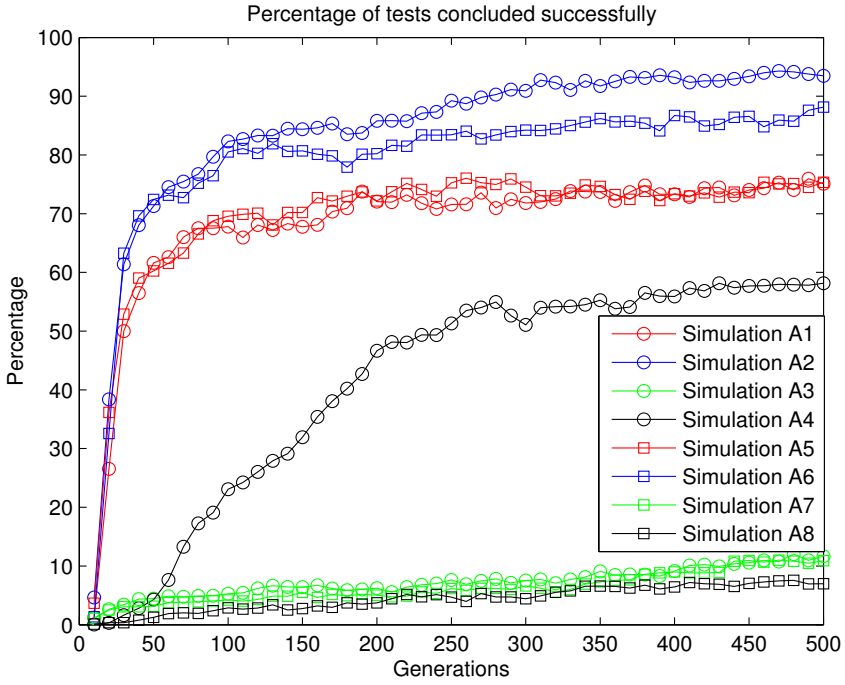


Figure 5.12: Simulations A: comparison for the success rate (average of 5 evolutionary runs)

Table 5.4: Simulations A: resume of the main results (average of the last 10 generations, based on 5 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
A1	110.49	353.67	75.09	98.83
A2	315.18	354.56	93.46	100
A3	−152.46	287.64	11.68	32.59
A4	55.32	279.99	58.14	97.52
A5	111.66	353.82	75.33	99.02
A6	240.19	354.19	88.14	100
A7	−142.2	284.11	10.89	29.91
A8	−287.03	215.53	6.97	16.4

encoding the MAV-target distance in a discretised way), we have promoted the architecture used in Simulation A2 as the one to be used as a basis for the following extensions of the model.

Although, we should not overlook the fact that several architectures generated a very good result for what concerns the best success rate, equal or at least very close to 100% for five of the eight controllers tested. Our analysis has mainly focused on the average values as these reflect very well how quickly and easily the evolutionary process could find a solution to the problem faced, thus allowing us to make clear distinctions between the different topologies. But if we look at the problem with "more practical" eyes, there are five controller architectures (A1, A2, A4, A5, and A6) that can cope pretty effectively with the task. exceeding the 90% threshold we have defined in previous chapter.

5.4.2 Obstacle avoidance (simulations B)

The second set of experiments [319] introduces a tight grid of obstacles into the environment. The obstacles act as no-fly zones for the MAVs, thus making the task the teams have to tackle slightly more complex as the aircraft have to add obstacle-avoidance behaviour to their target tracking capability.

Deployment of the obstacles

Eighteen obstacles have been deployed inside the environment in a way that matches the location of the main buildings present in the Canary Wharf area taken as a reference.

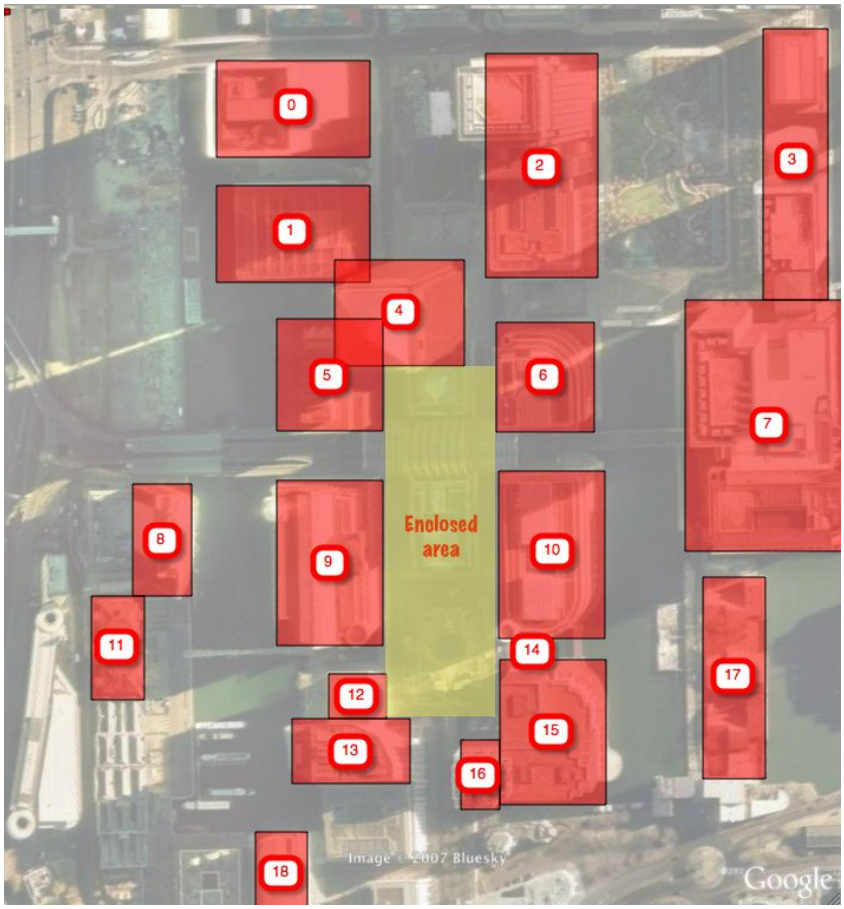


Figure 5.13: Overview of the obstacles distributed along the environment

The coordinates of all obstacles are denoted in Table 5.5 and their location is graphically shown in Figure 5.13. It is worth mentioning (as we will sometimes refer to absolute polar coordinates later on in this chapter) that the $(0, 0)$ point lies in the top-left corner of the figure. The coordinates of a specific "enclosed area", that MAVs can access only executing a "proper" obstacle-avoidance behaviour, is portrayed in Figure 5.13 with a yellow colour can be seen in Table 5.6.

Selection of the proper sensors configuration for obstacle avoidance

In order to be able to avoid the obstacles, the MAVs need a sensorial apparatus that enables them to perceive those and to react accordingly. The solution we

Table 5.5: Simulations B: (X, Y) coordinates of the obstacles present in the simulated environment

Obstacle	Width	Height	Starting X	Starting Y	Final X	Final Y
0	130	82	180	44	310	126
1	130	82	180	150	310	232
2	95	190	408	38	503	228
3	55	230	643	17	698	247
4	110	90	280	213	390	303
5	90	95	231	263	321	358
6	83	93	417	266	500	359
7	133	213	577	247	710	460
8	50	95	109	403	159	498
9	90	140	231	400	321	540
10	90	142	419	392	509	534
11	45	88	74	498	119	586
12	49	38	274	564	323	602
13	100	55	244	602	344	657
14	35	18	430	534	465	552
15	90	123	420	552	510	675
16	33	59	387	620	420	679
17	53	171	592	482	645	653
18	44	65	213	698	257	763

Table 5.6: Simulations B: (X, Y) coordinates of the enclosed area

Width	Height	Starting X	Starting Y	Final X	Final Y
84	289	328	308	412	597

have decided to use consists of a set of simulated ultra-sonic (US) sensors installed onboard the aerial vehicles. Two questions immediately arise concerning how many of these sensors to use on each MAV and in which direction to point them. To address these two issues, the performances generated by four different configurations of sensors (see Figure 5.14) have been compared in order to identify the best suited one. The four configurations correspond to the number of simulation setups studied, labelled B1, B2, B3, and B4 respectively.

In configuration B1, one single forward-facing US sensor is employed (see Figure 5.14(a)). Three sensors are used in setups B2, B3, and B4 instead. Configuration B2 relies on one forward-facing sensor and two respectively oriented at $+45°$ and $-45°$ respect to the aircraft heading (see Figure 5.14(b)). Setups B3 and B4 (Fig-

ures 5.14(c) and 5.14(d) respectively) vary the orientation of the two side-looking sensors: $\pm 30°$ in B3 and $\pm 20°$ for what concerns B4.

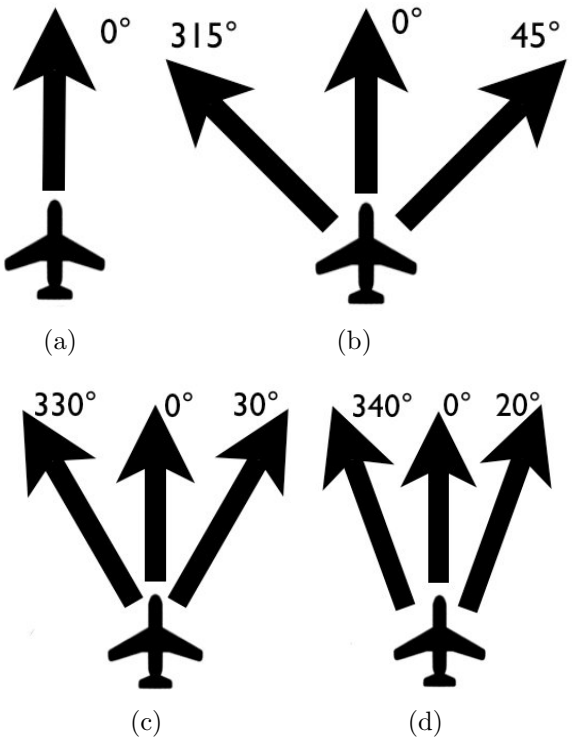


Figure 5.14: The four different configurations of ultra-sonic sensors tested on the simulated MAVs: (a) one single forward-facing US sensors (simulation B1); (b) one forward-facing sensor, and two oriented at $\pm 45°$ (simulation B2); (c) one forward-facing sensor, and two oriented at $\pm 30°$ (simulation B3); (d) one forward-facing sensor, and two oriented at $\pm 20°$ (simulation B4)

Each sensor returns a numeric value, which represents the distance between the MAV[18] and the closest obstacle detected, according to the discretisation criterion illustrated in Table 5.7. The maximum distance the ultra-sonic sensors can spot obstacles at is $30px$[19] and the readings are not affected by noise. Categories of obstacles that MAVs can spot are the target, the teammates, the buildings, and the environment boundaries also[20].

[18]We assumed that the sensors are installed above the centre of mass of the aircraft rather than on its nose, thus slightly affecting the readings at close range.

[19]Considering that in these simulations a MAV is $2px$ long, assuming $1px \approx 25cm$, $30px$ can be roughly assimilated to $7.5m$. Considering that inexpensive ultra-sonic sensors on sale on the private market can easily cover up to $3m$ (see for example those sold by Parallax, `http://www.parallax.com/tabid/768/ProductID/92/Default.aspx`) simulating more accurate sensors capable of reading twice as far should not be considered a wrong design assumption.

[20]Allowing the MAVs to detect the environment boundaries is a choice made in order to accelerate the evolutionary process avoiding that otherwise well performing MAVs could fail some tests due to exiting the (invisible) boundaries of the reference environment.

Table 5.7: Simulations B: values returned by the simulated ultra-sonic sensors depending on the distance to the closest obstacle perceived

Distance [px]	Discretised value
$d > 30$	0.0
$30 \geq d > 27$	0.1
$27 \geq d > 24$	0.2
$24 \geq d > 21$	0.3
$21 \geq d > 18$	0.4
$18 \geq d > 15$	0.5
$15 \geq d > 12$	0.6
$12 \geq d > 9$	0.7
$9 \geq d > 6$	0.8
$6 \geq d > 3$	0.9
$d \leq 3$	1.0

Modifications to the neural network controller and to the simulation details

In order to accommodate and process the additional input information available to the MAVs in this scenario, the structure of the autonomous controller has had to be modified accordingly. Two new neural network topologies have been designed: one specific for Simulation B1, the other three equivalently working for B2, B3, and B4.

These two new topologies present an additional number of input neurons corresponding to the number of US sensors employed (1 in B1, 3 in B2, B3, and B4). To process this information, more neurons have been inserted into the hidden layer, which now contains 12 units in B1 and 15 in the alternative topology.

Figure 5.15 shows, from a graphical point of view, the topology of the neural network controller employed in Simulations B2, B3, and B4.

For what concerns the general simulation details, to compensate the increased difficulty of the control problem the navigation task has been made slightly easier, increasing the MAVs turn-rate per time step (from $[-10°; 10°]$ to $[-20°; 20°]$), and reducing the amount of movement performed in the same time unit (from $2.81px$ to $2px$). The energy consumption has been reduced accordingly, from $3EU$ to $2.14EU$.

The fitness function (reported in Equation 5.3) has been subjected to minor changes with respect to Equation 5.2 by increasing the relative importance of the ϵ

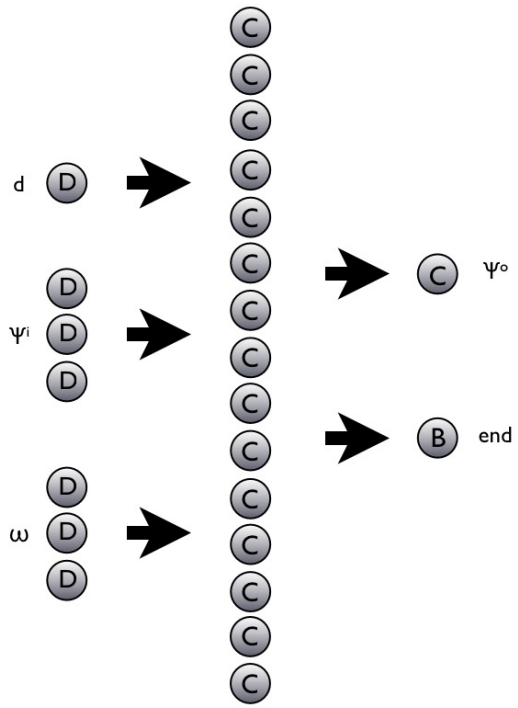


Figure 5.15: Graphical representation of the NN controller used in simulations B2, B3, and B4. Its topology consists in 7 input neurons (1 for encoding the MAV-target distance, 3 for the MAV-target angle, 3 for the ultra-sonic perception,) 15 units processing the information in the hidden layer, and two output neurons (yaw and "end-operation" respectively) [D: discrete; C: continuous; B: Boolean]

factor (thus stressing the importance of performing the obstacle-avoidance behaviour properly). This function still operates as the previous one, leading the evolutionary algorithm to evolve the controllers step-by-step.

$$f = -\alpha + \frac{\beta}{50} + 50\sigma + 10\epsilon \tag{5.3}$$

Each MAV team is tested four times with the target deployed in randomly selected positions within specific areas. Twice the target will be inside the "enclosed area at the centre of the environment, surrounded by buildings and with narrow entrances. Twice it will be deployed instead outside that area, on a point not closer than $5px$ to an obstacle or to the environment boundaries. In order to increase the generalisation capabilities of the model, apart from starting each test with different headings, MAVs are now deployed in different positions. Table 5.8 shows the new deployment rules used in this set of simulations.

Finally, the evolutionary process has been extended in duration and it now lasts for $2,000$ generations rather than the previously used amount of 500, due to the more

Table 5.8: Simulations B: initial deployment of MAVs and target (0° is considered the heading for a MAV facing North, then the angle is measured clockwise)

Agent	X coord. [px]	Y coord. [px]	Heading (α)
Target (inside)	$[328; 412]$	$[308; 597]$	N/A
Target (outside)	$[0; 710]$	$[0; 760]$	N/A
MAV_0	$[21; 642]$	20	$135° \pm 10°$
MAV_1	690	$[461; 739]$	$225° \pm 10°$
MAV_2	20	$[21; 739]$	$45° \pm 10°$
MAV_3 (a)	$[21; 214]$	740	$315° \pm 10°$
MAV_3 (b)	$[258; 690]$	740	$315° \pm 10°$

complex task which requires more "time" for the evolution of a controller properly dealing with it.

Results

The main results obtained from this set of simulations are summarised in Table 5.9 for what concerns the three most critical parameters measured: average fitness, maximum fitness, and success rate.

Table 5.9: Simulations B: resume of the main results (average of the last 10 generations, based on 5 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
B1	51.09	397.44	61.65	97.81
B2	257.27	413.53	86.34	100
B3	238.09	413.08	84.27	100
B4	257.62	413.55	87.62	100

Figure 5.16 contrasts the average fitness values obtained in the four experimental setups. The first noteworthy observation that comes to the eyes is that simulation B1 is the one which clearly performs the worst in the overall group. This outcome was to some extent expected, since the employment of a single ultra-sonic sensor does not allow the MAVs to know in which direction to turn when facing an obstacle. This point is demonstrated by looking at Table 5.10 where it can be seen how simulation B1 is the one in which the "casualty rate" is the highest due to MAVs crashing against obstacles and attempting to exit from the environment boundaries

(respectively 0.99 against an average of 0.27, and 0.22 against 0.17). The other three architectures (B2, B3, and B4) seem to perform in quite a similar way with slightly more positive results provided by B4.

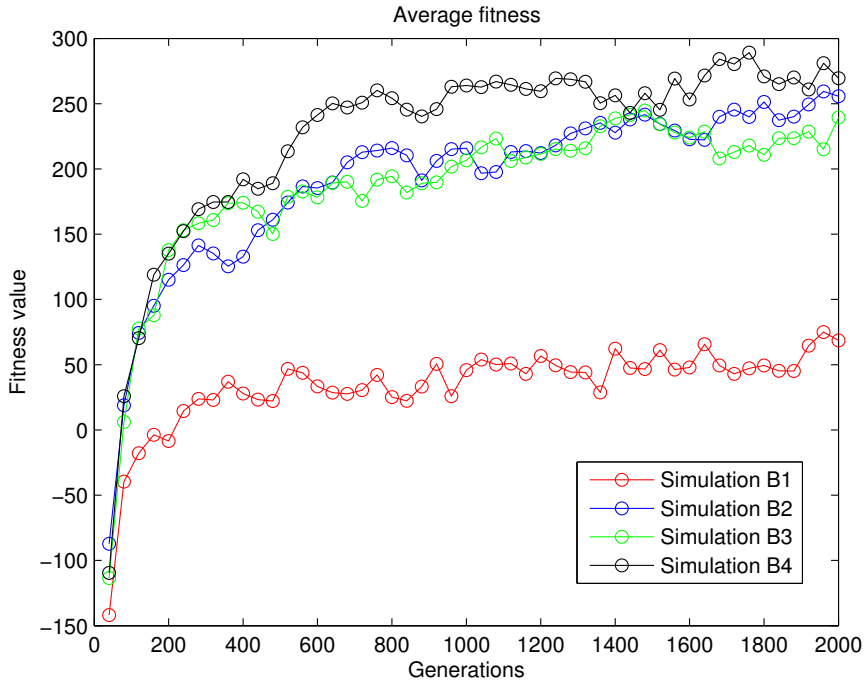


Figure 5.16: Simulations B: comparison for the average fitness (average of 5 evolutionary runs)

Figure 5.17 compares instead the fitness of the best individuals. Again, we can see how simulation B1 reaches well below average score. A maximum and homogenous level of fitness is reached instead by the other three setups in about 600 generations.

In Figure 5.18 the average success rate is reported, confirming the findings found so far. It is nonetheless interesting to highlight how the results scored by B2, B3, and B4, included in the $[80\%; 90\%]$ range, are just slightly below those obtained by the best controllers in the A group.

Given the similarity in the results obtained by B2, B3, and B4 (and, again, keeping in mind that the best success rate is 100% or close to it for all the experimental setups), in order to identify one of the architectures tested as the best one, we have therefore decided to look at the end-test statistics for the average team of each setup at the end of the evolutionary process (see Table 5.10 for a global resume). The advantage of Simulation B4 becomes evident. With respect to Simulations B2 and B3 there are a wider number of MAVs still operative at the end of test (2.3 vs. 2.17

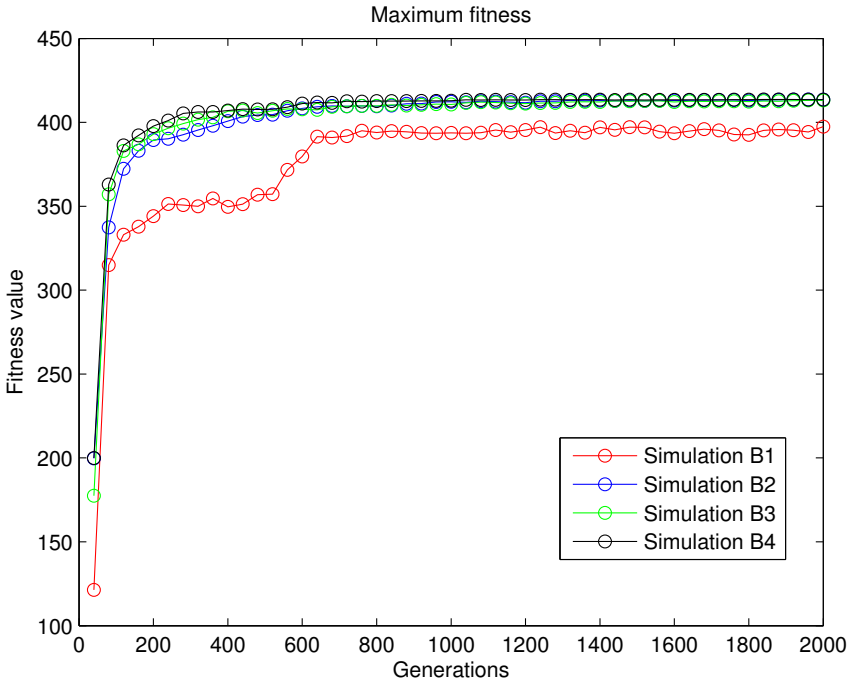


Figure 5.17: Simulations B: comparison for the best fitness (average of 5 evolutionary runs)

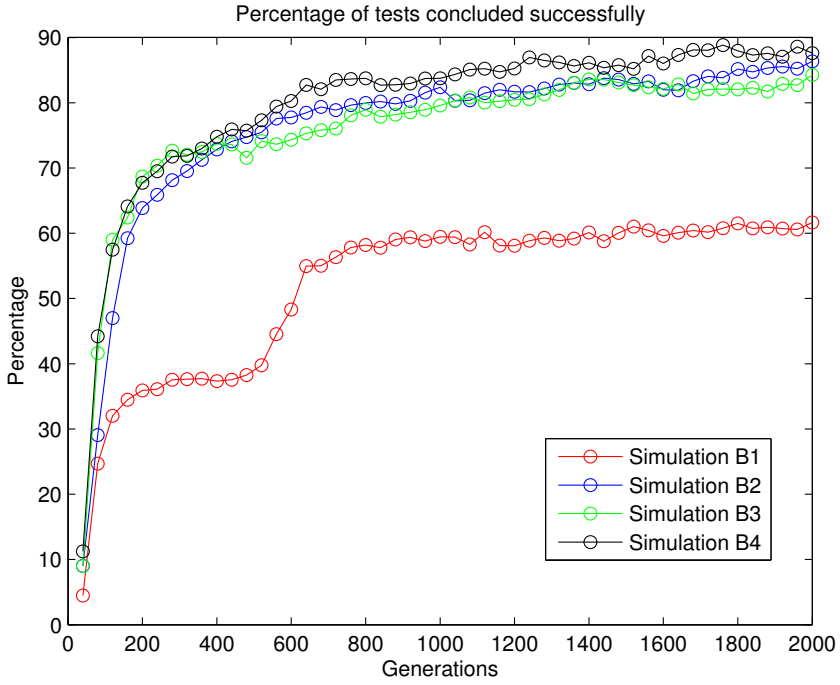


Figure 5.18: Simulations B: comparison for the success rate (average of 5 evolutionary runs)

and 2.2 respectively), mainly due to to better abilities in both obstacle-avoidance (on average, 0.22 MAVs crash against a building vs. 0.32 and 0.28 respectively) and flying within the environment boundaries (0.15 aircraft exit, on average, from the designated limits in Simulation B4, vs. 0.2 in B2 and 0.16 in B3). The only pitfall is the average amount of MAVs that run out of energy during the test: 0.23 for B4 vs. 0.18 for B2 and 0.27 for B3.

Table 5.10: Simulations B: condition of the MAVs at the end of the "average test" (average of 5 evolutionary runs)

Sim.	Still operative	Activated their Boolean output	Crashed against a building	Out of bounds	Out of energy	Collided
B1	1.18	1.22	0.99	0.22	0.32	0.05
B2	2.17	1.05	0.32	0.2	0.18	0.05
B3	2.2	1.03	0.28	0.16	0.27	0.06
B4	2.3	1.03	0.22	0.15	0.23	0.05

Compared to Simulations A, the proportion of MAVs that ran out of energy during a test was significantly higher for all of the B setups. Looking at the behaviour exhibited by the aircraft during the simulations it can be seen how this result is due to "loops" into which the MAVs may sometimes fall. A MAV enters a closed and narrow area of the environment and starts flying in a circle, getting stuck because its US sensors keep detecting the same obstacle pattern thus "instructing" the neural controller to continue performing the same manoeuvre. This is a problem that in Evolutionary Robotics happens frequently but typically affects computer simulations only. This phenomenon does not often reflect in reality (*i.e.* when the controllers evolved in simulation are transferred to real robots) because of two main reasons: the noise in the sensor readings and the non-regularity of the environment. These two effects together continuously create different sensorial patterns and, in turn, different behavioural responses that prevent the MAVs from ending up in a loop.

Generalisation experiments and technical improvements

In order to test the general applicability of the model developed, experiments concerning generalisation to a different environment have been carried out in collabora-

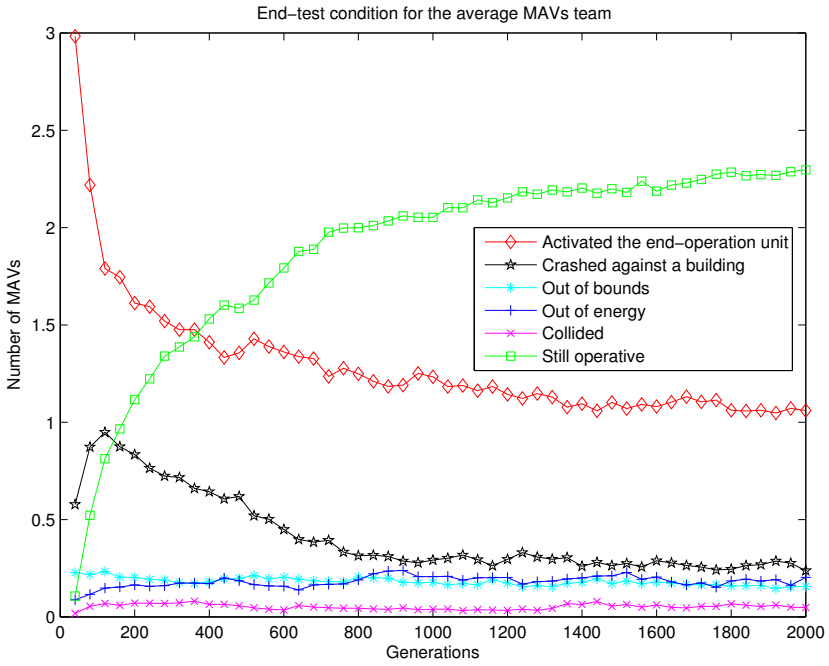


Figure 5.19: Simulation B4: condition of the MAVs at the end of the "average test" (average of 5 evolutionary runs)

tion with Franck Zetule, who published the results obtained in his MSc thesis [406]. The new experimental setup developed, based upon simulation B4, involves a smaller reference environment (600 by 600px rather than 710 by 760px), with a layout of obstacles inspired by the La Défense district in Paris, France (see Figure 5.20).

Apart from the new environment being smaller than the one originally used, the presence of high-density buildings and a consequent narrower enclosed area where the target is (alternatively) deployed has provoked some troubles for the genetic algorithm in evolving working controllers. In order to obtain a successful evolutionary process, it has been required to make some modifications of the original fitness formula. The fitness function has been adjusted as shown in Equation 5.4.

$$f = \frac{\upsilon}{8} - \alpha + \frac{\beta}{10} + 50\sigma + 10\epsilon \tag{5.4}$$

This formula differs from Equation 5.3 due to the smaller denominator applied to parameter β (10 instead than 50) and particularly for the introduction of the υ parameter, which represents the average difference between the distance of the MAVs from the target at the beginning and at the end of a test (*i.e.* measuring to

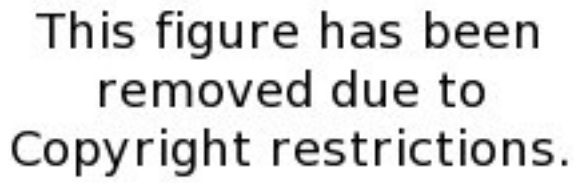


Figure 5.20: The 2D simulated environment used by Zetule for the generalisation experiments (Paris, La Défense district). Source: [406]

what extent the MAVs have gone close to the target during the testing).

Led by this function, the evolutionary process takes place smoothly as demonstrated in Figure 5.21. After 2,000 generations, the average percentage of successful tests for this experimental setup has exceeded the 85% level. At the end of the evolutionary process the best controllers have scored a 100% result.

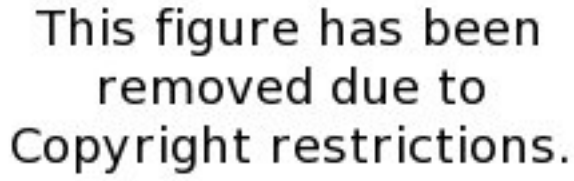


Figure 5.21: Zetule's experiment: average and best fitness (average of 5 evolutionary runs). Source: [406]

Even if not conclusive, this investigation has highlighted how it could be feasible

to adapt the basic model described in this paper to any kind of environment. Although, it is not to be taken for granted that the original fitness formula might fit well to differently shaped and sized scenarios. The modifications made on this case have been marginal, but more studies are required in order to identify a general rule to follow when applying our model to different simulated environments.

5.4.3 Moving target (simulations C)

In this new experimental setup [319], labelled with the letter C, the target is able to detect and to move away from the approaching MAVs. This new property has been introduced to increase the complexity of the task and to test the robustness of the evolved controllers in front of a dynamically reacting environment.

Every simulation starts as usual, with the target deployed in a random position within the environment (either inside or outside the "enclosed area") and stationary. Four MAVs are deployed with starting positions and headings calculated as in Table 5.8. Their goal still consists of avoiding obstacles while reaching the target.

The main difference is in that, at each time step, if a MAV happens to be closer than $17px$ to the target, the latter switches (with probability 0.5) to "MAV detected mode". When the target is in this special "MAV detected move" it will start moving attempting to run away from the approaching aircraft. The rules of movement for the target are straightforward. When in "MAV detected mode" it will check, at any time step and before the MAVs move, which of the still operative aircraft is the closest. Once it has identified the closest "menace", the target has then to decide in which direction it should move. As it does not have a specific orientation, the target can move in whatever direction it likes. The evaluation process is carried out comparing among eight alternative locations around its centre of mass. These positions are respectively located at its North, North/East, East, South/East, South, South/West, West, and North/West (implementing what is called a Moore neighbourhood [390]). The distance of these locations from the target (equal for all of them) depends on the moving speed of the latter. The final decision is made comparing the Cartesian distance between the closest MAV and each of the potential

destinations. The target then selects and moves into the cell which maximises its distance from that aircraft.

An example of escaping movement adopted by the target can be seen in Figure 5.22.

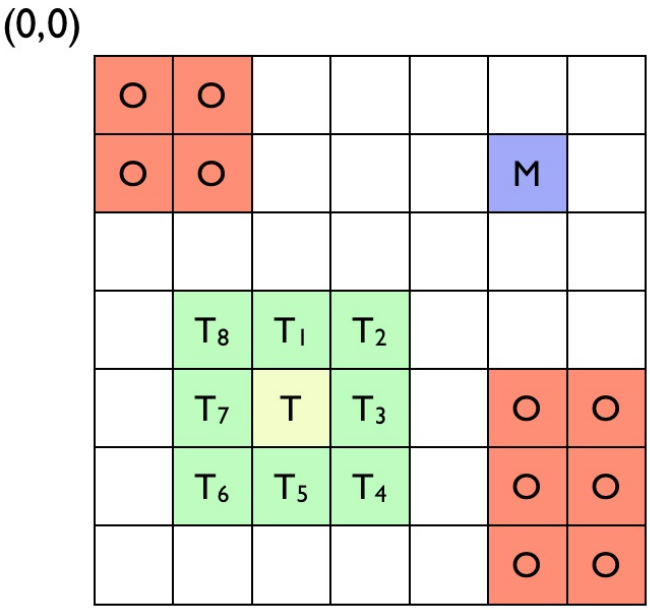


Figure 5.22: The options available to the target for escaping the approaching MAV. In this example the target T is in cell $(4, 2)$ and the MAV in $(1, 5)$. The target, assumed to be in "MAV detected mode," has to move in one cell among $T_1, T_2, ..., T_8$. In this case, the choice will be for cell T_6 $(6, 1)$ as it is the one that maximises its distance (to $1.41px$) from the approaching MAV [M: MAV, T: target, O: obstacle]

The target will keep escaping from the aircraft as long as all of the detected MAVs are operative and the distance between the target and the closest aircraft does not reach/exceed the $48px$ threshold.

Five simulations have been carried out where we vary the escaping speed of the target. These different speeds in the various simulations respectively correspond to different fractions of the MAVs speed (M_s): $M_s/2$ (simulation C1,) $M_s/3$ (simulation C2,) $M_s/4$ (simulation C3,) $M_s/5$ (simulation C4,) and $M_s/6$ (simulation C5).

The topology used for the neural controller is the one developed for simulation B4, which also implies that the MAVs use three ultra-sonic sensors respectively oriented at $-20°$, $0°$, and $20°$ compared to the aircraft heading.

The fitness function employed for evaluating the performance of the teams is the same Equation 5.3 used for simulations B. The only modification made in the

evolutionary algorithm has been an extension from 2,000 to 2,500 of the number of generations elapsed, as preliminary experiments demonstrated how reaching a stationary state takes longer in this scenario than in the previous ones.

Results

The results obtained are summarised in Table 5.11. Looking at the data collected we can easily identify a threshold of sorts. Simulations C3, C4, and C5 seem to perform equally well according to the various parameters measured. Simulation C2 produces a significantly worse performance in terms of average fitness, but can be considered to be performing reasonably well if we take into account both the maximum fitness and the success rate, as it produces results comparable with those obtained with a slowly moving target. In simulation C1, where the target moves at half the speed of the aircraft, the success rate of the MAVs drops instead, as does the average fitness, while the maximum fitness is comparable with the values obtained by the other simulations.

Table 5.11: Simulations C: resume of the main results (average of the last 10 generations, based on 5 evolutionary runs)

Sim.	Av. fitness	Max fitness	Percentage of tests concluded successfully	Max. succ. rate [%]
C1	138.93	395.18	54.48	94.1
C2	198.08	409.67	78.28	99.17
C3	250.68	411.74	81.89	100
C4	258.72	409.9	83.3	100
C5	242.42	413.05	84.06	100

The graph about the average fitness (Figure 5.23) highlights these differences in terms of performance between the different architectures, thus reinforcing the findings identified so far.

The maximum fitness plot (Figure 5.24) reaches a maximum and homogeneous level (about 400 fitness points) in about 1,000 generations for Simulations C2, C3, C4, and C5. Simulation C1 follows instead a slower growing trend and ends up reaching a steady state in about 2,000 generations, although stabilising on a value slightly lower than the one scored by other setups.

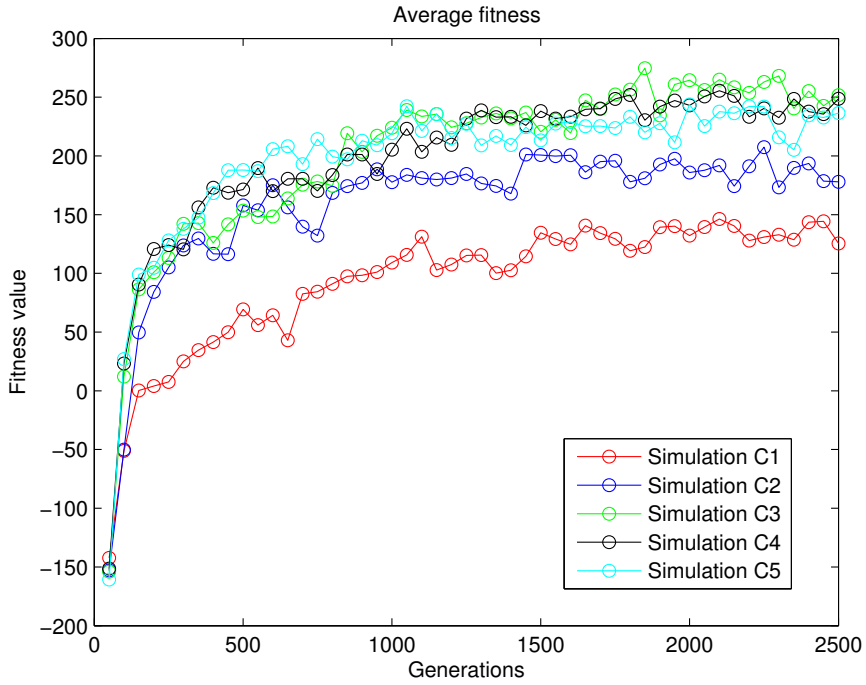


Figure 5.23: Simulations C: comparison for the average fitness (average of 5 evolutionary runs)

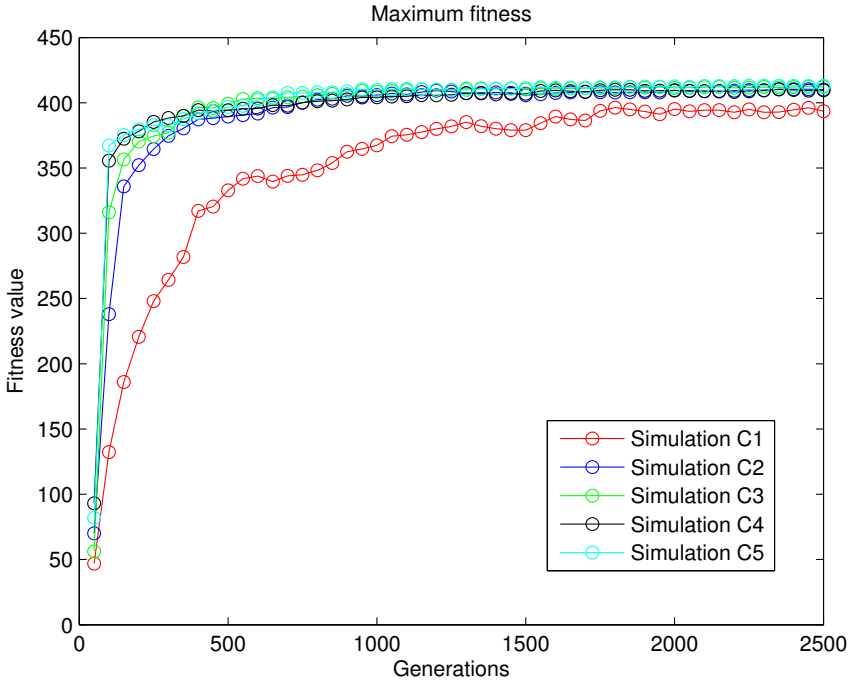


Figure 5.24: Simulations C: comparison for the best fitness (average of 5 evolutionary runs)

Figure 5.25 focuses on the success rate of the various setups. This graph, compared to the one in Figure 5.23, better highlights the homogeneity in the results generated by Simulations C3, C4, and C5 (slightly over 80%), and the difference with C2 (about 78%), and in particular with C1 (which just scores about 55%).

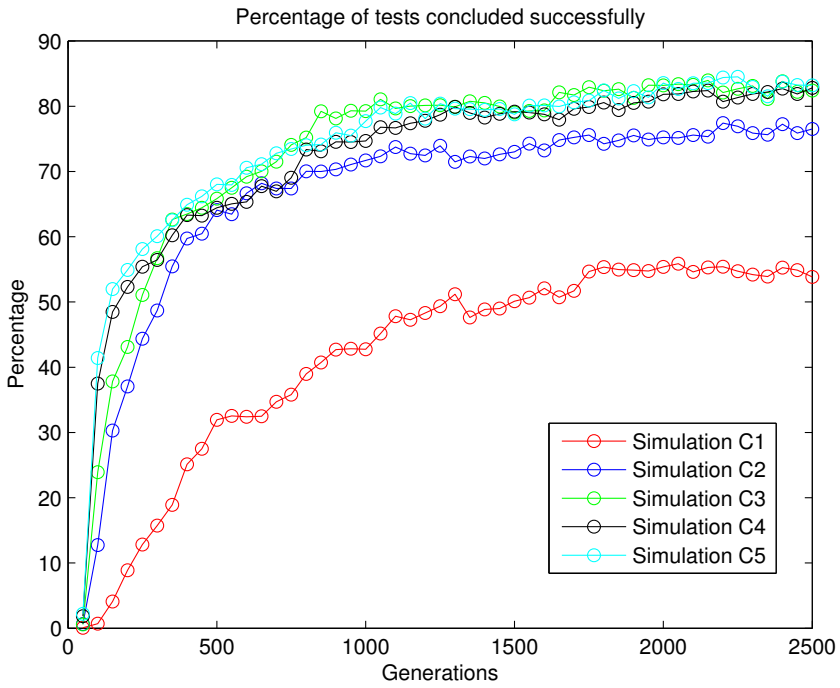


Figure 5.25: Simulations C: comparison for the success rate (average of 5 evolutionary runs)

We can now compare the results obtained by the new set of simulations with those generated in the B scenario (obstacles and fixed target). The comparison is performed by taking into account the best subset of C simulations (C3, C4, and C5) and contrasting them with Simulation B4, as the best one came from the previous experimental setup. The reason for focusing on C3, C4, and C5 - ignoring at the same time C2 and C1 - is due to some practical considerations, specifically, in relying on the hypothesis that the target to be tracked would be a person. A typical MAV platform could easily reach and maintain a cruise speed of about $60km/h$. One quarter of this velocity, assuming that the lower bound of this range is being used, roughly corresponds to $15km/h$ (with one fifth being approximately equal to $12km/h$, and one sixth to $10km/h$). Considering that the speed of an average person moving within a crowded environment could be approximated into the $[4km/h; 7km/h]$ range while walking, and $[12km/h; 15km/h]$ while running (ignor-

ing how this speed would be maintainable just for a short period of time), we might argue that the evolved controllers are able to accomplish the tasks that they are subject to with a high degree of confidence even in presence of a target with a moving speed, in a urban environment, comparable to the one of a human being.

On the basis of the considerations above, we can now look at the comparative plots. It should be noted that the comparison is limited to the first 2,000 generations, as no more were elaborated for Simulations B.

Figure 5.26 contrasts the average and the best fitness values in B and C. The first thing that comes to the eyes is the similarity in the results shown. The average fitness, as expected, is slightly lower for Simulations C as the task, involving a moving target, is significantly more difficult than the previous one. Although, this difference is not particularly relevant: 269.5 fitness points for B, 245.63 for C. Such a difference completely disappears if we look at the maximum fitness (*i.e.* the fitness value for the best individuals/controllers within a certain generation). In this case, the two curves tend to reach extremely similar levels of performance, thus indicating that the best controllers evolved in Scenario C perform, when tackling a target moving at no more than one quarter of MAVs' cruise speed, with the same proficiency as those evolved to deal with a static target.

Figure 5.27 compares the success rate obtained in the two scenarios. Here the difference between B and C can be more easily seen than when looking at the average fitness values plotted in Figure 5.26. The percentage of tests concluded successfully is 87.62% in B, 82.80% for Simulations C.

The main conclusion drawn from this experiment is that our simulation setup can evolve MAV controllers that are able to navigate through unknown environments and autonomously reach a target, not only when the latter is stationary on a certain position of the environment, but also when it is able to move away from the approaching aircraft. The only constraint is that, in order to keep a "good" average success rate, the target should not be able to move faster than one quarter of MAVs speed. In terms of potential application of these controllers in real life scenarios, the results suggest that such controllers could be employed for example

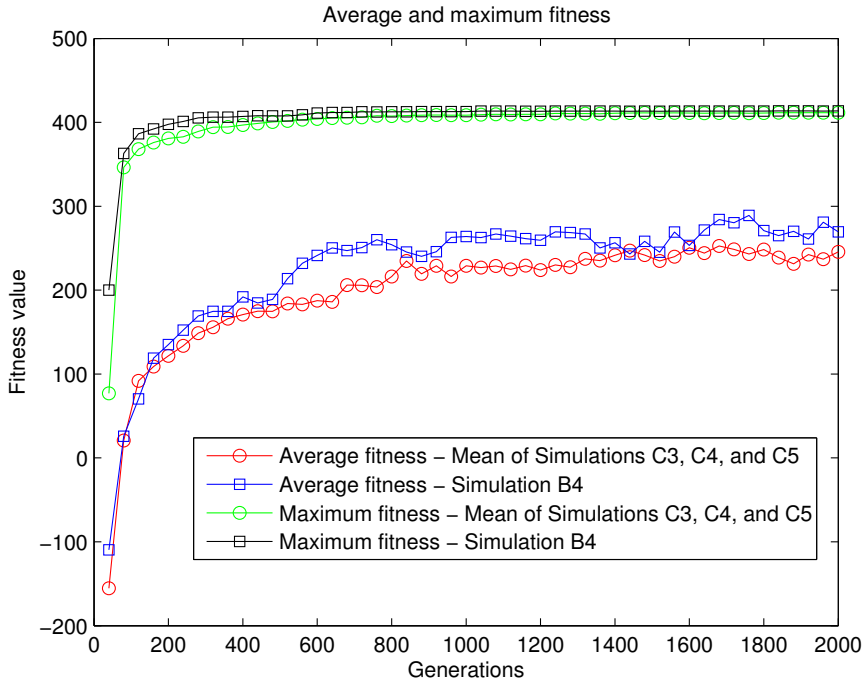


Figure 5.26: Simulations B and C: comparison for the average and best fitness (average of 5 evolutionary runs)

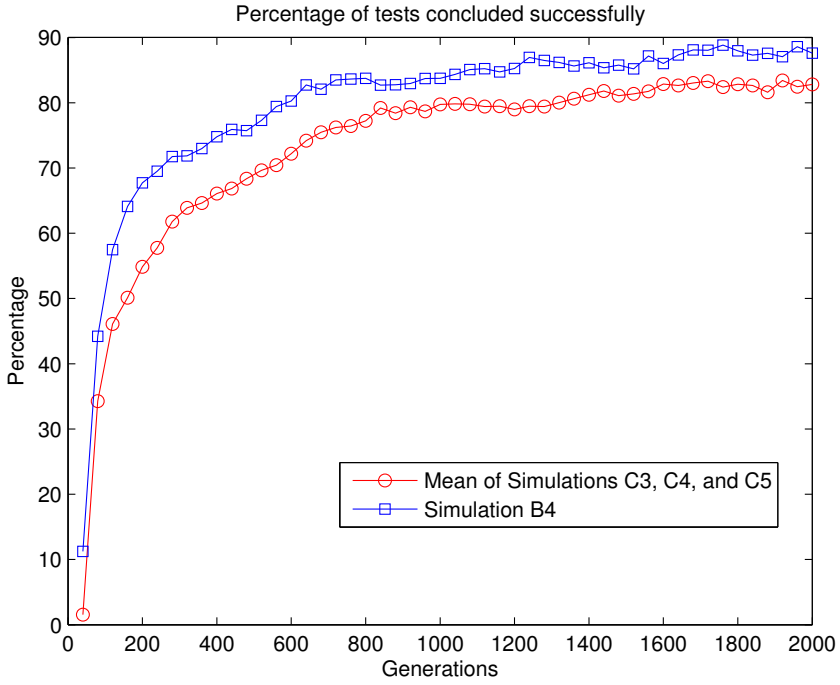


Figure 5.27: Simulations B and C: comparison for the success rate (average of 5 evolutionary runs)

using people, walking across an urban environment, as moving targets (ignoring, at the moment, all the implications and side effects related to the potential usage of hazardous payloads within crowded environments). But, again, the best evolved controllers (see the last column of Table 5.11) have demonstrated the ability to cope successfully with any target's speed. Even, quite unexpectedly, with targets moving at half the speed of the MAVs and hardwired for the maximum efficiency possible.

5.4.4 Implicit cooperation (simulations D)

The setup labelled with the letter D, described by Ruini *et al.* [326, 321], adds the constraint of requiring two MAVs to activate their end-operation units in proximity of the target at the same time (*i.e.* within a limited maximum number of time steps apart from each other, since the simulation works in discrete time) in order to succeed in the test.

The target begins each training epoch with the assigned status of "healthy". When one of the MAVs manages to reach the position in which the target is and is able to activate its end-operation unit within a $2.48px$ distance (*i.e.* the same situation that in the previous setups would have concluded the test as successful), the status of the target switches to "damaged". If a second MAV manages to activate its end-operation neuron when close enough to the target while the latter is still in "damaged" mode, the test would be considered a success. In the case of no MAVs managing to complete the task within 10 time steps since the target switched to "damaged", the target will restore to its original "healthy" condition and the simulation will goes on as usual, till the accomplishment of the task or the failure of the entire team.

In order to ensure that the aircraft can satisfy the new goal, we have provided them with the capability of gathering new pieces of information from the environment. Each member of the team is now able to detect both the status of the target (healthy rather than damaged) and the presence of a teammate within a 30px distance. This information is fed as input to the neural controller through two additional Boolean neurons. Apart from inserting these additional neurons, the

neural network maintains the same characteristics as before. Figure 5.28 provides a graphical representation of the new controller topology.

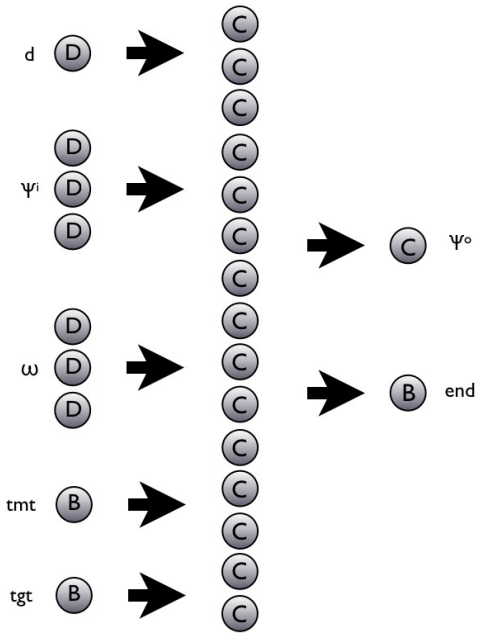


Figure 5.28: Graphical representation of the NN controller used in Simulations D. Its topology consists in 9 input neurons (1 for encoding the MAV-target distance, 3 for the MAV-target angle, 3 for the ultra-sonic perception, 1 for the detection of a teammate, 1 for the status of the target) 15 units processing the information in the hidden layer, and two output neurons (yaw and "end-operation" respectively) [D: discrete; C: continuous; B: Boolean]

We assume that the information about the target status is provided to the operative MAVs by the overall satellite-based system hypothesised before. The system receives the signal sent by an aircraft which has activated its end-operation unit at a distance compatible with the $2.48px$ distance threshold and interprets this communication as the happened "damaging" of the target. As in any real life scenarios where a task has to be performed cooperatively, the agents involved in it need to be provided with the ability to communicate, whether explicitly (*i.e.* intentionally), or implicitly (*i.e.* non-intentionally), amongst themselves. In this experimental setup we introduce a simple form of implicit communication merely consisting of the capability to detect the presence of teammates within a delimited perception field. In more details we define the experience of an implicit communication exchange as the ability to detect the presence of a teammate within the own sensorial space. This can be considered communication as information is provided by observation, it is implicit because the actor observed does not necessarily intend to communicate any

information through their behaviour. A work based upon a very similar principle can be seen in Pagello *et al.* [285].

The two new neurons implement a sort of logic OR. Apart from being in the proximity of the target, in order to decide the proper moment in which to activate its end-operation unit a MAV needs to know either that there is a teammate close to it (within a $30px$ distance, thus making it possible for the other MAV to reach the target within 10 time steps), or that the target is currently in damaged state, or that both conditions (proximity of a teammate and target's damaged state) are true.

The fitness function has been significantly modified also in order to allow the new desired behaviour to evolve. We have introduced two new concepts of "target approached" and "target damaged". At the end of a test, we define the target as approached if at least one MAV has activated its end-operation unit within a $63px$ distance from it. The target is considered damaged instead if at least one MAV has managed to do the same within the $2.48px$ threshold. These modifications tend to recreate what we could see as a sort of incremental evolutionary process (although not "formalised" as proper incremental evolution, a thing which has been done successfully by Barlow *et al.* [27] instead). The MAVs learn at first how to perform the simplest of the sub-tasks (*i.e.* avoid obstacles and approach, although quite roughly, the target), then they progressively move toward the following sub-task of increasing difficulty (getting closer and closer to the target), which in turn makes the accomplishment of the overall task possible.

Putting all this together, the new fitness function is expressed in Equation 5.5:

$$f = \gamma\frac{\chi}{4} + \eta\frac{\chi}{2} + \lambda\chi + 10\epsilon + \frac{\beta}{50} \tag{5.5}$$

where: γ is the number of tests concluded with at least one MAV approaching the target in the sense we have defined above; η is the number of tests concluded with at least one MAV damaging the target; λ is the number of tests concluded successfully and χ, a parameter arbitrarily chosen in order to assign different specific weights to γ, η and λ, equals to 50. Parameters ϵ and β have a similar meaning to those in

Equations 5.2 and 5.3, as they respectively represent the total number of MAVs still operative at the end of the all tests, and the average amount of energy retained by the MAV that had eventually concluded successfully the test.

For what concerns the evolutionary algorithm, it has to be considered that the new fitness function introduces several new parameters, thus drawing a fitness landscape that might easily be seen as more complex than what the previous fitness functions were creating. With this in mind the evolutionary algorithm has to be modified to cope effectively with the new scenario. Every team is now tested twelve times rather than four, the evolutionary process lasts for 5,000 generations (twice as much as in C), and ten evolutionary runs rather than five are evaluated.

Two different experimental setups have been elaborated for this scenario. One in which the target is a static one (Simulation D1), and one in which the target is able to move away from the approaching MAVs (Simulation D2).

Results

Before looking at the numerical results, we will discuss a qualitative description of the behaviour that has emerged. The strategy the MAVs evolve is straightforward, but nonetheless very effective. They independently look for their way to the target as in the previous experimental setup, without any interactions (if not purely generated by chance) with the teammates. Once the first MAV gets close to the target, instead of immediately activating its end-operation unit it keeps flying in circle around the location of the target. Only when a teammate arrives in the proximity of the target as well (thus being detected by the aircraft already there), the first MAV activates the end-operation procedure. Quickly following, the second MAV, detecting the target as damaged, does the same without waiting for the other members of the team to arrive, thus successfully concluding the test. An example of the evolved behaviour can be observed on Figure 5.29.

Looking at the figure showing the flight paths followed by the aircraft we can also identify a tentative wall-following behaviour exhibited by some of the MAVs (MAV #4 in particular). Although no walls or alternative physical boundaries

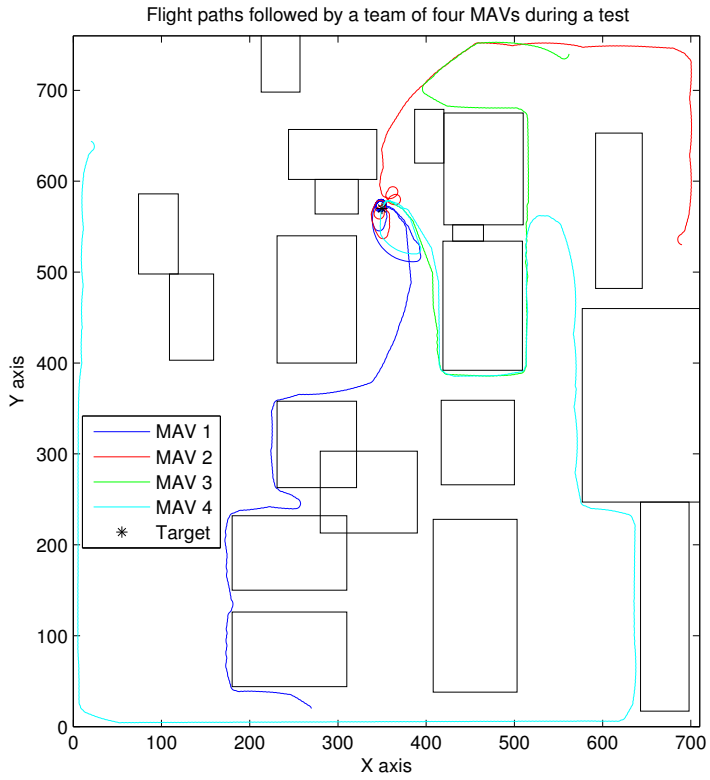


Figure 5.29: Simulation D1: flight paths followed by a team of MAVs cooperatively approaching the target

are surrounding the environment in which the simulation takes place, the ultrasonic sensors have been set up to consider some geographical coordinates (those corresponding to the edges of the simulation area) as obstacles. The controllers have therefore evolved to exploit this characteristic, specifically using obstacles and environment boundaries to make the task of navigating straight easier.

This is an approach that, as far as the author's knowledge goes, has not ever been implemented on autonomous controllers installed on physical MAVs, but which could nonetheless be worth investigating.

Now we can have a look at the data generated by the experiments. For what concerns the setup in which a non-moving target is employed (Simulation D1), Figure 5.30 presents the average and the best fitness values scored at each of the 5,000 generations elapsed. As this is the first experimental setup in which this fitness function is used, we can not perform any meaningful analysis of this graph other than study the trend, which develops smoothly as expected and reaches a steady state in about 4,000 generations.

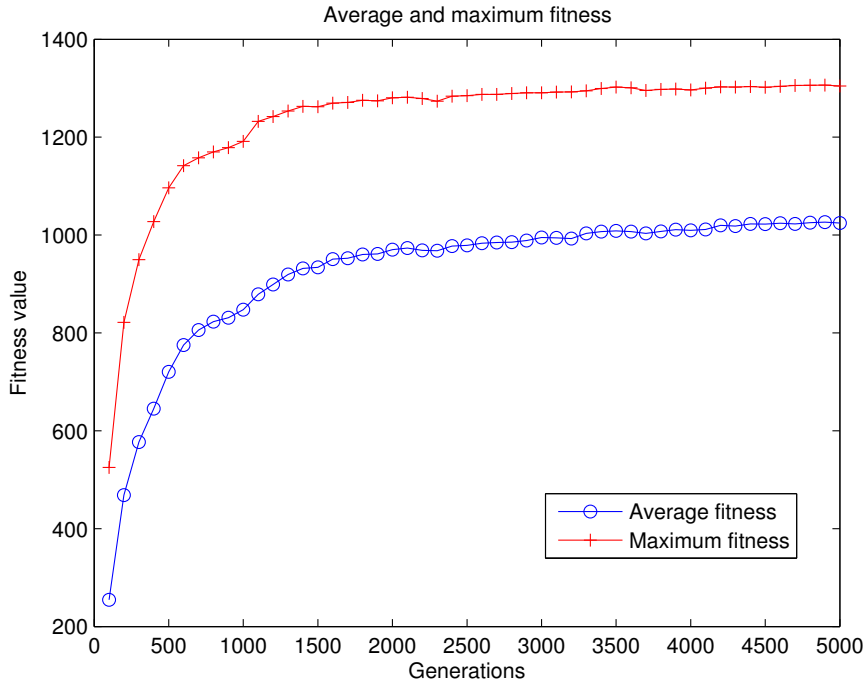


Figure 5.30: Simulation D1: average and best fitness (average of 10 evolutionary runs)

Figure 5.31 focuses on the success rate. Contrary to the graphs measuring the same data from the previous setups, in this one, three curves are plotted. The overall success rate is represented by the red curve with vertical lines as markers; the blue curve with circular markers shows the percentage of tests where at least one of the MAVs gets "damaged"; the green curve with diamond markers highlights the proportion of tests in which at least one MAV managed to "approach" the target (in the criteria used in the fitness function, *i.e.* to activate its end-operation unit when located closer than $63px$ to it). The simulations carried out using a fixed target have produced a surprisingly good performance. On average, for the individuals belonging to the last generation, more than 70% of tests (72.06%) are successful, while nearly 90% (88.08%) finish with the target correctly reached at least once. The percentage of targets "approached" quickly reaches its steady state, which exceeds the 90% threshold (92.16% as average of the last 100 generations).

The graph in Figure 5.32, concerning the condition of the evolved MAVs at the end of an average test, looks significantly different compared to B4 (the best among the setups involving a static target and obstacle avoidance capabilities). The first and most obvious difference is the amount of MAVs that have activated

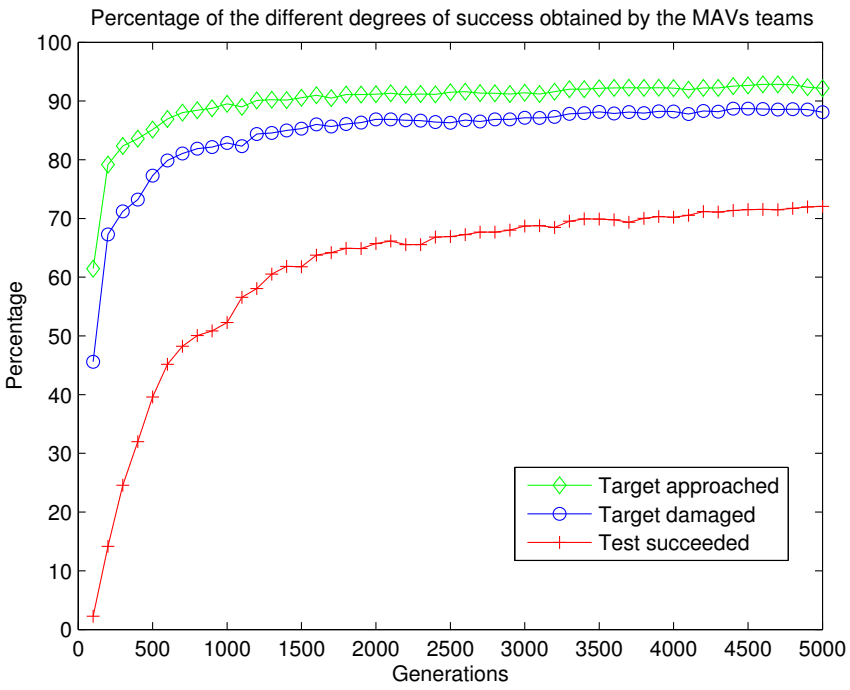


Figure 5.31: Simulation D1: percentage of tests concluded either successfully, with the approaching, or with the damaging of the target (average of 10 evolutionary runs)

their end-operation unit. In this scenario, the value increases from 1.03 to 2.2453. The difference from the optimum (which would be 1 in simulation B4, 2 in D1) is significantly larger in this experimental setup, as a side effect of the lower success rate achieved. This statistic impacts on the number of aircraft still operative at the end of the test, which drops from 2.3 to 0.99. But there are also other factors contributing to this result. The percentage of MAVs colliding with each other has increased from a negligible 0.05 to a more tangible 0.16 as an outcome of the evolved behaviour, which makes the MAVs aggregate in proximity of the target and fly in a circle around him. This is a behaviour that, other than increasing the likelihood of a collision, also increases the amount of aircraft running out of energy (from 0.23 to 0.33). On the other side, MAVs evolved in this scenario a much better obstacle avoidance capability probably due to the lengthening of the duration of the evolutionary process. Although the number of vehicles crashed against a building increases from 0.22 to 0.27, this effect is compensated by the amount of MAVs exiting from the environment boundaries during each test, which has dropped from 0.15 to 0.002.

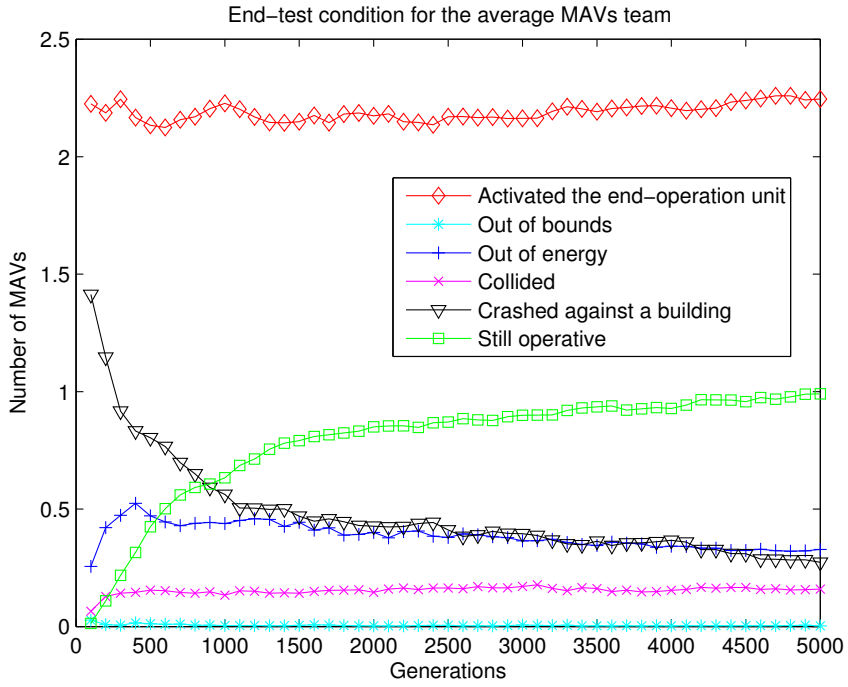


Figure 5.32: Simulation D1: condition of the MAVs at the end of the "average test" (average of 10 evolutionary runs)

The situation is more interesting for what concerns Simulation D2. Figure 5.33 shows the average and maximum fitness values registered in this scenario, both significantly lower than those obtained in D1: 804.7 v. 1024.6 for the average fitness, 1169.1 v. 1304.4 for the one of the best individual. Particularly significant is the relative drop in the curve relating to the maximum fitness, which is an unusual finding in our simulations as most of the times the maximum fitness remained similar across the different scenarios, while the trend for the average fitness curve was varying from scenario to scenario.

Figure 5.34 analyses in more detail the determinants of these significant differences in fitness. In the new setup, the fact that the target is able to move dramatically reduces the overall success rate of the MAVs, which drops below the 50% threshold (48.94%.) Also affected, although with a minor impact, are the statistics related to the percentage of tests concluded with the "damaging" of the target (78.81% v. 88.08%) and of those where the target has been at least approached by a MAV (86.03% v. 92.16%).

Figure 5.35, focusing on the end-test condition for the "average" MAV team (for which the rough data are provided in Table 5.12), provides the most useful findings.

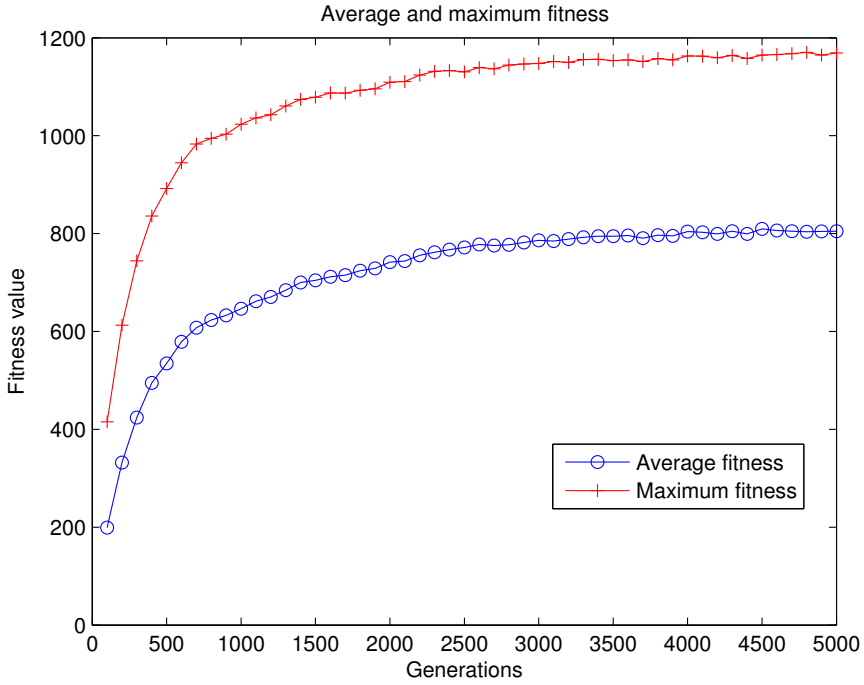


Figure 5.33: Simulation D2: average and best fitness (average of 10 evolutionary runs)

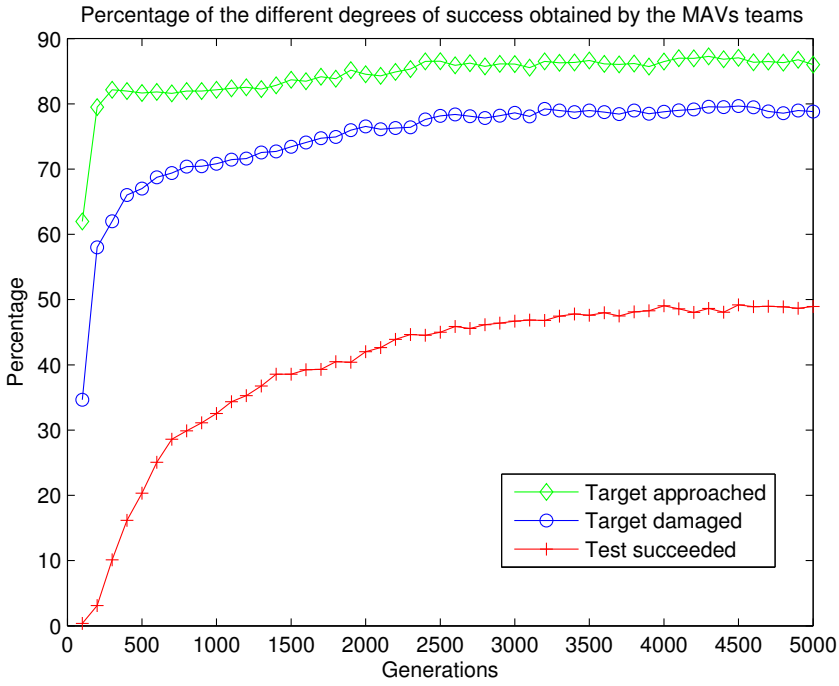


Figure 5.34: Simulation D2: percentage of tests concluded either successfully, with the approaching, or with the damaging of the target (average of 10 evolutionary runs)

The main difference is in the amount of MAVs still in operation at the end of the average test, falling to 0.59. This drop is not due to a bigger proportion of aircraft activating their end-operation unit. The data shows the opposite effect instead: 2.10 in D2 compared with the 2.24 obtained in D1. The low survival rate is due to an increased proportion of MAVs colliding against each other (0.29 v. 0.16), running out of energy (0.4 v. 0.33), exiting from the environment boundaries (0.03 v. 0.002,) and in particular crashing against a building (0.58 v. 0.27).

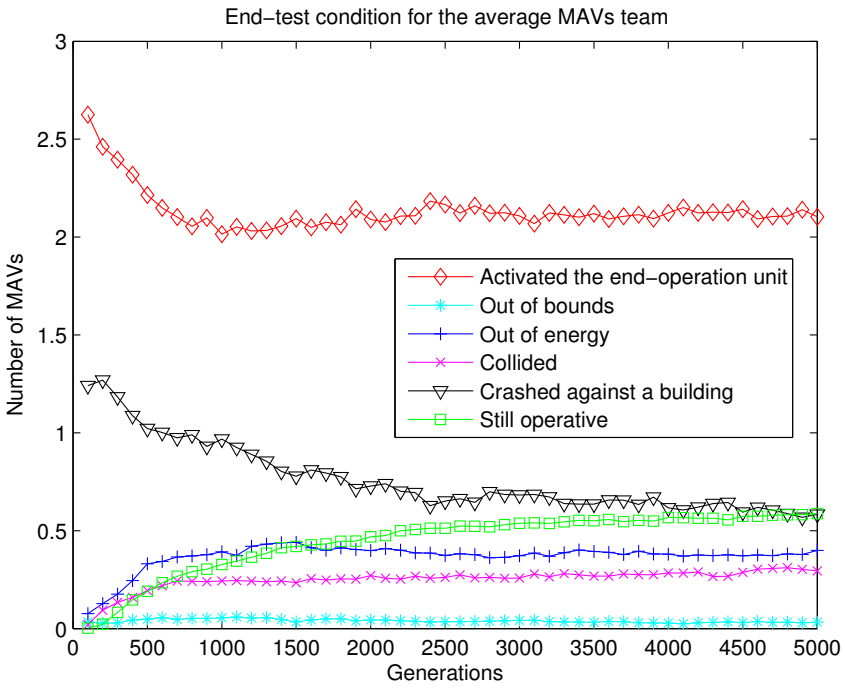


Figure 5.35: Simulation D2: condition of the MAVs at the end of the "average test" (average of 10 evolutionary runs)

Table 5.12: Simulations D: condition of the MAVs at the end of the "average test" (average of 10 evolutionary runs)

Sim.	Still operative	Activated their Boolean output	Crashed with a building	Out of bounds	Out of energy	Collided
D1	0.99	2.24	0.27	0.002	0.33	0.16
D2	0.59	2.10	0.58	0.03	0.4	0.29

A detailed comparison between the results obtained in the two setups can be found in Table 5.13.

Table 5.13: Simulations D: resume of the main results (average of the last 10 generations, based on 10 evolutionary runs) [SR: success rate]

Sim.	Av. fit.	Max fit.	SR [%]	Tgt dam. [%]	Tgt app. [%]	Max SR [%]
D1	1024.6	1304.4	72.06	88.08	92.16	97.51
D2	804.7	1169.1	48.94	78.81	86.03	89.92

Overall, a target able to move constitutes a major issue for the MAVs, as they find it more difficult to implement the strategy that emerged in D1, which is still the one they seem to imitate. The first aircraft approaching the target can have in fact a hard time flying around the target when it moves, for example, close to a building (when a test starts, the target is always deployed at a certain distance from both the buildings and the environment boundaries, but the problem arises when one of the MAVs approaches it). The continuous movements made by the target can easily cause the attacker to make mistakes, leading it to crash or, for example, in situations in which the first MAV misjudges the right moment in which to activate its end-operation unit causing the second aircraft to arrive too late on the target. Simulation D2 is one of the few (the only one if we do not take into account the preliminary ones used to determine the best way of encoding the input data) we have seen so far that can not reach the 90% threshold for the success rate of the best evolved controllers.

Workarounds on the GA (simulations E)

In order to improve the convergence speed of the evolutionary algorithm and to explore the solution space in a more efficient way, two new experiments have been carried out.

The first of these new setups (labeled E2) maintains the same characteristics as Simulation D1, though implementing three different genetic operators::

1. *selection operator*. As before, the best team in every generation is copied to the following one without any modifications (elitism). Then 94 pairs of parents are chosen for reproduction via a fitness-proportionate selection implemented as “roulette wheel” sampling [257];

2. *crossover operator*. This operator has been introduced in the form described by Montana and Davis [262]. Each of the selected pairs of parents generates a single offspring, thus creating 94 new individuals at every generation. Crossover works in the following way: for each non-input neuron of the offspring, one of the two parents is selected with 50% probability; the child inherits from the chosen parent the input connection weights to that neuron, as well as the associated bias;

3. *mutation operator*. The mutation operator affects all of the 94 offspring generated through crossover. For each neural network, three non-input neurons are randomly selected (all of them having the same probability of being chosen). The biases and all the incoming connection weights of the selected neurons are mutated through the addition of a value picked from a uniform distribution ranging between -0.5 and $+0.5$.

The remaining five individuals (not six as elitism is used) are created with randomly assigned connection weights and biases in order to preserve the algorithm from the risk of premature convergence.

The second setup (Simulation E2) is virtually identical to E1, with the only difference being the implementation of the selection operator. In E2, the fitness values used to calculate the size of the roulette wheel's slices are scaled (through the "sigma-scaling" method).

The results obtained in these two new experimental setups, detailed on the second and third row of Table 5.14, have highlighted a strong performance decreasing in comparison with D1, thus suggesting that it would be advisable to avoid the implementation of any modifications to the evolutionary algorithm used so far. At the same time, given this poor performance, the author has not found it useful to run an additional test employing the modified genetic algorithm on the D2 scenario.

Some explorative analyses have also been conducted using a binary genome, rather than one based on real values. Employing both Boolean and Gray Code encodings, with single and multi-points crossovers and different mutation rates, the results indicated yet again a significant difficulty for the network to reach a weight

Table 5.14: Comparison between simulations D1 and E

Sim.	Av. fitness	Max fitness	Percentage of tests concluded successfully	Max. succ. rate [%]
D1	1024.6	1304.4	72.06	97.51
E1	771.3	1133.7	47.47	88.29
E2	856.59	1253.9	56.69	97.03

set appropriate for the task, and therefore these conditions have been ignored too.

5.5 Conclusions

It is now time to summarise the results of the experiments described in this chapter. In paragraph 4.6.1 we have introduced our measure of success, which simply consists in the 90% accuracy for the best evolved controllers in performing the task required.

In the first experimental setup (simulations A) we aimed at identifying the best controller architecture for the simplest of the tasks studied, *i.e.* navigating autonomously towards a static target, within an obstacle-free environment, and performing a certain operation (represented by the activation of a specific neural network output) once there. Eight different neural network topologies (A1-A8) were contrasted and some of them produced very convincing results. The 90% success threshold, for example, was comfortably exceeded by two topologies, A2 and A6, even just looking at the average performance obtained by the controllers belonging to the last few generations (93.46% and 88.14% respectively). If we look at the performances generated by the best controllers of every configuration instead, five architectures (A1, A4, and A5, apart from the aforementioned A2 and A6) passed the test successfully, scoring values equal or close to 100%. The three remaining topologies (A3, A7, and A8) performed significantly worse, unable to reach even a 33% threshold. We can therefore conclude that most of the architectures tested resulted in being perfectly capable of implementing basic two-dimensional navigation, notwithstanding the constraints imposed on the motion of the MAVs. Looking in detail at the characteristics of the various architectures in relation to the results they scored, we can conclude that there is no a clear most effective way for encod-

ing the MAV-target distance, thus either using a discrete or a continuous encoding does not seem to affect the performance of the controllers too much (with a slight preference towards discrete encoding). The results related to the encoding of the MAV-target angle show instead a clear advantage provided by architectures relying on a segmentation of the angle in a certain number of subspaces, numbered by binary notations.

Positive results have been obtained in the second experimental setup (simulations B, obstacles introduced into the environment) as well, where the performances of 4 different controllers (varying the number and orientation of the ultra-sonic sensors and consequently the network topology in order to accommodate the different input) were compared. Only simulation B1 (in which a single forward-looking sensor was employed) failed in generating an average success rate for the evolved controllers exceeding the 90% threshold (stopping far below, precisely at 61.65%). Rather, simulations B2, B3, and B4 all obtained values well over the threshold, respectively 86.34%, 84.27%, and 87.62%. The best controllers performed with 100% accuracy or (simulation B1) very close to it.

In the third experimental setup (simulations C) the target can spot an approaching MAV and move accordingly trying to escape from it. We have contrasted five scenarios in which we varied the speed of the target. When the target's moving speed was lower than one fourth of MAVs' speed the average success rate exhibited by the controllers exceeded the 80% value (81.89% for C3, 83.3% for C4, and 84.06% for C5). The same results were not achieved with targets moving at a higher speed: 78.28% success rate for targets moving at one third of the MAVs' speed, 54.48% when escaping at half the speed of the hunter. Again we should look nonetheless to the best rates achieved by the evolved controllers, which highlights again the same one-fourth threshold: in C3, C4, and C5 the controllers have performed with a 100% accuracy, but also C1 and C2 exceed comfortably the 90% threshold (94.1% and 99.17% respectively) used to determine the "success" in our experiments.

For what concerns the fourth experimental setup (simulations D) the coordination required to the MAVs has made the task significantly more difficult to achieve

for their controllers. None of the two experiments carried out (simulation D1, in which the MAVs had to cooperatively approach a static target, and simulation D2, where the target attempts to escape) reached the 90% threshold in relation to the average success rate, falling short at 72.06% and 48.94% respectively. The success rate of the best-evolved controllers exceeded the "approval" threshold for simulation D1 (scoring 97.51%), but not for D2 (stopping at 89.92%).

In the set of simulations labelled with the letter E we have analysed some modifications to the GA component of our experiments. Although the results obtained might be considered satisfying looking at the performances generated by the best evolved controllers (at least for E2, which scored a good 97.03% accuracy, while E1 stopped at 88.29%) the average success rate dropped significantly, thus not giving us any good reason for studying them in more depth.

This chapter has also presented some analysis carried out in order to test the validity of the simulation model we have created in terms of generalisation. A few modifications to the fitness function used for the genetic algorithm has allowed to evolve controllers coping effectively with a different environment than the one used in simulations A-E.

Chapter 6

Simulation Experiments in 3D Environments

This chapter illustrates the second of the three computer models we have developed for the purposes of this Ph.D. research. What we present here is a three-dimensional extension of the 2D model presented in chapter 5. As with the previous model, the MAVs are engaged in autonomous navigation toward a certain target area. However this time the environment is obstacle-free. The vehicles rely upon a mixture of local and global information: the assumption underlying the model is still the one that consists in an "upper-level" system, aware of the location of the target, always available and able to broadcast this information in real-time to the aircraft. The MAVs can match this information with their own knowledge (*i.e.* proprioceptive information related to their current spatial position/orientation, and sensor readings) in order to find the path to be followed to reach the area in which the target is located.

6.1 Software simulator

The computer simulator used for the implementation of the 3D model, written in C++, has been developed relying on a few open-source libraries, namely Irrlicht[1] as 3D engine, NNFW to manage the neural networks-related aspects, and Qt because

[1] `http://www.irrlicht.sourceforge.net`

of the particular data structures these libraries offer, combined with the possibility of quickly implementing multi-threading support.

Notwithstanding the addition of a third dimension to the simulator, no physics engines[2] have been used, mainly for two reasons. First, this work explicitly looks at the navigation problem from an higher perspective than the one which is typically adopted in control systems literature. In order to focus on intelligent navigation, we simply assume that the robotic aircraft we simulate are able to respond to high-level commands (e.g., "*yaw* 1° *clockwise*," or "*pitch* 0.3° *up*") generated by a controller in the desired and expected way. This assumption is easily justified as we can simply think of an autopilot system (as those introduced in chapter 3) embedded on the aircraft, which takes care of the "low-level" issues (flight stabilisation and implementation of the navigation instructions received), combined with an onboard computer which runs the controller software, thus processing the information available and generating the commands to be executed. Once we implement in our simulator non-physics based, but at the same time non-unrealistic, aircraft dynamics our purposes are satisfied. Finally, the Evolutionary Robotics approach generally requires a significant amount of time for the evolutionary process to reach a steady state. Avoiding the use of a physics engine allows for a significant reduction of the computation time required to perform the evolution of the autonomous controllers.

The 3D simulator has been developed as consisting of two components running independently from each other: the evolutionary engine and the viewer. We will analyse them in the next two sub-paragraphs.

6.1.1 The evolutionary engine

The evolutionary engine is a command line tool, which performs the evolution of the controllers, reading the fundamental simulation settings from an external text file. The parameters the user can set by modifying this file are several:

[2]A physics engine is a computer software that provides an approximate simulation of certain physical systems, such as rigid body dynamics (including collision detection), soft body dynamics, and fluid dynamics, of use in the domains of computer graphics, video games and film. Examples of popular software physics engine often used in applications similar to the ones presented in this thesis are Open Dynamics Engine (ODE, `http://www.ode.org`), and Newton Game Dynamics (`http://newtondynamics.com`).

- *Size of the simulated environment* [X, Y, Z];

- *MAV parameters*: starting position (either distributed along the four corners of the environment or clustered together in a random position), movement length per time-step, starting amount of energy, energy consumption per time-step, range of the end-operation unit (how far from the target the MAV must be for its end-operation module to work properly), output multiplier (the absolute values generated by the yaw/pitch/roll neurons are multiplied by the value set here before being translated into commands and executed);

- *Target parameters*: these parameters define the behaviour of the target when it is allowed to move. In detail we have: first detection range (if the target has not detected a MAV yet, how close the aircraft must be to be noticed), first detection probability (what is the probability that a MAV closer than "first detection range" to the target will be detected by the latter), last detection range (how far a MAV must be, with respect to a target which has already spotted a MAV, in order to be detected), and target movement length (how far the target can move per time-step);

- *NN parameters*: network topology, number of units in the hidden layer;

- *GA parameters*: population size, number of MAVs per team, number of evolutionary runs, number of generations per run, number of testing epochs for each team, selection method (either rank selection or roulette wheel), number of controllers selected for reproduction at the end of each generation, number of offspring per parent, elitism [yes/no], probability of the mutation operator to modify a weight/bias of the parent being reproduced, minimum and maximum values for the range of the mutation operator, minimum and maximum of the range the connection weights and the biases can assume when the first generation is created;

- *Statistics*: directory where to save the results, number of data points to be displayed in the graphs, saving interval (how often the data should be saved on disk during an evolutionary process), RNG seed initialiser;

- *Incremental evolution*: enabled [yes/no], source directory containing the population to be further evolved;

- *Miscellaneous*: compile as a multi-thread application [yes/no], debug mode (continuously prints various outputs on the screen) [yes/no], timer enabled (used to measure the duration of the evolutionary process) [yes/no].

The decision to store all of these parameters into an external text file makes it significantly easier for the experimenter to automatically run several evolutionary processes through proper console scripts, and faster to adjust one or more parameters with debugging or performance comparison purposes. At the end of every few generations (depending on the "*saving interval*" parameter introduced above) the simulator saves a series of statistics into distinct text files on the disk:

- *Alive*: average number of MAVs per team alive at the end of a test;

- *Completion attempts*: average number of MAVs activating their end-operation unit during a test;

- *Energy remained*: average amount of energy left to the MAVs when a test ends;

- *Fitness (average)*: average fitness value for the entire population;

- *Fitness (maximum)*: best fitness value across the entire population;

- *Out of bounds*: average number of MAVs that attempted to exit the boundaries of the environment during a test;

- *Out of energy*: average number of MAVs running out of energy during a test;

- *Success rate (overall)*: overall percentage of tests concluded successfully for all the members of a certain generation;

- *Success rate (maximum)*: percentage of tests concluded successfully by the best team in a given generation;

- *Target distance (average)*: average distance between the target and the MAV which was the closest when it activated its end-operation unit during a test (average value for the entire population);

- *Target distance (minimum)*: average distance between the target and the MAV which was the closest when it activated its end-operation unit during a test (minimum value for the entire population).

Compared to the statistics collected by the 2D simulator discussed in chapter 5 we can see that their number has been reduced. A couple of metrics are not collected anymore indeed: the number of MAVs collided against each other during a test (switching to a 3D environment has made the likelihood of such collisions so low that they can be safely ignored), and the number of MAVs crashed against buildings (as there are no buildings in the new scenario).

The simulator relies on the dedicated NNFW functions to save the evolved controllers to the disk in XML format, at the end of the evolutionary process, using the same file structure described in the previous chapter.

6.1.2 The viewer

The viewer, the second main component of our 3D software simulator, is an application with its own GUI (designed in Irrlicht) capable of loading from the memory an evolved controller, assigning it to a certain number of MAVs and graphically displaying their flight behaviours to the end user in order to study the reasons behind certain behavioural patterns that might have emerged.

A screenshot of the viewer application can be seen in Figure 6.1. The picture highlights the 3D model of the MAV[3], a sphere representing the target (a squared bounding box is also visible), a series of real-time stats on the left-hand side of the application window and a few controls on the right-hand side. These controls[4] allow

[3]Downloaded from: `http://md2.sitters-electronics.nl`

[4]Some of these controls, and those referring to evolution in particular, are disabled. They appear in the picture as, initially, there were no plans for dividing the 3D simulator into an evolutionary engine and a separated viewer. This need arose later on and the earlier monolithic simulator, stripped off his evolutionary components, then became the viewer. Some buttons and controls were not removed in case they could have become useful at some point later. Rather they have been simply disabled.

the user to set the number of MAVs to be introduced inside the environment and the topology of their NN controllers. The user can start/stop a test at any time. When a test is running he is offered the possibility to switch between the different cameras available (one on the cockpit of each aircraft, one fixed on top of the simulated environment and looking down, one freely movable by the user).

Figure 6.1: Screenshot of the 3D simulator

The viewer also saves on the disk[5] a text file containing the flight paths followed by the various MAVs during the test. Technically, the structure of this file is extremely simple as it consists of a series of rows with values displayed along six columns (written in CSV format), according to the description provided in Table 6.1.

Table 6.1: Structure of the text file in which the flight paths followed by the MAVs during a test are memorised

Test ID	MAV ID	Time-step	X coord.	Y coord.	Z coord.

[5]By default on the same directory where the application executable is launched from.

6.1.3 Computation issues

With regard to computational aspects, this new simulator is "much heavier" and CPU-intensive than the one previously developed. Because of this the simulations presented in this chapter have been run on a dedicated computer grid rather than on a standard desktop machine. Specifically, all of the experiments have been carried out on a computer grid managed by Sun Grid Engine[6], consisting of 4 Apple™Xserve machines (each of which with two quad-core 2.66GHz Intel™CPUs and 4GB of RAM) awarded to our research group through the Apple ARTS program (see Appendix C). In order to benefit from the multitude of calculation cores available and based on the findings by Gautier [131], according to which the adoption of multi-threading programming methodologies can dramatically improve the speed of the evolutionary process for our 2D model, the evolutionary engine of the 3D simulator has been written so to be smoothly compiled and run as a multi-thread application [322].

6.1.4 Moving from 2D to 3D

Moving from a 2D to a 3D simulator implies that the degrees of freedom (DoF) available to the simulated aircraft increase from one to three. In the 2D scenario the MAVs rely in fact on a single DoF, since they can just rotate clockwise or anti-clockwise. An object located inside a three-dimensional environment, instead, can rotate around three different axes. We have already seen how, within the aeronautics field [76] these rotations are commonly named as a) *yaw* (Ψ), the rotation around the top-down axis; b) *pitch* (θ) the rotation around the wing-to-wing axis; and c) *roll* (Φ) the rotation around the nose-to-tail axis. These are the same rotations of the aircraft that we can simulate within our computer model.

From a control perspective, the introduction of rolling is the most significant addition to the previous model. According to the current roll angle of the aircraft, in fact, yaw and pitch rotations can produce completely different results. Making it difficult, for the controller, to correctly ascertain the potential outcome of any given

[6] `http://gridengine.sunsource.net`

manoeuvre if the orientation around the Φ axis is not taken into account.

6.1.5 Common features of all simulation setups

The 3D simulator we have developed has many of its main characteristics in common with the 2D model described in previous chapter. The simulations still focus on distributed control and group behaviour: the aircraft, driven by autonomous neural network based controllers, have to look for the target and, when close enough to it, activate a specific "end-operation" unit of their neural controllers (which can only be used once per life-span).

The reference environment, sketched in Figure 6.2, is a three-dimensional area with size $1,000$ (X) by $1,500$ (Z) by 600 (Y) Graphical Units (GUs[7]). The aircraft have an approximate length of $3.5GU$, while the target is constituted by a sphere with a $15GU$ radius. $15GU$ is also the same radius used for the end-operation procedure, *i.e.* a MAV needs to activate its Boolean output unit when closer to the target than that in order for the current test to be considered successfully (as long as a single MAV reaching the target is considered a successful outcome of the task).

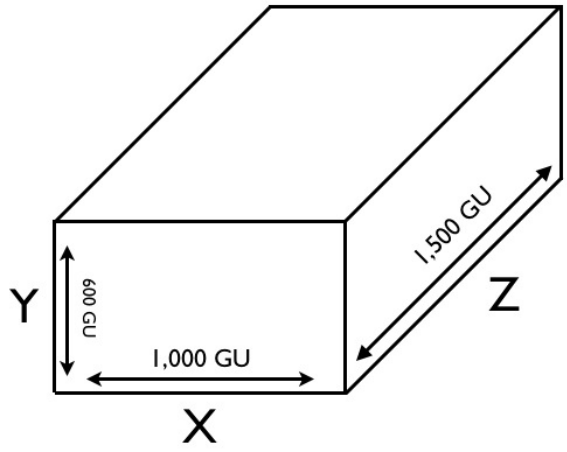


Figure 6.2: The simulation reference environment (the axes notation comes from Irrlicht, the 3D computer graphics engine adopted)

In the 3D model teams can be made up of a differing number of aircraft as opposed to the 2D model in which the team numbers were uniform. As we will see later, simulations A and B (where the task can be successfully performed by a single

[7]The use of a 3D computer graphics engine made our model using this unit of measure rather than pixels.

agent) will use individual MAVs for the evolutionary process, while teams of four MAVs will be used in simulations C because of the different requirements of the task.

Each controller is tested for a certain number of epochs being cloned in all of the MAVs members of the same team. Each of these tests starts with the aircraft deployed in different positions and with the target randomly assigned to approximately the middle of the environment. Table 6.2 resumes the rules followed by the simulator for the initial deployment of the agents in the different setups. When teams made of a single MAV are used, the aircraft is deployed alternatively in each starting position depending on the test epoch studied. When four MAVs are used instead, each of them will always start any test from a specifically designated position.

Table 6.2: 3D simulations: initial deployment of MAVs and target

Agent	X coordinate	Z coordinate	θ	Ψ	Φ
Target	$[200; 800]$	$[300; 1200]$	N/A	N/A	N/A
MAV_1	$[30; 100]$	$[1350; 1470]$	$-90°$	$[-55°; -35°]$	$0°$
MAV_2	$[900; 970]$	$[1100; 1220]$	$-90°$	$[35°; 55°]$	$0°$
MAV_3	$[30; 100]$	$[30; 150]$	$-90°$	$[-145°; -125°]$	$0°$
MAV_4	$[900; 970]$	$[30; 150]$	$-90°$	$[125°; 145°]$	$0°$

With regard to the starting orientations, they are measured according to the following conventions:

- θ: express the relation between the aircraft wing-to-wing axis and the environment Y axis. $0°$ is the aircraft deployed along that axis facing up, $-180°$ corresponds to the MAV still parallel to that axis but facing downwards;
- Ψ: assuming there is no Y axis, $45°$ identifies a MAV deployed in $X = 1000, Z = 1500$ facing the centre of the environment. The value increases clockwise within the $[-180°; 180°]$ range;
- Φ: $0°$ refers to the aircraft parallel to the ground. Rotating the right wing down would increase Φ, while rotating the right wing down would decrease it.

Table 6.2 does not explicitly mention the Y axis, as some of the experimental setups elaborated upon exclude the possibility for pitch rotations from the MAVs'

behavioural repository. In case of scenarios where pitch is allowed, each MAV is initially deployed at an altitude randomly picked within the $[200; 400]$ range, while the target is in $[15; 500]$. If pitch is not contemplated (thus recreating a pseudo-2D scenario inside a 3D world), the MAVs are all deployed at $10GU$ along the Y axis, and the target at $15GU$. The (X, Z) coordinates of the target refer to its centre, and this explains why Table 6.2 indicates different altitudes for the MAVs and for the target.

A MAV starts a test with a certain amount of Energy Units (EUs) available. During each time step it consumes $1EU$, while moving $2GU$ along its heading direction. The rotations generated by the controller in the time unit are included within the $[-3.0\,^\circ; 3.0\,^\circ]$ range. It is worth noting that, in order to simulate a more realistic flying behaviour, every time the aircraft performs a yaw manoeuvre a corresponding inverse amount of roll is automatically applied. For example, whenever the controller generates a $+0.7^\circ$ yaw, a corresponding rotation with magnitude -0.7° is automatically applied around the Φ axis. This effect has only been implemented in those experimental setups that include the possibility of independent rolling for the aircraft.

6.2 Neural network controllers

Several topologies of neural network based controllers have been tested, as it will be shown in more detail in Section 6.4.1.

The controllers used, for the most part, are made of fully connected feed-forward neural networks embodied into the MAVs. These controllers are fed with input information coming both from the external environment (location of the target) and from within the robot (own orientation around the three axes, which is in turn translated into relative orientation toward the target). This information is processed by the network, which uses it to determine the activation level of the output units, directly connected to the MAV's motor actuators.

Despite the way in which many of the experimental setups are described in the following pages, the aircraft will be somewhat limited in their behavioural capabil-

ities. This is demonstrated by the scenario with the greatest degree of freedom. In this case, the output layer is composed of four neurons. Three of these units (continuous) determine the rotations the MAV will perform in the time unit: yaw (Ψ^o), pitch (θ^o), and roll (Φ^o). The remaining neuron (*end*) is the aforementioned "end-operation" unit.

All of the non-input and non-Boolean neurons belonging to the network are activated according to a log-sigmoid function (slope 1.0, see Equation 2.14), in which output values are within $[-1.0; 1.0]$. Summation (see Equation 5.1) is the only aggregation function used.

As before, the Boolean output can be activated only once during the entire individuals' life span. When this neuron turns to 1 - as well as when the MAV exits from the environment boundaries, collides against a teammate, or runs out of energy - the aircraft switches to "non-operative" mode. When no MAVs are operative anymore, the current test epoch is immediately considered concluded. A test of this type could be considered either successful or unsuccessful. In the case of an activation of the end-operation unit by the last aircraft it would depend upon the MAV-target distance when this event has taken place and also upon the requirements of the experimental setup being analysed.

6.2.1 Encoding of the input information

With regard to the encoding of input information, we have to first identify the entire set of inputs that the MAVs can receive at any time step. This data consists in: $\Delta\Psi^i$, which is the MAV-target horizontal angle (*i.e.* the 2D angle calculated on the X and Z axes and compared with the heading value Ψ); $\Delta\theta^i$ which is the MAV-target vertical angle (*i.e.* the 2D angle calculated on the X and Y axes and compared with θ); Φ^i (a normalisation of the rough Φ value); d^i, which is the MAV-target distance.

All of these inputs can either assume continuous or discrete values according to the architecture under examination. Before looking at the encoding used for the different input information, we will highlight the ranges of $\Delta\Psi$, which is $[0°; 360°]$, and the one used by both $\Delta\theta$ and Φ, which corresponds to $[-180°; 180°]$.

In relation to the MAV-target horizontal angle, its discretisation follows similar rules to those employed in the 2D simulator, with the only difference consisting in a more accurate decomposition of the space surrounding the MAV, which is now done using 16 different subspaces. In order to map these subspaces, four neurons are required. Gray code encoding, as shown in Table 6.3, has been used.

Table 6.3: Discretised encoding of the MAV-target horizontal angle

Original angle ($\Delta\Psi$)	Discretised value ($\Delta\Psi^i$)
$(x \geq 348.75°)\|\|(x \leq 11.25°)$	0000
$11.25° < x \leq 33.75°$	0001
$33.75° < x \leq 56.25°$	0011
$56.25° < x \leq 78.75°$	0010
$78.75° < x \leq 101.25°$	0110
$101.25° < x \leq 123.75°$	0111
$123.75° < x \leq 146.25°$	0101
$146.25° < x \leq 168.75°$	0100
$168.75° < x \leq 191.25°$	1100
$191.25° < x \leq 213.75°$	1101
$213.75° < x \leq 236.25°$	1111
$236.25° < x \leq 258.75°$	1110
$258.75° < x \leq 281.25°$	1010
$281.25° < x \leq 303.75°$	1011
$303.75° < x \leq 326.25°$	1001
$326.25° < x \leq 348.75°$	1000

In the continuous form, the input for the neural network assumes instead a value determined by Equation 6.1.

$$\Delta\Psi^i = \begin{cases} \dfrac{\Delta\Psi}{180}, & \text{if}\Delta\Psi \leq 180° \\ 1 - \dfrac{\Delta\Psi - 180}{180}, & \text{if}\Delta\Psi > 180° \end{cases} \tag{6.1}$$

The vertical MAV-target angle is discretised following the same principle used for the encoding of the horizontal angle. The only differences, highlighted in Table 6.4, are due to the different range of values this parameter can assume which can in turn affect the discretisation map.

Equation 6.2 shows the simple formula used for the continuous normalisation instead.

Table 6.4: Discretised encoding of the MAV-target vertical angle

Original angle ($\Delta\theta$)	Discretised value ($\Delta\theta^i$)
$(x \geq -11.25°)\&\&(x \leq 11.25°)$	0000
$11.25° < x \leq 33.75°$	0001
$33.75° < x \leq 56.25°$	0011
$56.25° < x \leq 78.75°$	0010
$78.75° < x \leq 101.25°$	0110
$101.25° < x \leq 123.75°$	0111
$123.75° < x \leq 146.25°$	0101
$146.25° < x \leq 168.75°$	0100
$(x > 168.75°)\|\|(x \leq -168.75°)$	1100
$-168.75° < x \leq -146.25°$	1101
$-146.25° < x \leq -123.75°$	1111
$-123.75° < x \leq -101.25°$	1110
$-101.25° < x \leq -78.75°$	1010
$-78.75° < x \leq -56.25°$	1011
$-56.25° < x \leq -33.75°$	1001
$-33.75° < x \leq -11.25°$	1000

$$\Delta\theta^i = \frac{\Delta\theta}{180} \tag{6.2}$$

The current roll angle is represented through a single neuron, both when discretised and when its value is used in a continuous way. The discretisation process works in accordance with Table 6.5.

When the input is encoded in a continuous form, the normalisation is done according to Equation 6.3.

$$\Phi^i = \frac{\Phi}{180} \tag{6.3}$$

In the case of the MAV-target distance, this information is always encoded using a single neuron, whether its values are discrete or continuous. The discretisation process is made in accordance with Table 6.6.

When this input is kept continuous, the only reference used for the normalisation process is the maximum distance allowed by the environment. This distance corresponds to $1,900GU$. The conversion is then performed through the formula in Equation 6.4.

Table 6.5: Discretised encoding of the MAV bank angle

Original angle (Φ)	Discretised value (Φ^i)
$(x \geq -2°)\&\&(x \leq 2°)$	0
$2° < x \leq 9°$	0.05
$9° < x \leq 18°$	0.1
$18° < x \leq 27°$	0.15
$27° < x \leq 36°$	0.2
$36° < x \leq 45°$	0.25
$45° < x \leq 54°$	0.3
$54° < x \leq 63°$	0.35
$63° < x \leq 72°$	0.4
$72° < x \leq 81°$	0.45
$81° < x \leq 90°$	0.5
$90° < x \leq 99°$	0.55
$99° < x \leq 108°$	0.6
$108° < x \leq 117°$	0.65
$117° < x \leq 126°$	0.7
$126° < x \leq 135°$	0.75
$135° < x \leq 144°$	0.8
$144° < x \leq 153°$	0.85
$153° < x \leq 162°$	0.9
$162° < x \leq 171°$	0.95
$171° < x \leq 180°$	1
$(x \geq -9°)\&\&(x < -2°)$	-0.05
$-9° > x \geq -18°$	-0.1
$-18° > x \geq -27°$	-0.15
$-27° > x \geq -36°$	-0.2
$-36° > x \geq -45°$	-0.25
$-45° > x \geq -54°$	-0.3
$-54° > x \geq -63°$	-0.35
$-63° > x \geq -72°$	-0.4
$-72° > x \geq -81°$	-0.45
$-81° > x \geq -90°$	-0.5
$-90° > x \geq -99°$	-0.55
$-99° > x \geq -108°$	-0.6
$-108° > x \geq -117°$	-0.65
$-117° > x \geq -126°$	-0.7
$-126° > x \geq -135°$	-0.75
$-135° > x \geq -144°$	-0.8
$-144° > x \geq -153°$	-0.85
$-153° > x \geq -162°$	-0.9
$-162° > x \geq -171°$	-0.95
$-171° > x \geq -180°$	-1

Table 6.6: Discretised encoding of the MAV-target distance

Distance (d)	Discretised value (d_d)
$950 < d$	0
$633.3 < d \leq 950$	0.025
$475 < d \leq 633.3$	0.05
$380 < d \leq 475$	0.075
$316.67 < d \leq 380$	0.1
$271.43 < d \leq 316.67$	0.125
$237.5 < d \leq 271.43$	0.15
$211.11 < d \leq 237.5$	0.175
$190 < d \leq 211.11$	0.2
$172.72 < d \leq 190$	0.225
$158.33 < d \leq 172.72$	0.25
$146.15 < d \leq 158.33$	0.275
$135.71 < d \leq 146.15$	0.3
$126.67 < d \leq 135.71$	0.325
$118.75 < d \leq 126.67$	0.35
$111.76 < d \leq 118.75$	0.375
$105.56 < d \leq 111.76$	0.4
$100 < d \leq 105.56$	0.425
$95 < d \leq 100$	0.45
$90.48 < d \leq 95$	0.475
$86.36 < d \leq 90.48$	0.5
$82.61 < d \leq 86.36$	0.525
$79.17 < d \leq 82.61$	0.55
$76 < d \leq 79.17$	0.575
$73.08 < d \leq 76$	0.6
$70.37 < d \leq 73.08$	0.625
$67.86 < d \leq 70.37$	0.65
$65.52 < d \leq 67.86$	0.675
$63.33 < d \leq 65.52$	0.7
$61.29 < d \leq 63.33$	0.725
$59.37 < d \leq 61.29$	0.75
$57.57 < d \leq 59.37$	0.775
$55.88 < d \leq 57.57$	0.8
$54.28 < d \leq 55.88$	0.825
$52.78 < d \leq 54.28$	0.85
$51.35 < d \leq 52.78$	0.875
$30 < d \leq 51.35$	0.9
$15 < d \leq 30$	0.95
$0 \leq d \leq 15$	1.0

$$d_c = 1 - (d/1900); \tag{6.4}$$

6.3 Evolutionary algorithm

Following what the ER paradigm dictates, the proper sets of synaptic weights and biases for the neural network controllers to perform the desired tasks are obtained by running a GA.

The starting population used consists of 30 individuals[8], with connection weights and biases randomly assigned at the beginning of the evolution within the $[-10.0; 10.0]$ range. The genome is implemented via parametric encoding, with each gene constituted by a real value, representing either a connection weight or a bias.

When all the members of the current generation have been evaluated according to a proper fitness function, the five individuals that scored the best performances are selected for reproduction. The best individual is copied to the next generation without any modifications (elitism), and replicated in four offspring. The other four selected individuals all generate five offspring each, originally identical to the parents. All of the offspring (except for the one produced by the elitism operator) are subject to a process of random mutation which affects each of their genes - with probability 0.1 - by a uniformly distributed random value picked within the $[-0.05; 0.05]$ range.

Five new individuals, with a random set of connection weights and biases in the $[-10.0; 10.0]$ range, are introduced into any new generation to reduce the risk of premature convergence within the population. The process is reiterated for a certain amount of generations and then repeated from the scratch for a few times (we will call each of this run "evolutionary seed" or "evolutionary run") in order to obtain results that are the least affected by randomness as possible.

[8]The relatively small number chosen for the population size is inspired by the tradition of the "Sussex approach" [154] and preferred in this case over our previous approach because of the increased complexity of the software simulator, which means, in turn, a longer time required for the evolution.

6.4 Experiments

Using the model described above, three different experimental setups have been elaborated. The details of these experiments and the results obtained are described in the following subsections. As for the model described in previous chapter, the experimental setups designed for the 3D simulator are of increasing complexity. We have mentioned in the introduction of the current chapter that we will not be using obstacles with this simulator. The reason behind this is that to evolve an autonomous controller for navigating a 3D environment has proven to be a challenging issue in itself. We have therefore decided not to introduce, at least at this stage, further elements making the experimental setups more complicated. In the first scenario a static target is deployed inside the environment and the MAVs have to navigate to it and, once there, to activate their "end-operation" unit as usual. In the second scenario the target moves away from the approaching MAVs. Finally, the third scenario requires cooperation amongst the MAVs, as at lest two of them are required to approach the target (either stating or capable of moving) at the same time

6.4.1 Basic scenario (simulations A)

The first scenario (simulations A, described in Ruini *et al.* [321, 323, 325]) acts as an experimental benchmark. Other than evaluating whether the proposed approach could work in a 3D scenario or not, what we are interested in investigating in this first set of simulations are the performances generated by different neural network topologies relying on contrasting input sets and internal organisations.

The scenario is quite basic, as it only consists of a single MAV navigating through an obstacle-free environment looking for its way to the target. The MAV has $5,000EU$ to start with, thus being able to fly for a total of $10,000GU$. The evolutionary process is iterated for $20,000$ generations[9], with every controller being tested four times. Ten evolutionary seeds are evaluated and their results averaged

[9]As for the 2D simulator, also in this new model the length of the evolutionary process has been determined by empirical analysis carried out to evaluate how long the GA would require to reach a steady state.

together.

Inspired by the analysis carried out by Zetule and presented in the previous chapter, the fitness function used for measuring the performance of the controllers in this setup has been modified and now just involves two parameters: α, which represents the average value - across the four epochs of testing - for the difference between the MAV-target distance at the beginning and at the end of a test (thus representing the distance covered, as done in the study by Zetule); β, indicating instead the overall number of tests succeeded. α is set to 0 in case the MAV has concluded the test because it has exited from the environment boundaries (including crashing on the ground) or ran out of energy. Equation 6.5 shows the simple way in which these two parameters have been incorporated into the fitness function.

$$fitness = \alpha + \beta * 100 \tag{6.5}$$

Differently to the fitness functions used in conjunction with the 2D model, Equation 6.5 can rarely assume negative values. This only happens when the MAVs somehow manage to end a test farther to the target then they were at the beginning. As soon as the aircraft start to approximate the completion of the task, the value of α acts as a discriminant between competing controllers. β provides an additional boost for the fitness evaluation of the controllers capable of completing the task successfully.

Various neural network topologies have been tested. The variables that have been used in the different architectures are:

- *rotation axes*: having the possibility to perform all the three rotations (yaw, pitch, and roll) described or just a subset of these impacts the input/output units present on the controller;
- *short term memory*: present/absent;
- *hidden/internal layer*: present/absent;
- *input encoding*: a discrete/continuous input information stream gathered from the environment.

A summary of the 24 architectures analysed is reported in Table 6.7. When the hidden layer is used, the number of neurons in it has been arbitrarily set to ten. The presence or the absence of the hidden layer will also affect the way in which short term memory structures are implemented. Elman networks [101] are used when an internal layer of units is present and a Jordan [182] network when the topology under examination does not include such a component.

Table 6.7: Neural network controller architectures tested

Architecture	Pitch	Roll	Hidden layer	Memory	Input
1	No	No	No	No	D
2	No	No	No	No	C
3	Yes	No	No	No	D
4	Yes	No	No	No	C
5	Yes	Yes	No	No	D
6	Yes	Yes	No	No	C
7	No	No	Yes	No	D
8	No	No	Yes	No	C
9	Yes	No	Yes	No	D
10	Yes	No	Yes	No	C
11	Yes	Yes	Yes	No	D
12	Yes	Yes	Yes	No	C
13	No	No	No	Jordan	D
14	No	No	No	Jordan	C
15	Yes	No	No	Jordan	D
16	Yes	No	No	Jordan	C
17	Yes	Yes	No	Jordan	D
18	Yes	Yes	No	Jordan	C
19	Yes	No	Yes	Elman	D
20	Yes	No	Yes	Elman	C
21	Yes	No	Yes	Elman	D
22	Yes	No	Yes	Elman	C
23	Yes	Yes	Yes	Elman	D
24	Yes	Yes	Yes	Elman	C

Figures 6.3 illustrates from a graphical point of view the topologies of the most representative controllers tested. For every architecture a corresponding simulation has ben ran (A1 is the simulation experiment in which the controllers use topology 1, in A2 they rely on the architecture 2, etc.).

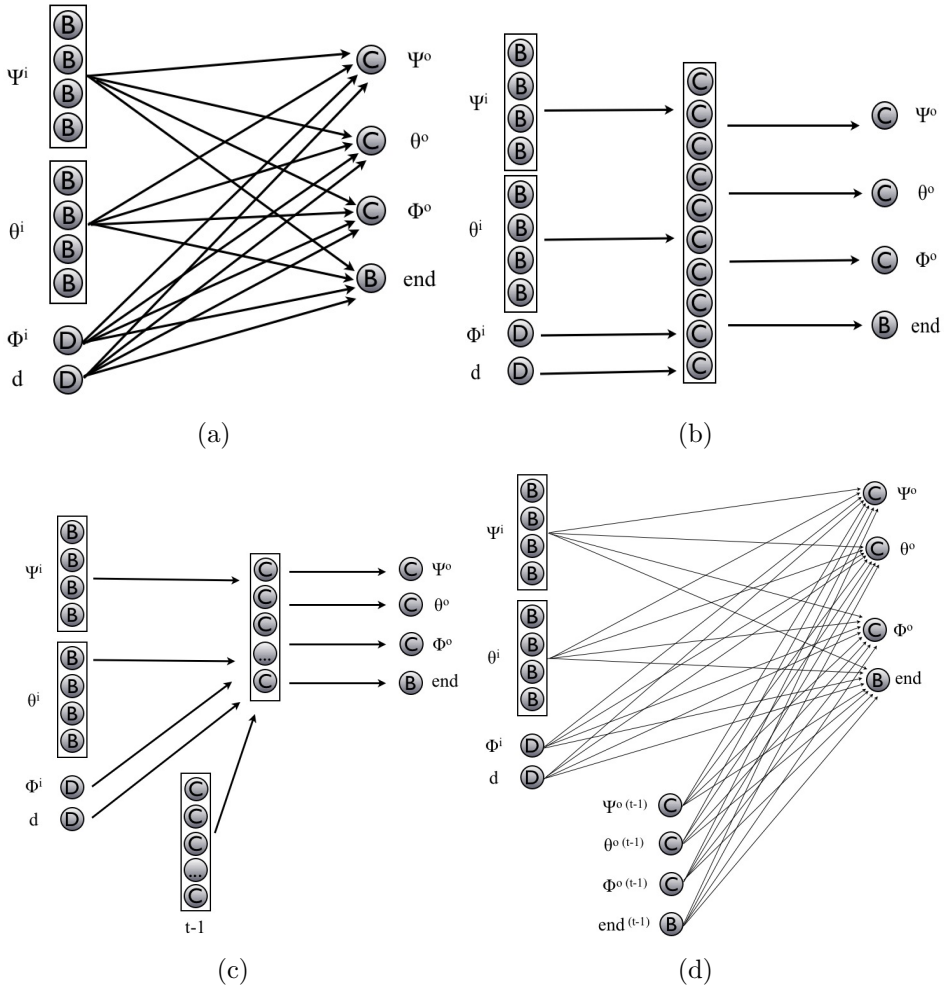


Figure 6.3: Some of the most representative NN topologies tested: (a) full I/O, no hidden layer, no memory; (b) full I/O, hidden layer, no memory; (c) full I/O, no hidden layer, memory structure (Jordan); (d) full I/O, hidden layer, memory structure (Elman). In all of these examples the architectures are supposed to accommodate discrete input

Results

The desired behaviour has emerged from the evolutionary process in a relatively limited number of generations for most of the experimental setups. An example of the evolved behaviour is visible in Figure 6.4, while a summary of the main results (average fitness, best fitness, average success rate, best success rate) is included in Table 6.8. Looking at the data presented in this table, we can see that, among all the simulations using architectures allowing yaw, pitch, and roll (A5, A6, A11, A12, A17, A18, A23, and A24), A5 is the one that has produced the best results both looking at the average success rate and to the efficiency of the best controllers evolved.

Table 6.8: Simulations A: resume of the main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
A1	986.2	1,425.6	80.26	100
A2	963.79	1,425	78.25	99.9
A3	856.52	1,393.4	62.84	99.15
A4	780.25	1,262.4	44.75	78.69
A5	711.82	1,310.5	46.41	93.28
A6	383.02	743.46	0.71	6.09
A7	988.63	1,425.6	79.52	100
A8	976.43	1,428.3	79.94	100
A9	827.56	1,386.3	61.61	99.72
A10	917.51	1,403.8	68.09	99.6
A11	693.23	1,267.9	40.95	87.39
A12	599.58	1,045.8	22.79	46.84
A13	974.59	1,421.3	78.48	100
A14	926.91	1,415.2	74.65	99.91
A15	715.75	1,266.4	39.82	82.8
A16	452.43	824.13	0.44	8.05
A17	389.93	824.98	2.1	17.27
A18	353.63	705.16	0.09	2.64
A19	790.91	1,345.8	59.8	97.74
A20	779.04	1,340.1	55.67	96.92
A21	447.12	916.26	9.04	32.63
A22	397.22	777.72	1.38	8.81
A23	319.87	715.71	0.83	8.54
A24	320.83	639.66	0.02	0.7

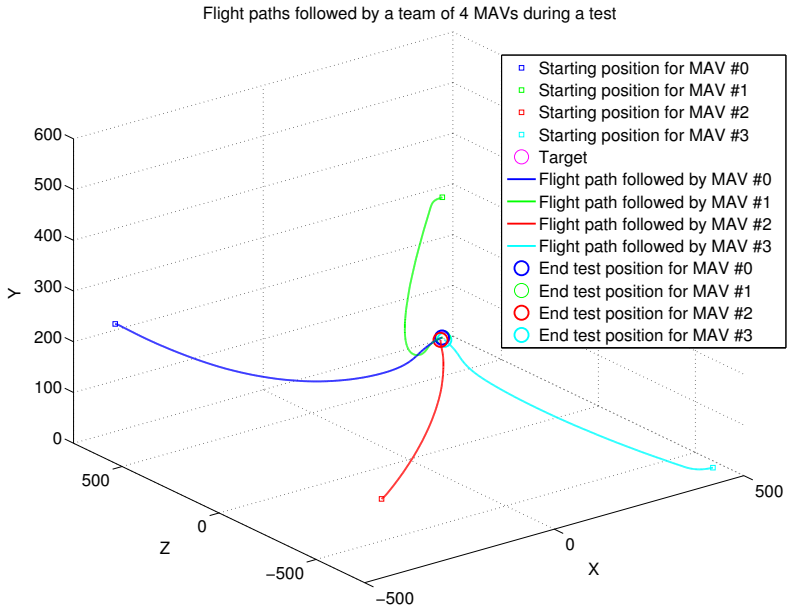


Figure 6.4: Simulation A5: flight paths followed by four individual MAVs sharing the same evolved controller, moving towards the target from the four corners of the environment

Figure 6.5 displays the average and maximum fitness values across the $20,000$ generations elapsed. Both the curves tend to reach a steady state in between eight and nine thousands generations. It is interesting to see how the values scored by the best individuals are nearly twice as big as those registered by the average ones. This effect is most likely due to the small population size adopted, as this makes the impact of the individuals with random connection weights and biases introduced at any new generation, generally performing badly, significantly higher.

Figure 6.6 shows the average success rate across the entire population at the various generations, along with the success rate scored by the best controllers. The stabilisation is achieved in $8-9,000$ generations for the average values only. The best controllers keep evolving after that stage and the related curve takes slightly longer (about 11,000 generations in total) to reach its maximum value (which corresponds to over 93%).

Figure 6.7 focuses on the amount of energy left to the MAV when it has operated the end-operation unit, averaged for the various tests. As expected the trend is toward an increasing of the amount of energy saved across the generations. The final state reached is not particularly stable, due to the fact that the target is always

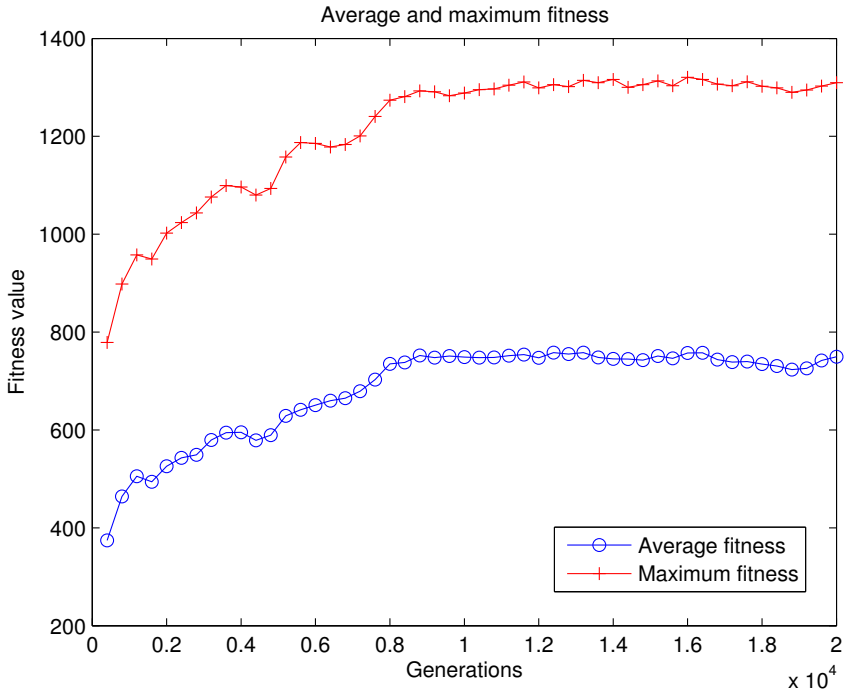


Figure 6.5: Simulation A5: average and best fitness (average of 10 evolutionary runs)

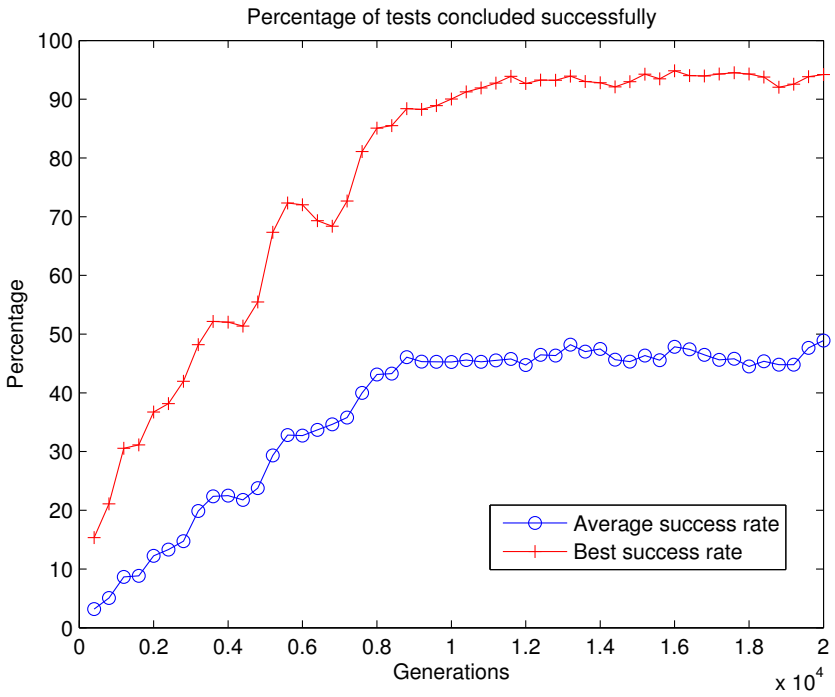


Figure 6.6: Simulation A5: percentage of successful tests (average of 10 evolutionary runs)

deployed in random positions at the beginning of each test. This fact results in the MAVs flying for different lengths of time, depending upon where the target has been deployed. Using teams made of several MAVs could have overcome this problem, but this is not necessarily the case, as the evolutionary process often tends to design controllers that have "preferential" ways to approach the target (*e.g.* always navigate to approach it from the right).

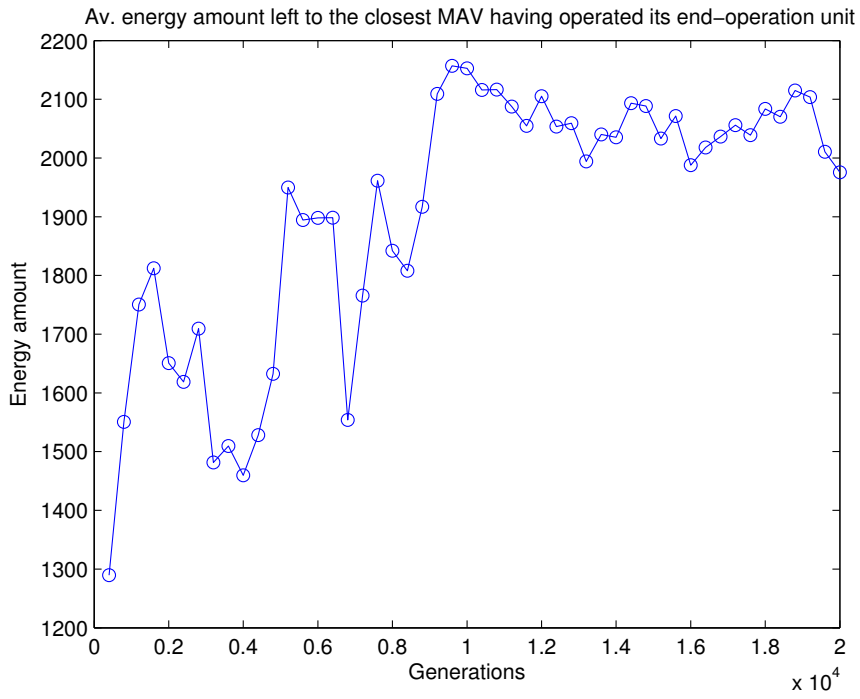


Figure 6.7: Simulation A5: amount of energy left to the MAV activating its end-operation output unit closest to the target (average of 10 evolutionary runs)

The comparative analysis of the results will mainly focus on the data obtained by the neural architectures that process the real time input without relying on any memory structure, as they have clearly outperformed their competitors. Figure 6.8 shows a comparison between architectures 1-12 for what concerns the success rate obtained by the best individual at the last generation.

Looking at this in more detail, it is noticeable that the sets of rotations made available to the MAVs have a significant impact on their performance. When only yaw is permitted (thus recreating a 2D scenario) controllers that are able to perform the desired task with a 100% accuracy level emerge in just a few generations. The same applies, although following a slightly slower process, when both yaw and pitch are used. The introduction of roll has a significant impact and leads to worse

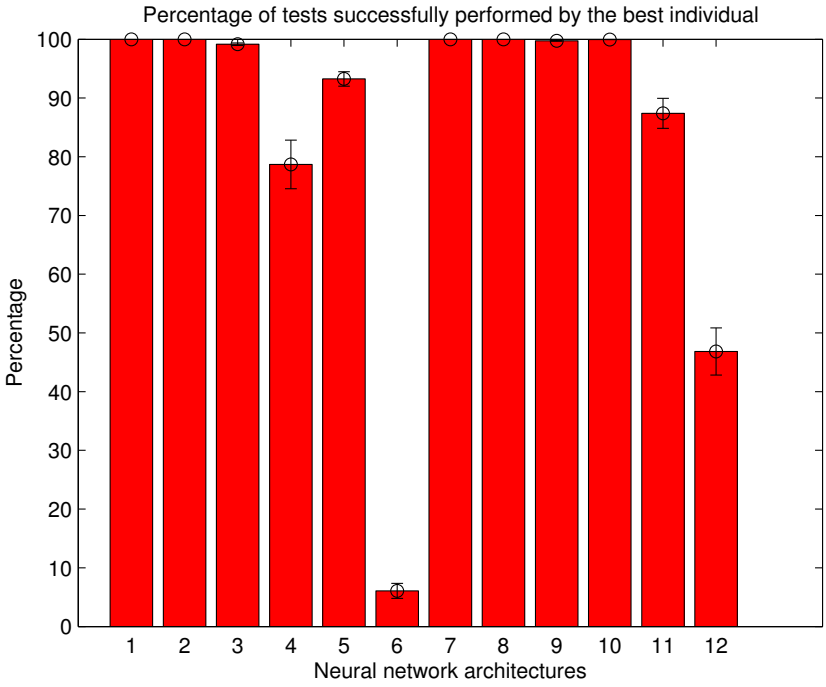


Figure 6.8: Bar plot displaying the maximum success rate obtained by the best individuals evolved with the various controller architectures deprived of memory components. The standard error - calculated as standard deviation divided by the number of evolutionary seeds ran - is also shown (average of the last 10 generations, based on 10 evolutionary runs)

performances, these are acceptable for both architectures 5 and 11 (respectively 93.28% and 87.39% as maximum success rate), but are far below average for 6 (6%) and 12 (46.8%). Figures 6.9 and 6.10 compare the success rate of the various architectures, grouped by the set of rotations they have available. The graphs show the average and maximum success rate respectively.

For what concerns input encoding, discretised input has always led to better or at wort equal results compared with continuous encoding. The difference is particularly evident if contrasting the results scored by architectures 5 and 6, or 17 and 18. Figures 6.11 and 6.12 contrast the average and best success rates obtained on average by architectures using discrete input against those relying on information encoded in a continuous fashion.

The use of a hidden layer has proven to be beneficial for the performances of the controllers when no memory structures are employed. With simulations A5 and A11 being exceptions to this (in this case, architecture 5, without a hidden layer, has performed better than its counterpart having internal neurons). The controllers

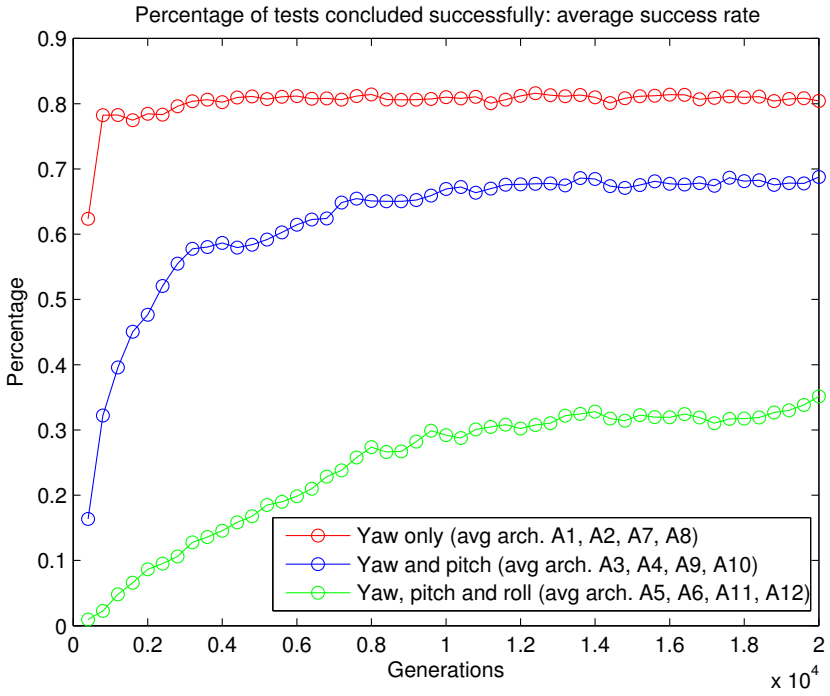


Figure 6.9: Simulations A: comparison for the average success rate in function of the sets of rotations available to the aircraft (average of 10 evolutionary runs)

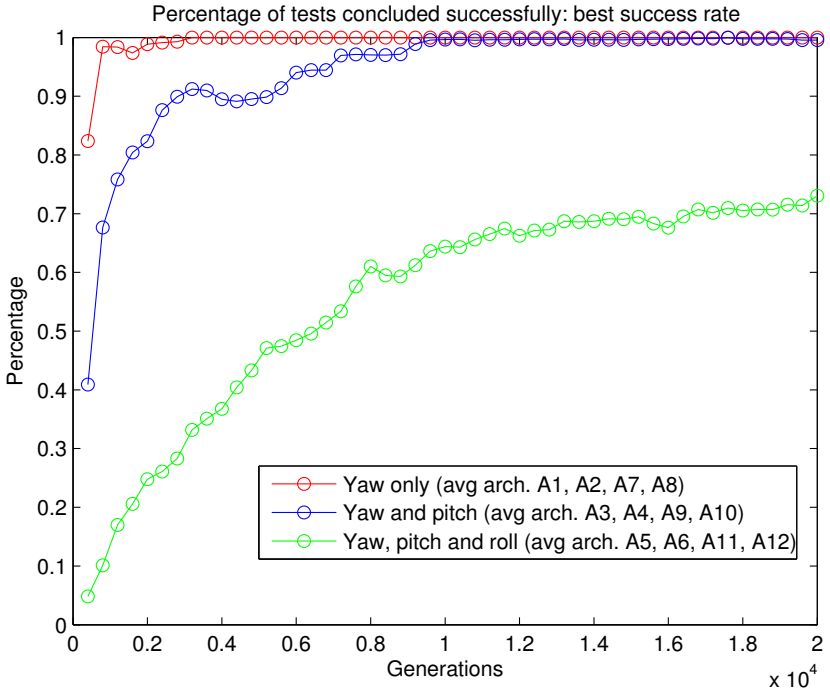


Figure 6.10: Simulations A: comparison for the maximum success rate in function of the sets of rotations available to the aircraft (average of 10 evolutionary runs)

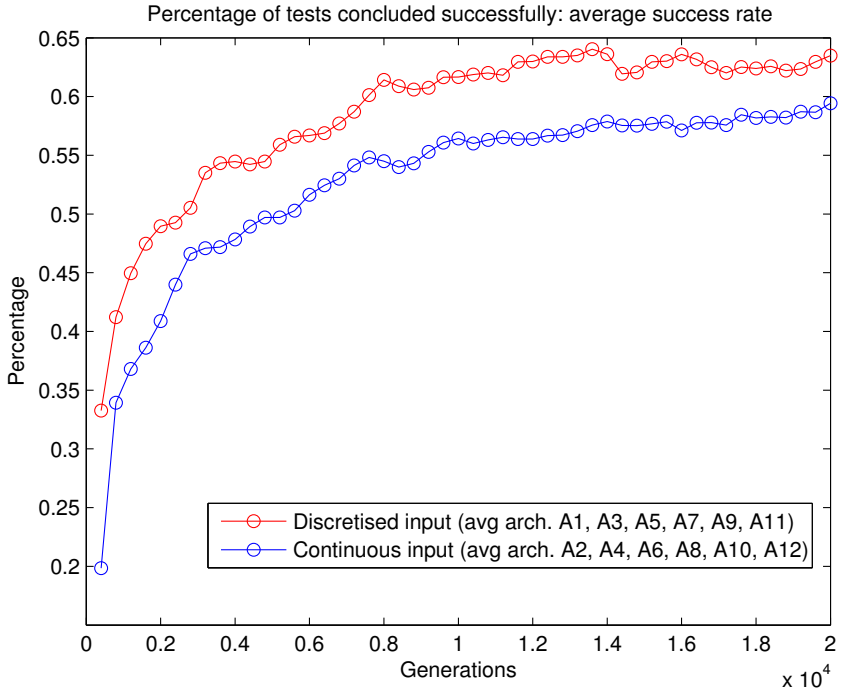


Figure 6.11: Simulations A: comparison for the average success rate in function of the kind of encoding used for the input information (average of 10 evolutionary runs)

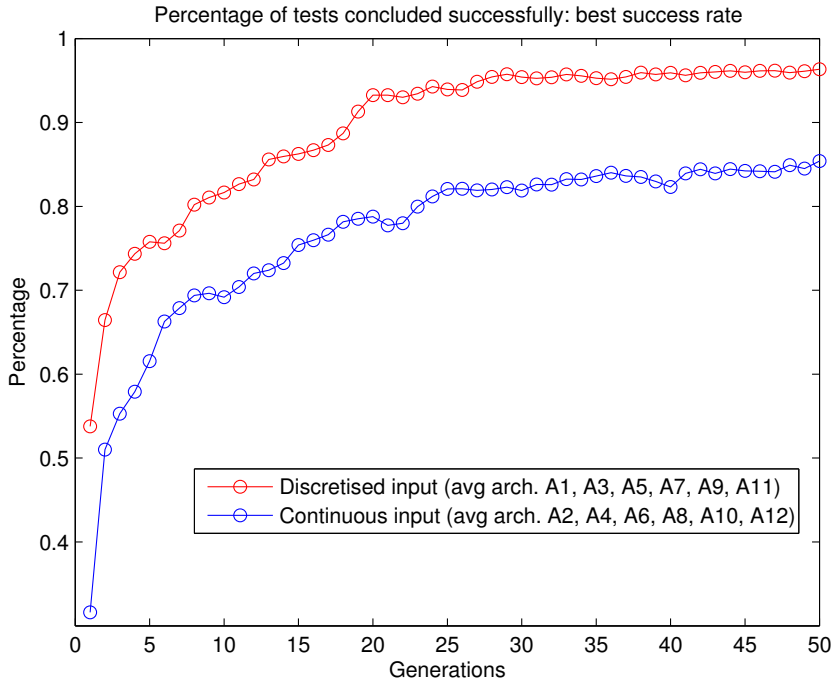


Figure 6.12: Simulations A: comparison for the maximum success rate in function of the type of encoding used for the input information (average of 10 evolutionary runs)

with a layer of internal units have in fact outperformed the two-layer networks. The situation is completely different when memory is used. In this case (but again with one exception, constituted by the comparison between simulations A16 and A22) the lack of a hidden layer seems to be beneficial. As before, this result could be explained by the increase in dimensionality of the search space generated by the addition of ten more neurons, with respective synaptic connections and biases. A graphical summary of these results, highlighting the similarity in the data obtained, can be found in Figures 6.13 and 6.14.

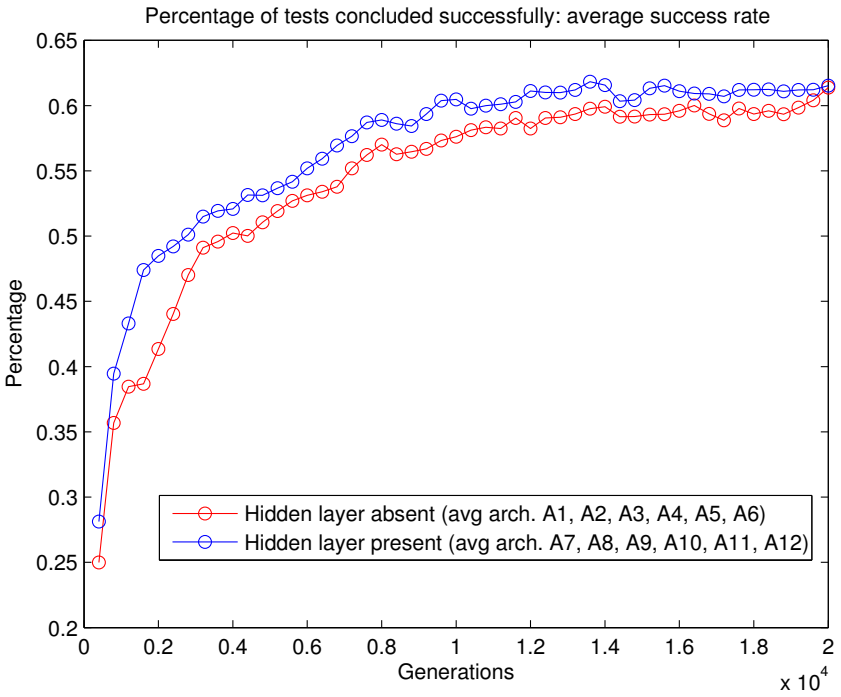


Figure 6.13: Simulations A: comparison for the average success rate in function of the presence/absence of a hidden layer (average of 10 evolutionary runs)

Most likely due to the simplicity of the elaborated scenario, which does not require any additional ability for the MAV than just pointing to the target area, the architectures providing memory to the controller have not generated any benefits. Worse still, the controllers implementing Elman and Jordan networks have scored significantly lower success rates than those based on purely feed-forward networks. The performances are comparable only for the simplest situation (no pitch and no roll), and partially for the second simplest setup (yaw and pitch, but no roll). However this effect is probably due to the significantly larger search space the evolutionary algorithm has to investigate in order to find an optimum point when the

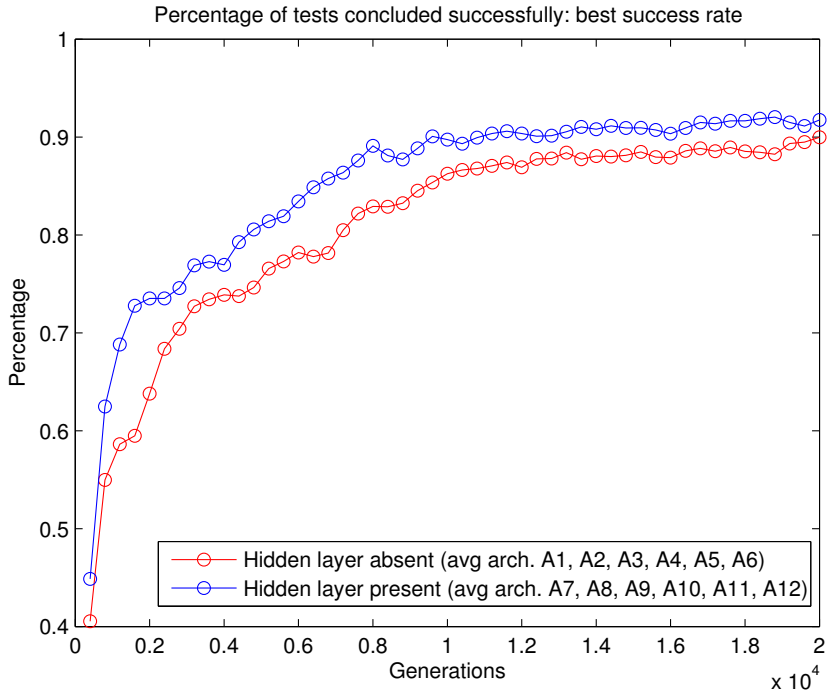


Figure 6.14: Simulations A: comparison for the maximum success rate in function of the presence/absence of a hidden layer (average of 10 evolutionary runs)

additional connection weights associated to the memory structures come into play.

Single and multi-threading performance comparison

One of the experimental setups described in this chapter has been exploited for carrying out further, and more "technical", experimentations. As we have mentioned above, Gautier [131] has investigated the adoption of a multi-threading programming approach to the 2D computer model presented in chapter 5, finding strong evidence for an improvement of the evolutionary algorithm speed when such methodology is employed. Utilising Gautier's findings, we have therefore decided to optimise the source code of the 3D simulator in order to allow it to benefit from the possible execution on computers driven by multi-core processors (as it is the case for the grid on which we have run our simulation experiments).

From a technical point of view, the adaptation of our simulator into a multi-thread application has been made possible by the use of QtConcurrent[10], a framework within the Qt libraries which provides high-level APIs that make it possible to write multi-threaded programs without the need to manually write low-level thread-

[10] `http://doc.qt.nokia.com/latest/qtconcurrent.html`

ing primitives.

We have performed a systematic comparison between identical simulations[11] executed in single and multi-thread mode on different machines [322]. In this experiment we have used one of the machines from the P-ARTS system, plus a standard desktop computer (an Apple Mac Pro belonging to the "early 2008" generation, equipped with a single $2.8GHz$ quad-core Intel Xeon processor).

The simulation used as a reference is A9, which has been evolved for ten seeds, each of these 50 generations long. Table 6.9 shows the results obtained in terms of execution times (measured in milliseconds).

Table 6.9: Simulation A9: comparison between single and multi-threading in terms of execution times [MP: Mac Pro, XS: Xserve, ST/MT: single/multi-thread] (measured in *msec*)

Seed	MP-ST (s)	XS-ST (s)	MP-MT (s)	XS-MT (s)	ΔMP (%)	ΔXS (%)
1	115,959	116,370	34,818	27,943	-69.97	-75.99
2	154,174	154,548	70,539	15,467	-54.25	-89.99
3	155,169	155,705	38,774	32,041	-75.01	-79.42
4	148,620	149,216	43,214	34,274	-70.92	-77.03
5	155,831	159,446	75,149	21,021	-51.78	-86.82
6	211,174	212,136	44,934	24,225	-78.72	-88.58
7	165,884	166,569	61,842	45,306	-62.72	-72.8
8	216,879	217,586	52,754	38,534	-75.67	-82.29
9	69,081	69,417	25,706	18,206	-62.79	-73.77
10	102,108	102,555	36,730	20,634	-64.03	-79.88
Avg	149,487.9	150,355	48,446	27,765.1	-66.59	-80.66

Despite the differences in the frequency at which the two processors run and the variations in the two internal computer architectures, the single-threaded version of the simulator has obtained virtually identical execution times on the two computers used.

Figures 6.15 and 6.16 compare the execution times obtained by the simulation that was run in single and multi-threading configurations on the two machines used for the experiments.

First of all, as expected, the execution time drops when moving from single to

[11] With the word "identical" we refer to simulations for which the random number generator has been initialised on the same value, thus producing identical sequences of pseudo-random numbers.

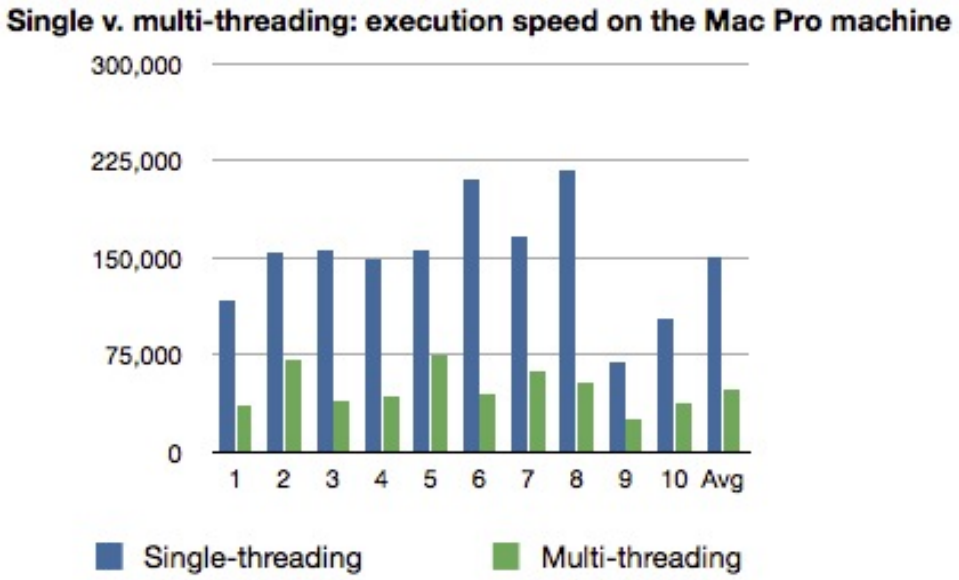


Figure 6.15: Comparison between single and multi-threading in terms of execution speed (measured in *msec*) on the Apple Mac Pro machine

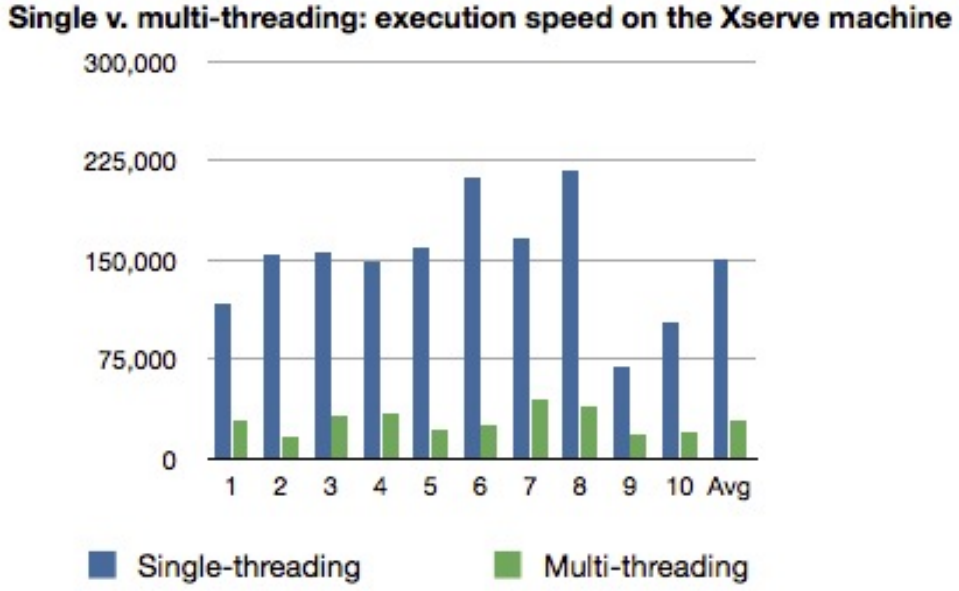


Figure 6.16: Comparison between single and multi-threading in terms of execution speed (measured in *msec*) on the Apple Xserve machine

multi-threading. The improvement has been significantly greater for the Xserve machine than for the Mac Pro one because of the larger number of processing cores available (eight rather than four). On the Mac Pro machine the execution time has been reduced by 66.59%, while on the Xserve machine the improvement has been calculated as −80.66%. These are slightly further from the theoretical maxima, that are respectively −75% and −87.5%, but considering that a certain overhead has to be taken into account and that multi-threading has been implemented in the easiest way possible (which is not likely to be the most efficient way) the results can be considered rather good.

6.4.2 Moving target (simulations B)

The second experimental setup prepared (simulations B, presented in Ruini *et al.* [323]) is similar to the previous one. The only difference consists in the fact that now the designated target can move away from the approaching MAV. At any time step the target can be in either one of two different states: "MAV detected" or "MAV not detected". When in "MAV not detected" mode, the target scans its surroundings at any time step - before the aircraft moves - looking for a vehicle within a $35GU$ distance from its centre. In the case of this condition being satisfied, the target switches to "MAV detected" mode with probability 0.5. When the target is in the "MAV detected" state, it has to move into a new place. This movement is alternated with the one made by the MAV at any time step.

Table 6.10 shows the 26 different locations the target can chose between when deciding in which direction to move, based on its current (X, Y, Z) coordinates. These points are calculated in order to ensure they are equidistant from its centre (*i.e.* points on the surface of an imaginary sphere sharing its origin with the centre of the target and having a ray equal to its movement speed) and representative of the entire neighbourhood area the target could end up in. The target does not have any kind of preference and its movements are not affected by inertia: at any time step it simply moves to the position which will maximise its distance from the MAV.

Table 6.10: Simulations B: possible movement destinations for the target

New X	New Y	New Z
X	Y	$Z + d$
$X + d * \cos(\frac{\pi}{4})$	Y	$Z + d * \sin(\frac{\pi}{4})$
$X + d$	Y	Z
$X + d * \cos(\frac{3}{4}\pi)$	Y	$Z - d * \sin(\frac{3}{4}\pi)$
X	Y	$Z - d$
$X - d * \cos(-\frac{3}{4}\pi)$	Y	$Z - d * \sin(-\frac{3}{4}\pi)$
$X - d$	Y	Z
$X - d * \cos(-\frac{\pi}{4})$	Y	$Z + d * \sin(-\frac{\pi}{4})$
X	$Y + d$	Z
X	$Y + d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(0)$
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{\pi}{4})$	$Y + d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(\frac{\pi}{4})$
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{\pi}{2})$	$Y + d * \cos(\frac{\pi}{4})$	Z
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{3}{4}\pi)$	$Y + d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(\frac{3}{4}\pi)$
X	$Y + d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(\pi)$
$X - d * \sin(\frac{\pi}{4}) * \sin(-\frac{3}{4}\pi)$	$Y + d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(-\frac{3}{4}\pi)$
$X - d * \sin(\frac{\pi}{4}) * \sin(-\frac{\pi}{2})$	$Y + d * \cos(\frac{\pi}{4})$	Z
$X + d * \sin(\frac{\pi}{4}) * \sin(-\frac{\pi}{4})$	$Y + d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(-\frac{3}{4}\pi)$
X	$Y - d$	Z
X	$Y - d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(0)$
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{\pi}{4})$	$Y - d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(\frac{\pi}{4})$
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{\pi}{2})$	$Y - d * \cos(\frac{\pi}{4})$	Z
$X + d * \sin(\frac{\pi}{4}) * \sin(\frac{3}{4}\pi)$	$Y - d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(\frac{3}{4}\pi)$
X	$Y - d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(\pi)$
$X - d * \sin(\frac{\pi}{4}) * \sin(-\frac{3}{4}\pi)$	$Y - d * \cos(\frac{\pi}{4})$	$Z - d * \sin(\frac{\pi}{4}) * \cos(-\frac{3}{4}\pi)$
$X - d * \sin(\frac{\pi}{4}) * \sin(-\frac{\pi}{2})$	$Y - d * \cos(\frac{\pi}{4})$	Z
$X + d * \sin(\frac{\pi}{4}) * \sin(-\frac{\pi}{4})$	$Y - d * \cos(\frac{\pi}{4})$	$Z + d * \sin(\frac{\pi}{4}) * \cos(-\frac{3}{4}\pi)$

Assuming M_s the speed of the MAV, five different evolutions have been performed for each architecture, with the target moving respectively at a speed T_s equal to $\frac{M_s}{5}$, $\frac{M_s}{4}$, $\frac{M_s}{3}$, $\frac{M_s}{2}$ and M_s. Differently than what we have done with the 2D model, we have decided to test a new condition in which the target and the aircraft move at the same speed. At the same time we have decided to ignore moving speeds of the target lower than $\frac{M_s}{5}$, as we could not find any significant difference in the performance of the controllers evolved in 2D scenario dealing with targets slower than a certain threshold.

This experimental setup has been tested with 12 different topologies of neurocontrollers (1-12 introduced in previous paragraph), running twelve simulations (B1-B12) five times each (one for each possible target's speed). Considering the poor

performances generated in the experimental setup A by the architectures including memory structures, these have been ignored at this stage. In principle, it is true that memory structures could be extremely useful in tasks like target tracking, but this is only valid when the autonomous vehicles have to "understand" the path the target is following in order to predict the target's moves and attempt to anticipate them. Based on the movement rules for the target we have listed above, in our opinion this one would not be the case. As the target can move, at every time step, in any direction it likes, there is very little room for anticipation, thus very little to be gained by the implementation of a memory structure into the neural network controller.

To cope effectively with the very high degree of variance in the data that has been highlighted by some preliminary tests, the amount of evolutionary seeds elaborated has been extended to 20.

Results

The results presented in this section concern simulations C11 and C5 (relying on architectures 11 and 5 respectively), those that generated the best results in simulations A respectively for the subsets of networks with and without a hidden layer, and that allow the aircraft to yaw, pitch and roll.

As expected, varying the speed of the target has affected the MAVs' performances. The results are similar to those obtained with our previous work on the 2D simulator (see section 5.4.3). Apart from a general performance decrease if compared to the scenario in which the target is static, what we can observe again is the emergence of a threshold value for T_s that, once exceeded, makes on average the MAV unable to succeed in the task with a high degree of accuracy anymore. For targets moving at $\frac{M_s}{5}$, $\frac{M_s}{4}$ and $\frac{M_s}{3}$, the success rate for the best individual of the population is over 85%, both when the architecture is lacking a hidden layer (Figure 6.17), and when the network can rely on this additional computational capability (Figure 6.18).

More specifically, the percentage of tests concluded successfully ranges from

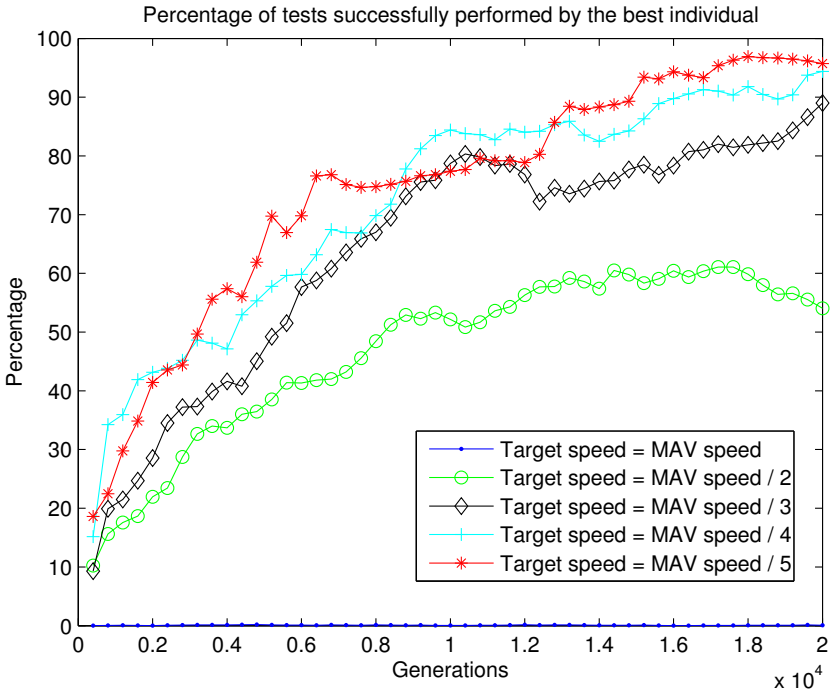


Figure 6.17: Simulation B5: comparison for the success rate (average of 20 evolutionary runs)

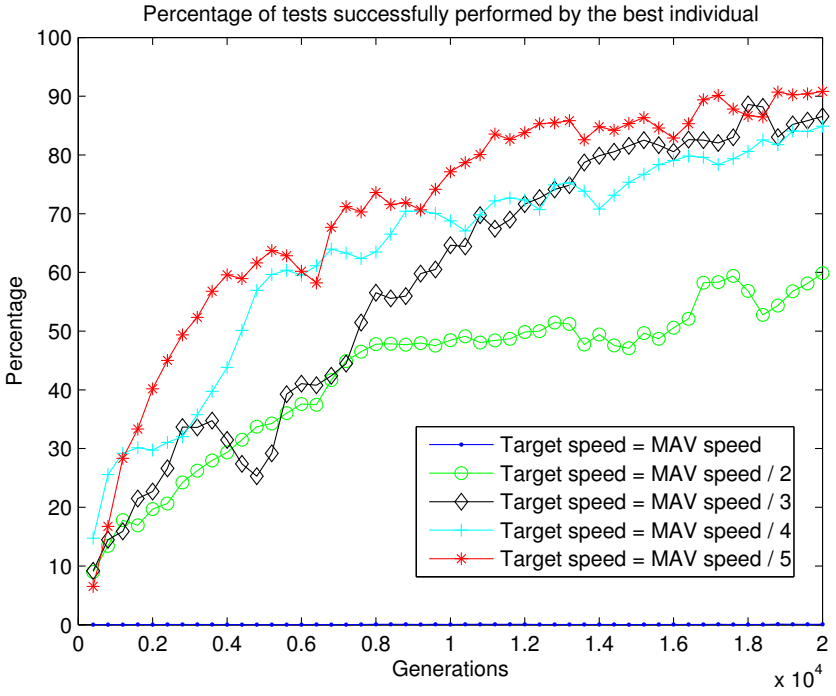


Figure 6.18: Simulation B11: comparison for the success rate (average of 20 evolutionary runs)

86.57% to 95.71%. A target moving at $\frac{M_s}{2}$ results in a much more difficult task for MAVs, with the two simulations respectively scoring 54.03% and 59.85%. As expected, when the target and the MAV move at the same speed ($T_s = M_s$), the latter practically never succeed in the task. As nature teaches us, it can indeed only happen by chance (*e.g.* because the prey gets stuck in some particular area of the environment) that a hunter could reach his prey if they both run at the same speed. This is even more true when the prey is hardwired for the best escaping efficiency possible as in our experimental setup.

Tables 6.11 and 6.12 summarise the results generated by simulations for simulations B5 and B11 respectively. What is interesting to see is how architecture 5 performed better than architecture 11 in terms of success rate also in this scenario, despite being more limited in terms of "computational capabilities."

Table 6.11: Simulation B5: resume of the main results (average of the last 10 generations, based on 20 evolutionary runs)

T_s	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
M_s	547.89	953.116	0.000002	0.0008
$\frac{M_s}{2}$	663.47	1,149.3	22.31	54.03
$\frac{M_s}{3}$	736.55	1,297.2	43.26	89
$\frac{M_s}{4}$	747.49	1,309.2	47.23	94.4
$\frac{M_s}{5}$	728.34	1,323.9	48.94	95.71

Table 6.12: Simulation B11: resume of the main results (average of the last 10 generations, based on 20 evolutionary runs)

T_s.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
M_s	553.4	947.63	0.000002	0.0006
$\frac{M_s}{2}$	659.69	1,150.5	22.43	59.85
$\frac{M_s}{3}$	724.51	1,269.4	40.86	86.57
$\frac{M_s}{4}$	731.98	1,280.5	43.53	84.88
$\frac{M_s}{5}$	731.99	1,296.3	44.38	90.83

6.4.3 Implicit cooperation (simulations C)

Simulations C involve cooperation among the MAVs, replicating in the new three-dimensional model the scenario described in section 5.4.4, except for two main differences: 1) the absence of obstacles into the environment; 2) the "stationarity" of the target which, in this scenario, is not able to move. The setup is more complicated than the one presented in simulations A and B because of the fact that rather than one single MAV, teams made by four MAVs sharing the same controller are now employed. In order to accomplish the task, at least two MAVs need to approach the target area and activate their Boolean neurons in quick succession. From a technical point of view, these modifications in the experimental setup have been coped with imposing that the target can be, at any given time, in either one out of two possible alternative states: either "healthy" or "damaged." The target starts each test as "healthy" but can be damaged later on during the task. The damaging of the target takes place when a MAV activates its Boolean unit within $15GU$ from the target centre. The test is considered to be successfully concluded when another MAV manages to correctly approach the target (by activating the end-operation unit when close enough to it) when the latter is still "damaged". In order to guarantee the semi-synchronicity of the process, the target gets restored (*i.e.* switches back to the "healthy" status) after 200 time-steps of "damaged" mode.

The controllers used in this scenario (C1-C12, built on top of the 1-12 controllers seen at work for simulations A) have been modified to deal with the new conditions by introducing two new input units, specifically two Boolean neurons. These units respectively get activated when the target is in "damaged" status, and when a teammate is perceived within a certain distance range ($60GU$) from the aircraft embedding the controller. Figure 6.19 shows the topology of two of the modified neurocontrollers, C5 and C11, derived from architectures 5 and 11 respectively.

The motion of the MAVs has been modified as well, in order to allow for wider rotation angles. At any time-step they now move just $1GU$ rather than 2 as in the previous setups, thus indirectly doubling their manoeuvring range. The amount of energy available to the MAVs at the beginning of a test has been increased from

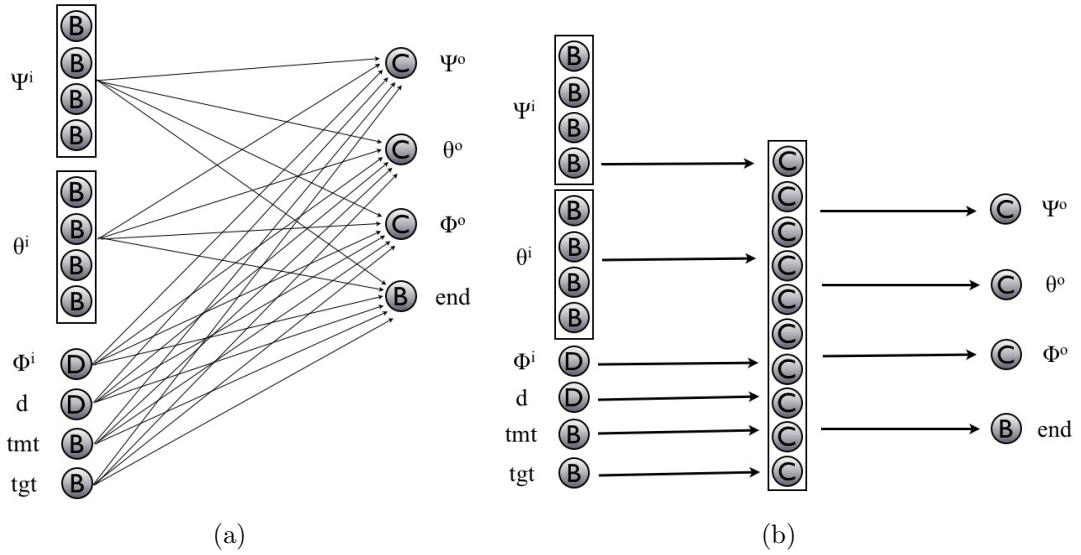


Figure 6.19: Graphical representation of the NN controllers (a) C5; (b) C11. The two new input units, labelled "tmt" and "tgt" respectively, are those in charge of informing the controller about the presence of a teammate within a certain distance range and signalling the status of the target ("healthy" or "damaged")

$5,000EU$ to $15,000EU$, in order to ease the evolution of the behaviour we have seen emerging in the 2D model (with one of the MAVs flying for a certain amount of time around the target waiting for a teammate to arrive).

The fitness function also has required a minor modification in order to cope with the new dynamics and the fact of having MAV teams rather than individual MAVs involved in the evolutionary process. Compared to Equation 6.5, Equation 6.6 introduces a new parameter, γ, which represents the amount of tests concluded half-successfully, *i.e.* with one single MAV managing to properly approaching the target. In this way the MAVs are also rewarded for approaching successfully the target, even without being able to conclude a test successfully. The rewards, obviously, become higher if they manage to perform the cooperative operation required. The α parameter has been modified to $\langle\alpha\rangle$, thus indicating that it is now representing the average distance traveled by all the four MAVs during the all tests.

$$f = \langle\alpha\rangle + 50\gamma + 100\beta \tag{6.6}$$

As in simulation B, the experiments have been carried out using 12 neural architectures only, *i.e.* all of those that do not rely on short-term memory elements and

modified with the introduction of two additional Boolean input units as described above.

Results

The strategy evolved by the best individuals is the same one observed in section 5.4.4. To be precise, the first MAV arriving in the proximity of the target does not activate its Boolean unit immediately. Instead it starts flying around the target area, waiting for another aircraft to arrive. When a teammate finally arrives, they both get as close as possible to the target and activate their Boolean neurons thus accomplishing the test. An example of this behaviour, observed from a top view, can be found in Figure 6.20. Figure 6.21 provides instead a three-dimensional view of the flight paths followed by a team of four MAVs driven by a different controller, but performing the same task.

Unfortunately, this behaviour has demonstrated to be particularly hard to be achieved in the 3D model. Simulations carried out using controllers based on architectures 1, 2, 7, and 8 (no pitch and roll rotations available) have generally generated decent performances (success rates for the best individual have been 89.71%, 39.9%, 93.06%, and 75.38% respectively). Nonetheless the performance of the MAVs has dramatically decreased when the wider sets of rotations available to the MAVs have been introduced. Figure 6.22 shows the results obtained evolving controllers based on architecture 5 (yaw, pitch, and roll possible). In this case, the best team can successfully conclude the task 12.79% of the time (just 2.04% for the average team), and simply manages to put the target in "damaged" status the remaining 58.23% of times.

Table 6.13 summarises the results obtained for the 12 neural architectures tested.

6.5 Experiments on incremental evolution

In section 2.5 we have introduced the topic of incremental evolution. The discussion we carried out concluded that to this day is still extremely difficult to provide a definitive answer to the dilemma of whether incremental evolution is "better" or

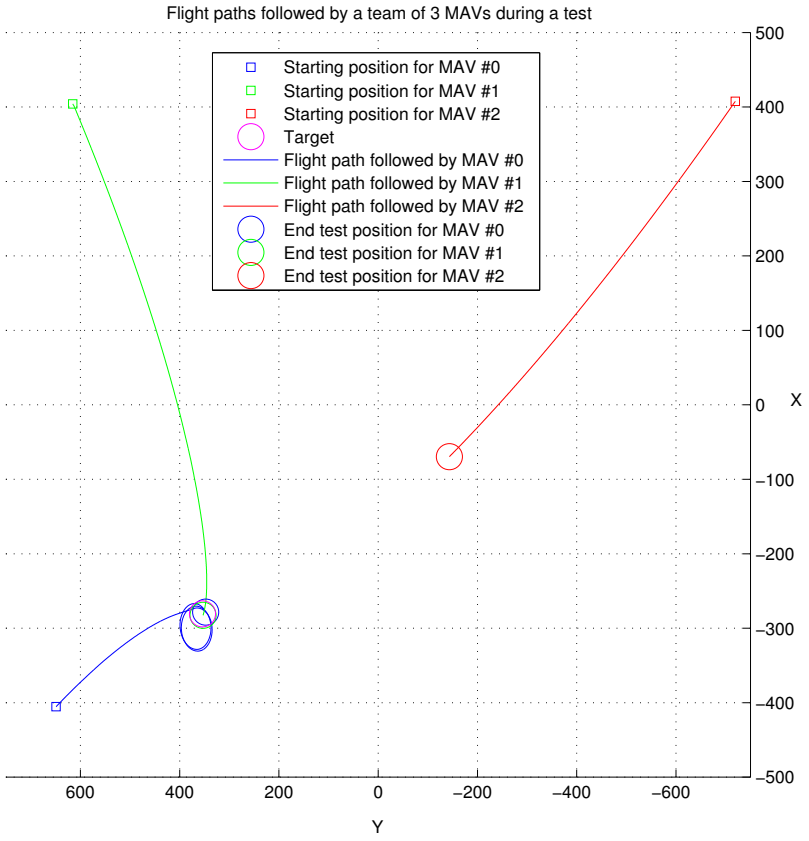


Figure 6.20: Simulation C2: flight paths followed by three individual MAVs sharing the same controller, and put into the environment at the same time, but moving from different starting positions. In this case MAV #0 is the first to arrive in the proximity of the target area in $(-280, 360)$. It then starts flying around it, while waiting for the arrival of a teammate in order to successfully conclude the test

Table 6.13: Simulations C: resume of the main results (average of the last 10 generations, based on 20 evolutionary runs)

Sim.	Av. fitness	Max fitness	Av. succ. rate [%]	Max succ. rate [%]	Approach rate [%]
C1	1,106.8	1,701	46.72	88.14	36.69
C2	1,011.4	1,505.3	31.64	58.18	43.16
C3	742.58	1,155.2	9.23	33.61	61.8
C4	571.6	843.57	0.81	6.41	64.51
C5	429.63	662.93	0.59	4.79	34.24
C6	377.15	593.19	0.15	2.05	27.31
C7	1,008.4	1,628.3	42.66	84.25	29.26
C8	1,228.2	1,834.6	53.91	91.84	17.54
C9	696.34	1,106	8.52	31.99	56.72
C10	701.98	1,016.8	1.36	12.03	70.09
C11	367.98	607.84	0.27	3.26	35.89
C12	589.48	893.63	2.14	13.56	53.74

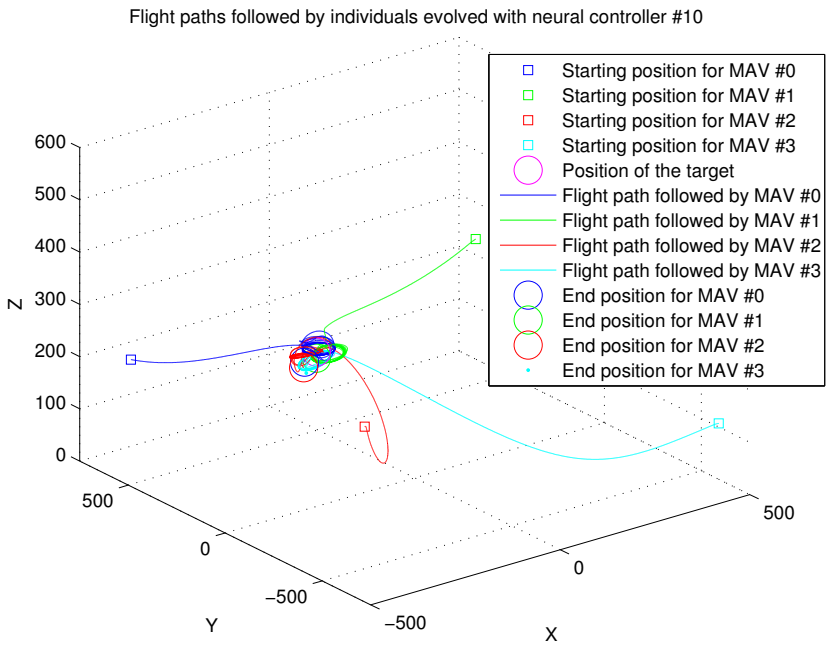


Figure 6.21: Simulation C5: flight paths followed by four individual MAVs sharing the same controller

not than direct evolution. The study presented in this section, in which incremental and non-incremental evolution approaches are compared for the 3D model discussed so far, aims to give an additional contribution to the topic with the awareness that it is difficult to provide any conclusive answer due to the amount of variables that should be taken into account for a comprehensive and definitive analysis.

6.5.1 Basic results for A, B, and C setups

For the comparative analysis between incremental and non-incremental evolution four new evolutionary processes have been run spanning the three experimental setups detailed in the previous pages. All the architectures have been subject to this test, except for those endowing memory structures (apart from those already mentioned, detailed reasons for this choice have been explained in Ruini *et al.* [324]).

Evolutions in scenarios A and B have been performed for different numbers of generations according to the complexity (intended as the number of connection weights) of the neural network controllers used: 5,000 generations for architectures 1, 2, 7, and 8; 10,000 generations for topologies 3, 4, 9, and 10; 20,000 generations for controllers 5, 6, 11, and 20.

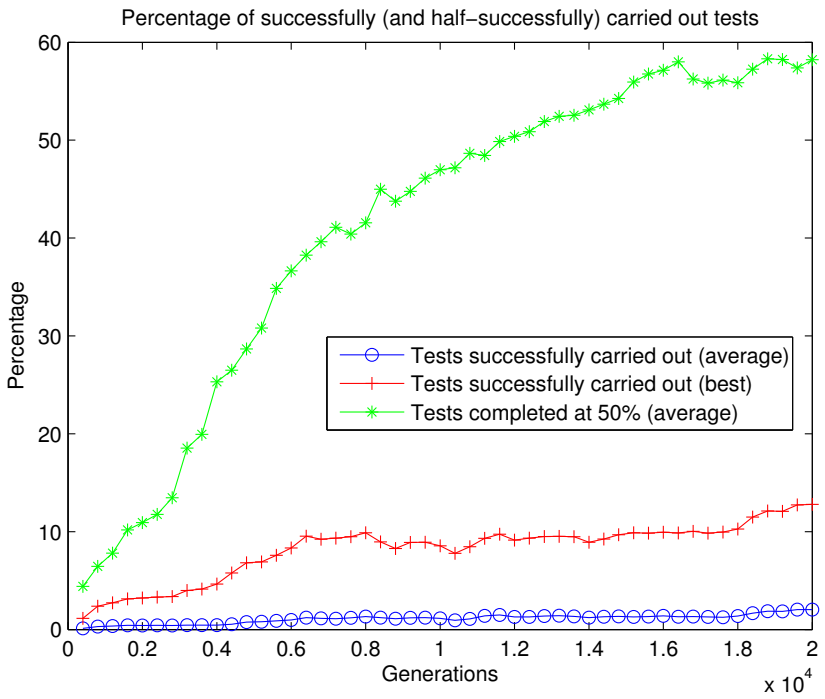


Figure 6.22: Simulation C5: percentage of successful and half-successful tests for the average and the best MAVs (average of 20 evolutionary runs)

Given M_s the speed of the MAV, different evolutionary processes have been elapsed for each architecture in scenario B, with the target moving at speeds T_s equal to $\frac{M_s}{2}$ and $\frac{M_s}{3}$ respectively.

The results obtained for scenario A are summarised in Table 6.14, while Tables 6.15 and 6.16 show the outcome of the simulations carried out within the B scenario for T_s equal to $\frac{M_s}{2}$ and $\frac{M_s}{3}$ respectively.

The two tables show the average fitness values scored by the entire population at the end of the evolution, as well as the maximum (*i.e.* the best individual's fitness). The average and best success rates (intended as percentage of tests concluded successfully) are also reported. Since at any given generation five new random individuals are introduced (corresponding to 16.66% of the entire population, which consists in 30 individuals), the average success rate, as expected, never gets better than 83.34%.

The results obtained are not surprising. As expected, because of the previous experiments carried out, the complexity of the controllers used affects the performance of the MAVs. For the simplest 2D scenario (architectures A1, A2, A7, and A8) all the topologies produce good results, leading to MAVs able to successfully perform

Table 6.14: Simulations A: resume of the new main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
A1	978.49	1,420.9	79.76	100
A2	989.28	1,426.5	81.71	100
A3	904.17	1,413.3	69.63	99.92
A4	858.61	1,331.4	56.77	89.16
A5	749.73	1,309.5	48.91	94.19
A6	602.73	1,050.9	18.57	46.93
A7	1,005.3	1,428.6	82.35	100
A8	997.06	1,430.6	82.36	100
A9	881.67	1,399.3	66.65	99.9
A10	934.68	1,413.7	72.47	99.86
A11	688.29	1,272	42.09	85.56
A12	6423.88	1,111.8	31.04	65.56

Table 6.15: Simulations B (non-incremental setup, $T_s = \frac{M_s}{2}$): resume of the new main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
B1	976.21	1,421	80.01	100
B2	976.97	1,420	78.23	100
B3	855.36	1,386.6	59.82	99.79
B4	744.23	1,252	40.76	81.97
B5	645.6	1,158.3	28.49	59.69
B6	551.61	996.78	15.97	39.6
B7	983.84	1,421.7	79.14	100
B8	933.41	1,430.4	81.4	100
B9	756.33	1,323.7	50.35	96.42
B10	851.72	1,336.5	53.76	87.24
B11	643.11	1,126.9	18.07	58.29
B12	700	1,162.4	33.84	66.09

Table 6.16: Simulations B (non-incremental setup, $T_s = \frac{M_s}{3}$): resume of the new main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
B1	979.82	1,422.8	79.48	100
B2	991.54	1,426.2	81.77	100
B3	889.18	1,407.7	69.18	100
B4	787.94	1,258.3	43.85	79.98
B5	673.22	1,212.7	33.54	71.37
B6	596.25	1,063.5	20.01	52.55
B7	995.34	1,428.3	80.81	100
B8	994.91	1,431.4	81.53	100
B9	806.97	1,354.3	54.97	97.85
B10	951.20	1,417	72.76	100
B11	577.7	1,090.6	19.55	48.29
B12	744.16	1,241.6	44.35	81.23

the task 100% of the time. The 3D scenario where only yaw and pitch are allowed (controllers A3, A4, A9, and A10) also proved to be not particularly challenging for the evolutionary process. The average success rate for the entire population slightly decreases but the best controllers still can, among all the cases, perform at least 80% of tasks with success. Things get more complicated when using controllers A5, A6, A11, and A12 (*i.e.* adding roll among the possible rotations available to the aircraft) and the results are unclear. In the A Scenario the discretisation of the input information seems to have a clearly positive impact on the performances of the controllers. For Scenario B we find instead evidence of both positive and negative impacts. Curiously, some of the controllers (*e.g.* A10) have evolved with more accurate behaviours for Scenario B ($T_s = \frac{M_s}{3}$) than for Scenario A.

The evolution in Scenario C, due to the requirement for the evolution of more complex behaviours, has been run for: 10,000 generations for architectures A1, A2, A7, and A8; 20,000 generations for architectures A3, A4, A9, and A10; 40,000 generations for architectures A5, A6, A11, and A12. The results from this setup are presented in Table 6.17. In this case the table contains an extra column indicating the percentage of tests concluded half-successfully, *i.e.* with at least one MAV having properly reached the target, but not two or more MAVs having done the same in a

coordinated fashion.

Table 6.17: Simulations C (non-incremental setup): resume of the new main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max fitness	Av. succ. rate [%]	Max succ. rate [%]	Approach rate [%]
C1	1,106.8	1,701	46.72	88.14	36.69
C2	1,011.4	1,505.3	31.64	58.18	43.16
C3	709.269	1,104.2	6.48	28.34	59.14
C4	683.148	948.38	0.37	6.36	76.36
C5	645.664	1015.2	5.33	23.9	57.94
C6	582.288	822.92	0.2	38.6	51.28
C7	1,008.4	1628.3	42.66	84.25	29.26
C8	1,228.2	1834.6	53.91	91.84	17.54
C9	743.587	1131.5	7.47	28.44	62.01
C10	752.741	1153.4	6.62	27.78	64.7
C11	562.598	917.09	3.65	18.55	47.13
C12	764.835	1123.9	5.8	23.16	67.7

As before, these results highlight worse performances scored by the controllers than those obtained within Scenarios A and B. The new task is indeed more complicated, since on top of the basic navigation behaviour the aircraft are now also required to coordinate amongst themselves. The difficulty is testified by the data relative to the maximum success rate obtained by the controllers in C that score good performances only for architectures A1, A2, A7, and A8, *i.e.* those in which the limited rotations available to the MAVs make the task relatively easier.

6.5.2 The incremental approach

The way in which an incremental approach has been introduced in this work is particularly straightforward. Incrementally evolving from Scenario A to B the connection weights and biases of the individuals belonging to the last generation of the best population evolved in A are loaded from the memory and used as starting point for the new evolutionary process. Things are slightly more complicated moving from Scenario A to C, because of the different network topologies used in the latter. In this case, the connection weights coming from the two extra input neurons are added to the A controllers and their associated values are set to 0 at the beginning of the

second stage of the evolutionary process (as suggested by Tomko & Harvey [368]).

Results of incremental evolution from scenario A to B

Starting from populations of individuals already evolved within the A scenario, the results demonstrate how $5,000$ generations of further evolution are enough to allow these individuals to generalise their target reaching abilities to moving targets as well. Therefore allowing performances comparable to those obtained with direct evolution in the B scenario.

The only situations in which this effect does not take place are when controllers A5 and A6 are used (furthermore, for these two architecture the performance of the controller dramatically decreases). Controller A10 is also affected due to the fact that a target moving at half the speed of the MAVs, generates worst results when incrementally evolved than when direct evolution is employed.

The absolute results obtained are summarised in Tables 6.18 and 6.19 for T_s of $\frac{M_s}{2}$ and $\frac{M_s}{3}$ respectively. Tables 6.21 and 6.22 show the comparison between the incremental and the non-incremental results.

Table 6.18: Simulations B (incremental setup from A to B, $T_s = \frac{M_s}{2}$): resume of the main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
A-B 1	955.64	1,428.1	82	100
A-B 2	1,002.6	1,434.1	81	100
A-B 3	864.54	1,378.6	54.77	98.5
A-B 4	783.57	1,321	52.64	96.67
A-B 5	602.36	1,110.4	13.58	49.75
A-B 6	579.55	970.23	1.16	14.85
A-B 7	1,007.4	1,435.3	82.28	100
A-B 8	999.05	1,434	81.64	100
A-B 9	877.67	1.393.3	63.06	99.6
A-B 10	688.51	1,149.1	23.59	53.17
A-B 11	554.38	1,167.4	30.26	81.08
A-B 12	772.39	1,298.9	45.16	94.72

Table 6.19: Simulations B (incremental setup from A to B, $T_s = \frac{M_s}{3}$): resume of the main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
A-B 1	952.86	1,408.7	75.46	99.65
A-B 2	994.81	1,432.1	81.83	100
A-B 3	928.29	1,413.3	62.71	99.85
A-B 4	851.27	1,359.3	57.77	95.7
A-B 5	624.53	1,143.5	19.71	58.88
A-B 6	597.66	964.96	0.53	8.67
A-B 7	978.43	1,424.5	79.31	100
A-B 8	995.91	1,432.1	82.22	100
A-B 9	887.86	1,400.9	64.54	100
A-B 10	741.83	1,321.9	44.66	94.05
A-B 11	611.16	1,237.6	38.62	91.03
A-B 12	796.49	1,333.7	50.18	96.42

Results of incremental evolution from scenario A to C

In this case the incremental evolutionary process has lasted for 10,000 generations, twice the duration of the one used for the B scenario, due to the more sophisticated neural architectures used. The absolute results are summarised in Table 6.20, while Table 6.23 shows the comparison between the incremental and the non-incremental results. At a first glance it is possible to see how the controllers evolved via incremental evolution do not score impressive results.

Table 6.20: Simulations C (incremental setup from A to C): resume of the main results (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max fitness	Av. succ. rate [%]	Max succ. rate [%]	Approach rate [%]
1	1,139.8	1725.5	48.93	90.01	34.01
2	1,347.2	1904.7	62.57	99.03	20.62
3	692.94	1046.9	4.77	22.99	68.16
4	512.01	755.6	0.23	4.28	66.39
5	475.42	742.07	0.58	7.11	47.88
6	545.11	929.72	2.89	18.73	30.52
7	1,196.9	1,776.6	52.88	92.89	30.58
8	1,386.5	1,921.5	65.43	99.62	17.82
9	699.83	1,076.4	7.31	26.93	60.6
10	710.44	1,058.3	4.09	18.96	67.22
11	400.55	654.01	0.84	6.46	35.19
12	568.79	910.57	3.09	14.59	53.39

Analysis of the results for incremental evolution

Regarding incremental evolution from A to B, the critical variable for the success of the incremental approach seems to be the complexity of the neural architecture used. For simple networks, as controllers A1-A6 are (feed-forward NNs without hidden layers), the effect seems to be limited. More complex architectures benefited much more from the incremental process instead. This phenomenon is evident comparing the results scored by architectures A5 and A6 against controllers A11 and A12. The average success rate of controller A5 dropped by 52.33% and 41.23% for $T_s = \frac{M_s}{2}$ and $T_s = \frac{M_s}{3}$ respectively, while architecture A6 scored -92.74% and -97.35%. On the other end, the performance of controller A11 increased by 67.46% and 97.54%; architecture A12 improved as well, scoring +33.45% and +13.15%. The simplest architectures - A1 and A2, as well as A7 and A8 - do not show any significant difference in the results obtained following the two alternative approaches, presumably because the ability to perform the task (which, for these controllers, is essentially 2D navigation) is easily learnt and direct evolution has already found close to optimal solutions. Things are more interesting and varied for the archi-

tectures of intermediate complexity, such as A3, A4, and A9[12]. Architectures A4 and A9 improved their performances among all the parameters measured, for both $T_s = \frac{M_s}{2}$ and $T_s = \frac{M_s}{3}$. For architecture A3 the fitness values scored (both average and maximum) are fairly similar among direct and incremental evolution, but the average and maximum success rates decreased.

Table 6.21: Comparison between incremental (A to B) and non-incremental (B) evolution ($T_s = \frac{M_s}{2}$) (average of the last 10 generations, based on 10 evolutionary runs)

Arch.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
1	−2.11%	+0.50%	+1.65%	0%
2	+2.62%	+0.99%	+4.82%	0%
3	+1.07%	−0.58%	−8.44%	−1.29%
4	+5.29%	+5.51%	+29.15%	+17.93%
5	−6.70%	−4.14%	−52.33%	−16.65%
6	+5.06%	−2.66%	−92.74%	−62.50%
7	+2.40%	+0.96%	+3.97%	0%
8	+7.03%	+0.25%	+0.29%	0%
9	+16.04%	+5.26%	+25.24%	+3.30%
10	−19.16%	−14.02%	−56.12%	−39.05%
11	−13.80%	+3.59%	+67.46%	+39.10%
12	+10.34%	+11.74%	+33.45%	+43.32%

Opposite results have been obtained by the incremental evolution from A to C, *i.e.* the further evolution of an architecture specialised in basic navigation to perform a cooperative task. In this case, the only controllers that have gained an advantage from the second evolutionary process have been the simplest ones: A1, A2, A7, and A8. All of the other's architectures (with the only exception of controller A6, for which direct evolution did not succeed thus making impossible an incremental evolution in the proper sense) have seen their performances dropping consistently, both in terms of average and maximum fitness, as for what concerns the success rate.

[12]For the purposes of this analysis, we do not take into account controller A10, for which incremental evolution has not been able to generate a proper behaviour.

Table 6.22: Comparison between incremental (A to B) and non-incremental (B) evolution ($T_s = \frac{M_s}{3}$) (average of the last 10 generations, based on 10 evolutionary runs)

Arch.	Av. fitness	Max. fitness	Percentage of tests concluded successfully	Max. success rate [%]
1	−2.75%	−0.99%	−5.06%	−0.35%
2	+0.33%	+0.41%	+0.07%	0.00%
3	+4.40%	+0.40%	−9.35%	−0.15%
4	+8.04%	+8.03%	+31.74%	+19.65%
5	−7.23%	−5.71%	−41.23%	−17.50%
6	+0.24%	−9.27%	−97.35%	−83.50%
7	−1.70%	−0.27%	−1.86%	0%
8	+0.10%	+0.05%	+0.85%	0%
9	+10.02%	+3.44%	+17.41%	+2.20%
10	−22.01%	−6.71%	−38.62%	−5.95%
11	+5.79%	+13.48%	+97.54%	+88.51%
12	+7.03%	+7.42%	+13.15%	+18.70%

Table 6.23: Comparison between incremental (A to C) and non-incremental (C) evolution (average of the last 10 generations, based on 10 evolutionary runs)

Sim.	Av. fitness	Max fitness	Av. succ. rate [%]	Max succ. rate [%]	Approach rate [%]
1	+2.98%	+1.44%	+4.73%	2.12%	−7.30%
2	+33.20%	+26.53%	+97.76%	+70.21%	−52.22%
3	−2.30%	−5.19%	−26.39%	−18.88%	+15.25%
4	−25.05%	−20.33%	−37.84%	−32.70%	−13.06%
5	−26.37%	−26.90%	−89.12%	−70.25%	−17.36%
6	−6.39%	+12.98%	+1345%	+385.23%	−40.48%
7	+18.69%	+9.11%	+23.96%	+10.26%	+4.51%
8	+12.89%	+4.74%	+21.37%	+8.47%	+1.60%
9	−5.88%	−4.87%	−2.14%	−5.31%	−2.27%
10	−5.62%	−8.25%	−38.22%	−31.75%	+3.89%
11	−28.80%	−28.69%	−76.99%	−65.18%	−25.33%
12	−25.63%	−18.98%	−46.72%	−37.00%	−21.14%

6.6 Conclusions

The main body of this chapter has presented the results obtained by the experimental setups already described in chapter 5 (with the notable exception consisting in no obstacles being deployed into the environment), applied to the new 3D simulator developed. The goal of these experiments was to evaluate whether autonomous neural network controllers, designed according to Evolutionary Robotics principle, could properly drive MAVs capable of manoeuvring over three dimensions (yaw, pitch, and roll) and dealing with tasks of various complexity. In this conclusive section we want to interpret the results obtained in the light of the "success threshold" introduced in chapter 4.

The first set of simulations was aimed to identify the best topologies for basic navigation, also to be used in the following experimental setups. Several architectures exceeded the 90% success rate for the best individuals driven by those, specifically topologies 1, 2, 3, 5, 7, 8, 9, 10, 13, 14, 19, and 20. The first thing that comes to the eyes is that most of these architectures (7 out of 11) do not embed any memory structure (either implemented as Jordan or Elmann network components). Secondly, at a closer look we can see that over 70% of these architectures (1, 2, 7, 8, 13, 14, 19, and 20) are those dealing with the simplest task possible, which is navigation relying on a single degree of freedom (yaw). The scenario they implement is not dissimilar to the one extensively discussed in the previous chapter, as it is essentially 2D navigation. Therefore the good performances are not surprising by any means. Things are more interesting for the remaining architectures (3, 5, 9, and 10). Topologies 3, 9, and 10 all deal with MAVs having two degrees of freedom (yaw and pitch) available and manage to make the aircraft able to carry out the tasks required. As expected, the main discriminant factor consists in the third rotation axes, *i.e.* roll, which makes the control task much more complex for the controllers (as we have previously discussed, the current roll angle of a MAV affects the outcome of every other further rotation it performs, either yaw or pitch, thus making difficult for the controller to figure out the correct series of manoeuvres to produce in order to reach a certain position in the space). The most interesting

results are those generated by architecture 5 (no hidden layer nor memory structures included, discrete encoding of the input information), the only one to exceed the 90% success threshold for the most complex scenario possible (yaw, pitch, and roll DoFs available). The success of this topology demonstrates how a simple neural network controller can successfully control a MAV for navigation tasks.

The second set of experiments involves a target able to move away from the approaching MAV. The movement repository of the target is dependent upon the degrees of freedom available to the aircraft, thus making the chasing task much more difficult than the one described in chapter 5 when the MAVs can perform pitch and yaw manoeuvres as the target is able to do the same but in much more time/space compared to its hunters. The results presented in this chapter are related to simulations carried out using architectures 5 and 11. Topology 5 is the one that performed best in the basic experimental setup, while topology 11, despite falling short in terms of success rate in simulations A, has been chosen as it was the one performing the best amongst those having a hidden layer. It was thought that this additional "computational capability" could have turned useful when the MAVs had to face more complex scenarios. The results do not show this effect. Instead architecture 5 is the one that again performed the best. All in all the results obtained were slightly worse than those gathered with the 2D simulator, but definitely comparable. The best controllers based on topology 5 scored in the 3D model a 90%+ success rate for targets moving at one fifth and one fourth of the MAVs' speed. For targets moving at one third of the speed of their chasers, the performance stopped at 89%, still very close to our success threshold.

The third experimental setup is the one involving cooperation, with at least two MAVs that are asked to reach the target and to activate their Boolean output units in quick succession. Unfortunately in this scenario the results obtained have not been favourable. Out of the 12 controller topologies tested, only one (architecture 8, yaw-only) carried out the test with a success rate sufficient to exceed the 90% success threshold we decided to use. Things have improved quite clearly adopting incremental evolution (architectures 1, 2, 7, and 8, *i.e.* all of those relying on a single

DoF, performed with success rates higher than 90%), but not enough to qualitatively modify the meaning of the results obtained in this scenario.

6.6.1 Multi-threading

One of the simulations described in this chapter, namely A9, has been used as benchmark for further technical analysis on the multi-threading topic. This simulation was run ten times for 50 generations on different machines, measuring the amount of time required to complete the evolutionary process whether multi-threading was enabled or not. As expected the results highlighted a massive improvement in computing performances when multi-threading was used. On the same machine, the comparison between the 3D simulator engine running as a single-thread application and with the threads of execution spread across the four cores available showed that in the latter scenario the time required for the computation decreased by 66.59%. Moving to a machine with eight processors available the boost in performance was even more evident, due to reduction in time measured as 80.66%.

What we have obtained is the practical demonstration that our simulator, but also most ER robotics in general as they tend to share the most important characteristics of the models described in this thesis, can benefit hugely from the adoption of multi-threading programming techniques. Also when these techniques are implemented in the most high-level way as possible.

6.6.2 Incremental evolution

Finally, an additional set of experiments with a prominent theoretical footprint has been carried out on the topic of incremental evolution. As it has been extensively discussed in Ruini *et al.* [324], it is difficult to draw, from a single experiment, definitive conclusions about the validity of a complex and widely applicable approach such as incremental evolution. Not only the particular task analysed, but also several additional variables seem to have had an impact on the profitable application of an incremental approach, as demonstrated by the study presented herein. The challenge consists in clearly identifying these variables and providing a theoretical framework

- a set of guidelines - that researchers willing to experiment in incremental evolution could rely on in the future in order to fully benefit from this approach.

Based on the results described in the latest sections of this chapter, we have identified at least two findings that warrant further study. First, it seems to be necessary to let the evolutionary algorithm free to explore a large space of solutions. Incremental evolution performed from Scenario A to B has been particularly beneficial for the more complex architectures (*i.e.* those characterised by a larger amount of connection weights), while generating limited results for the others. Second, the incremental process has to go through a series of closely linked steps. This has been demonstrated by the fact that direct evolution, in the testbed scenario discussed herein, has clearly outperformed the incremental approach for Scenario C. Further analyses are required to investigate in more detail these two aspects.

Chapter 7

Flocking Behaviour: Towards Experiments on Physical Robots

This chapter describes the last set of experiments carried out as part of the PhD research. What is presented herein is an alternative approach to collective behaviour and distributed control based upon flocking principles. Rather than concentrating exclusively on computer simulations, as done for the work presented in chapters 5 and 6, the emphasis shifts towards experimenting with real robotic platforms. The testbed platform used as reference is the *swinglet*, a lightweight mono/fixed-wing robotic aircraft produced by the Swiss company senseFly™.

In this chapter we will first illustrate the most relevant technical details about the aerial platform used. Then we will provide an overview of the computer model utilised for designing and testing autonomous controllers for the *swinglet*. The algorithms developed on the software simulator implement both individual/collective navigation (using approximated areas of attraction or more accurate GPS-like waypoints), and flocking (speed adjustment, heading alignment, Reynolds' boids-like) functionalities. The results of preliminary experiments carried out on real robots are presented.

The work described in this chapter has been made possible thanks to a collaboration with the Laboratory of Intelligent Systems (LIS) at the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. The author would like to thank all those involved in this joint effort. At the same time it is important to make clear

the boundaries between what has been done by the author and what by third parties. The author has personally designed the software simulator described in paragraph 7.2 and sub-paragraphs, identified the metrics presented in paragraph 7.3.1, and carried out the experiments detailed in paragraph 7.3.2. The flocking algorithm introduced in the first section of paragraph 7.3.3 has been designed and tested on simulation by the author, while the remaining work has been carried out by collaborators at the EPFL (a detailed list of the people who actually did the job is reported in the second section of paragraph 7.3.3). The robotics platform used (described in paragraph 7.1) has been entirely designed by researchers working at the EPFL and at senseFly[1TM].

7.1 Robotics platform used: senseFly's swinglet

The experiments presented in this chapter have been carried out using a customised version of the *swinglet*[2] (see Figure 7.1), a 420g light 80cm wing-span mono/fixed-wing MAV produced by senseFly. The *swinglet* is often used for aerial photography/surveillance and scientific investigations on outdoor flying robots [211]. Its main structure is made of expanded polypropylene, on which a single electric propeller - fuelled by a polymer lithium battery guaranteeing up to 60 minutes of autonomy (enough to approximately cover a $40km$ distance) - is mounted. According to the specifications provided by the manufacturer, the *swinglet* flies at a speed of between 10 and $15m/sec$ inclusive, it has a maximum turn rate of $45°/sec$ guaranteed by the use of two elevons[3] (one on each side of the aircraft) and can proficiently cope with wind currents as strong as $25km/h$. Additional details on the *swinglet* can be found in appendix A.8.

The configuration we have had access to comprises of a GPS receiver (a u-blox[TM]LEA-5H GPS module[4]), a rate gyroscope (Analog Devices[TM]ADXRS610[5]) used to measure the absolute yaw rate of the MAV, and two pressure sensors (im-

[1]senseFly[TM](`http://www.sensefly.com`) is an EPFL spin-off company.
[2]`http://www.sensefly.com/products/swinglet/`
[3]Elevons are combined ailerons/elevators.
[4]`http://www.u-blox.com/en/lea-5h.html`
[5]`http://www.analog.com/static/imported-files/data_sheets/ADXRS610.pdf`

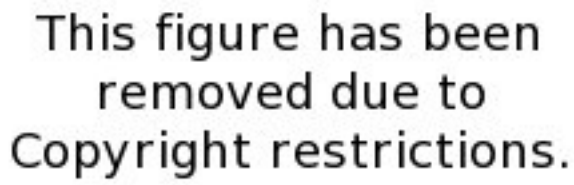


Figure 7.1: senseFly's *swinglet* MAV. Source: [157]

plemented as pitot tubes and belonging to the Freescale™ MPX series[6]). In order to exchange data with a ground-based station the *swinglet* is also equipped with a Digi™XBee-PRO PKG-U[7], which is a radio transmitter providing a 1.6km communication range. Standard R/C equipment is used for actuators, the motor controller and the battery.

A complete autopilot system has been built by Leven et al. [212] and installed on the aircraft through a dsPic33 micro-controller[8] (see Figure 7.2(a)). The approach followed by the designers of this system is different than those usually undertaken for implementing autopilots. Leven's technique can be considered "minimalist" since it only relies on two pressure sensors and a single axis rate gyro, rather than on a complete IMU (Inertial Measurement Unit[9]) or an AHRS (Attitude and Heading Reference Systems[10]) as is the common habit among the experts of the field[11]

[6] `http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=01126990368716`

[7] `http://ftp1.digi.com/support/documentation/90000831_A.pdf`

[8] `http://www.microchip.com/ParamChartSearch/chart.aspx?mid=14&lang=en&branchID=8183`

[9] *An IMU is an electronic device that measures and reports on a craft's velocity, orientation, and gravitational forces, using a combination of accelerometers and gyroscopes* (from: `http://en.wikipedia.org/wiki/Inertial_measurement_unit`).

[10] *An AHRS consists of sensors on three axes that provide heading, attitude and yaw information for aircraft. They are designed to replace traditional mechanical gyroscopic flight instruments and provide superior reliability and accuracy* (from: `http://en.wikipedia.org/wiki/Attitude_and_Heading_Reference_Systems`).

[11] The computer code constituting the controller can be downloaded from: `http://gna.org/projects/aeropic/`

(*e.g.* [189] and [191]).

The autopilot has direct reading access to all the sensors mounted on the aircraft and it can control both the propeller thrust and the servomotors that in turn lower/raise the elevons. In addition to flight stabilisation the system can therefore perform control of airspeed, altitude and heading turn rate. Some basic autonomous navigation functions, such as waypoint-based navigation, are already implemented within the system. Furthermore the autopilot provides an automatic landing function based on GPS, which - either on request or in case of a software/hardware failure - forces the MAV to fly towards a pre-specified landing spot, then makes it glide around it progressively reducing its altitude and the thrust of the propeller until the aircraft reaches the ground.

A flexible payload bay situated on the top surface of the aircraft - next to the battery compartment - allows the *swinglet* to transport up to $150g$ of additional equipment. In the available configuration the payload consists of both the above mentioned autopilot system, and a Toradex^{TM}Colibri PXA270 CPU board[12] running a minimal Linux distribution. The board has an "off-the-shelf" USB Wi-Fi dongle (a dual band Wireless-N Netgear^{TM}WNDA3100[13]) connected to it, which can be used for communication between the MAVs. The dongle has a $500m$ line-of-sight communication range, although its firmware has been modified in order to allow the experimenter to restrict this range as desired [158]. The onboard computer is directly connected to the autopilot (see Figure 7.2(b)), to which it can issue commands (namely desired turn rate, speed and altitude) via a software controller running on the CPU board.

With the use of the XBee radio link, the MAV behaviour can be monitored by a standard computer located on the ground running a dedicated software called *e-motion* (see a screenshot in Figure 7.3) developed by Beyeler *et al.* [38]. The computer simply needs to be connected to a proper XBee^{TM}device[14] capable of exchanging data with the radio unit installed on the aircraft in order to be used as ground station. *e-mo-tion* allows the user to switch the control of the aircraft between

[12] `http://www.toradex.com/Products/Colibri/Modules/Colibri_PXA270_312MHz`
[13] `http://www.netgear.co.uk/wnda3100.php`
[14] Each XBee^{TM}unit used on the ground-based station can connect up to a maximum of 3 MAVs.

This figure has been removed due to Copyright restrictions.

This figure has been removed due to Copyright restrictions.

(a) (b)

Figure 7.2: (a) the dsPic33 micro-controller upon which the autopilot has been built; (b) overall view of all the equipment hosted inside the payload bay. Sources: [212, 213]

the auto-pilot[15], the software controller running on the Linux board and a standard radio transmitter (in case of necessity the *swinglet* can be remotely controlled, which can be an extremely useful property to rely on during testing). Another interesting feature offered by the software running on the ground station is the possibility of logging all the flight data, thus relieving the controller operating on the Linux board from this task. In this way the code running on the onboard computer will be lighter (also potentially less bug-prone) and the user will not incur in the risk of filling the flash memory on the embedded computer, thus reducing the possibility of unexpected crashes.

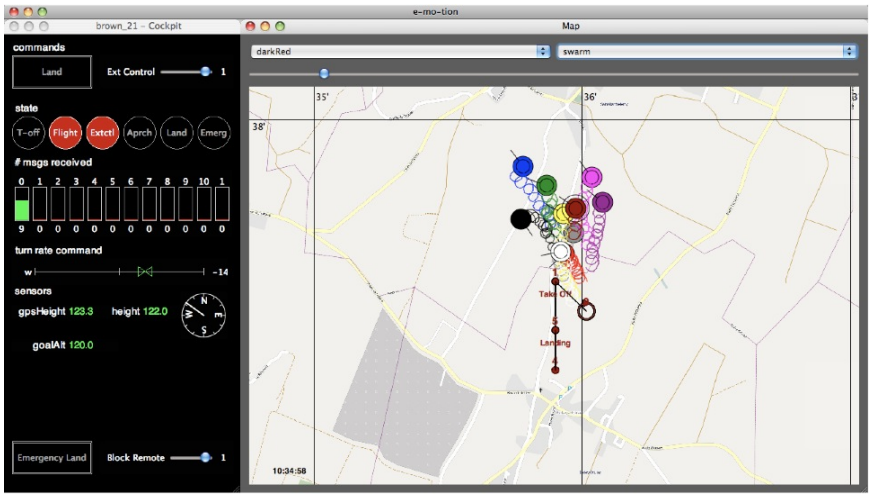


Figure 7.3: Screenshot of the *e-mo-tion* main interface during a flight test involving several MAVs

The *swinglet* can be seen in action by downloading the movies published on the

[15]Through *e-mo-tion* it is also possible to interact with the autopilot, for example deploying specific waypoints that the MAV will have to follow, or forcing a landing procedure.

EPFL's Laboratory of Intelligent Systems website[16].

7.2 Software simulator

The software simulator used for the preliminary testing of the controllers described in the current section (see Figure 7.4) is a modified version of the one discussed in Ruini *et al.* [323, 324]. As before, the simulator implements an incremental geometric flight model in discrete time steps [311]. The parameters of the model have been tuned in order to replicate, in the most accurate way possible, the constraints of the senseFly^{TM} *swinglet* platform, specifically in terms of speed range and turn rate per second.

The autonomous controllers managing the aircraft operate on two variables: airspeed (which can be increased or decreased) and turn rate (which can be modified instructing the MAV to perform a yaw turn, *i.e.* a rotation around its vertical body axe). The reason for reducing the controller to operate on these two dimensions relates to the autopilot system described within the previous section. As noted before, the autopilot provides both flight stabilisation and altitude control (other than being able to modify speed and turn rate), meaning that the autonomous controller can assume the MAV is always parallel to the ground and flying at the desired altitude, thus ignoring aspects such as current pitch and roll angles. In this way the MAVs can be considered to some extent a type of "2D flying robot", since their behaviour will only depend on rotations around one single axis, as is generally the case for ground-based vehicles.

The virtual reference environment implemented in this version of the simulator consists of a three-dimensional parallelepiped measuring 800x300x800GU (where the height is represented as the X axis; see Figure 7.5, which also highlights the coordinate systems used by Irrlicht and therefore in the rest of this chapter[17]).

[16] `http://lis.epfl.ch/smavs/`

[17] Within this context we use the term GU, "Graphical Unit", referring to the basic measure unit employed by Irrlicht graphics engine. We have endeavoured to obtain the following two relationships: 1GU≈1m, 1 time step≈0.1sec. In an adjustment to the parameters of the previous simulators, in this version the environment boundaries can be stepped over with no consequences for the MAVs.

Figure 7.4: Screenshot of the flocking software simulator

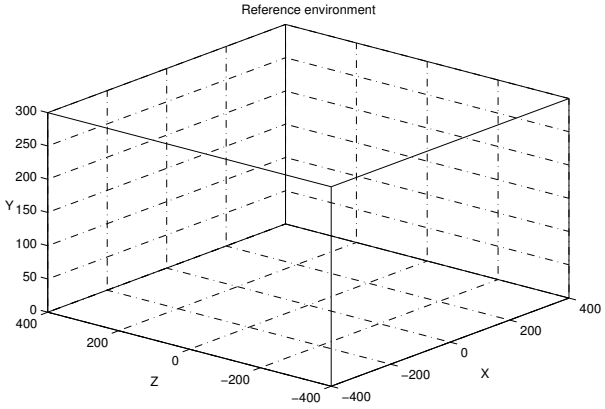


Figure 7.5: The simulation reference environment

The MAVs - having a size of 4.35x1.797x4.82GU - can fly across the environment at a speed included between 10 and $15 GU/timestep$. At any time step the MAVs first perform a turn rate (if so decided by the controller, in which case the rotation must be included within a $[-4.5°; 4.5°]$ range) then they are all moved in sequence.

Each aircraft moves along its current heading (after the yaw preliminary rotation) redeploying itself at a distance calculated according to Equation 7.1 (where i is used as a general index for indicating a non-specific MAV).

$$distance = \frac{MAV_i.speed + X \sim (0, 0.25)}{10} GU \tag{7.1}$$

The new coordinates are calculated as in Equation 7.2 (please consider that the

addition and the multiplication operations have to be interpreted as vector addition and scalar multiplication respectively).

$$\vec{MAV}_i^{t+1} = \vec{MAV}_i^{t} + distance * \vec{transformationVector} \tag{7.2}$$

transformationVector is a three-dimensional vector, for which the X, Y, and Z elements are defined as specified in Equation 6.3[18]:

$$\begin{aligned} X &= cos(MAV_i.\hat{x}) * sin(MAV_i.\hat{y}) * cos(MAV_i.\hat{z}) + sin(MAV_i.\hat{x}) * sin(MAV_i.\hat{z}) \\ Y &= cos(MAV_i.\hat{x}) * sin(MAV_i.\hat{y}) * sin(MAV_i.\hat{z}) - sin(MAV_i.\hat{x}) * cos(MAV_i.\hat{z}) \\ Z &= cos(MAV_i.\hat{x}) * cos(MAV_i.\hat{y}) \end{aligned} \tag{7.3}$$

A certain amount of noise - distributed according to a Gaussian distribution with mean 0 and standard deviation 0.25 (which in Equation 7.1, as well as in the next paragraphs, is defined as $X \sim (0, 0.25)$) - is added to both any yaw manoeuvre and forward movement performed by the MAVs simulated through this computer model. The reason for introducing noise consists in adding some degrees of uncertainty to the simulated flight dynamics, thus increasing the realism of the model and the robustness of the controllers tested in it [192].

The simulator allows the user to set several parameters before running an experiment[19]:

- *number of MAVs*: from a minimum of 1 up to a maximum of 12;
- *initial MAVs team formation*: horizontally aligned (2D), queued (2D), V-formation (2D), intervallic launch (2D), random (2D), horizontally aligned (3D), queued (3D), V-formation (3D), intervallic launch (3D), random (3D);
- *navigation task*: none, fly around the centre of the environment (leader only),

[18] These operations are performed within the software by the dedicated Irrlicht functions *setRotationDegrees()*, *setRotationRadians()*, and *transformVect()*.

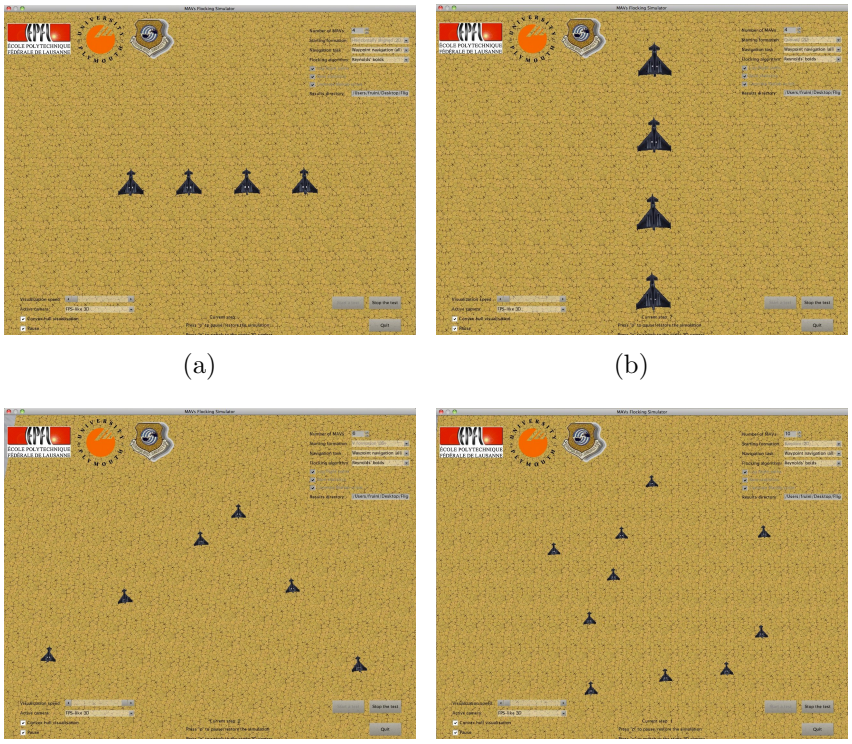
[19] Some of these parameters, such as the navigation task and the flocking algorithm used, can also be modified in real time while a simulation is running.

fly around the centre of the environment (entire team), waypoint navigation (leader only), waypoint navigation (entire team), follow the leader (who flies around the centre of the environment), follow the leader (who flies between two waypoints);

- *flocking algorithm*: none, speed adjustment, heading alignment (to the leader's heading), heading alignment (to the average neighbours' heading), heading alignment (to the leader's heading) + speed adjustment, heading alignment (to the average neighbours' heading) + speed adjustment, Reynolds' boids.

7.2.1 Initial formation

The initial formation parameter allows us to select the way in which the MAVs will start each test. The two main categories the user can choose from are 2D and 3D. The former means that the aircraft will all be flying at the same altitude, while the latter deploys the MAVs at different altitudes.



(a) (b)

(c) (d)

Figure 7.6: Different initial MAV teams formations: (a) 4 MAVs horizontally aligned; (b) 4 MAVs queued; (c) 6 MAVs reproducing a V-formation; (d) 10 MAVs randomly distributed

When horizontal alignment or queueing are selected (respectively shown in Figure 7.6(a) and 7.6(b)), the MAVs are respectively deployed side by side or forming a queue, standing in both cases at 10GU (d) distance far from each other[20]. V-formation and random deployment are implemented according to Algorithms 3 and 4, where: h indicates the desired altitude; N stands for the number of MAVs within the team; *areMAVsTooClose()* is a function that checks whether in the group there are two or more MAVs too close to each other, *i.e.* within a distance $< d$ between them; *randFloat()* is a function returning an uniformly distributed random float value included between the lower and upper boundaries specified in input[21].

Algorithm 3 Flocking simulator: V-formation MAVs deployment (3D)

```
d = 10;
MAV_1.x = 0; MAV_1.y = h; MAV_1.z = 0;
for i=2:N do
    MAV_i.y = h + randFloat(−5.0, 5.0);
    x = d ∗ i; z = d ∗ i;
    if i%2 = 0 then
        MAV_i.x = MAV_1.x − x;
        MAV_i.z = MAV_1.z − z;
    else
        MAV_i.x = MAV_1.x + x;
        MAV_i.z = MAV_1.z + z;
    end if
end for
```

Algorithm 4 Flocking simulator: random MAVs deployment (3D)

```
d = 10; d_2 = N ÷ 2.5;
while areMAVsTooClose() do
    for i=1:N do
        MAV_i.x = randFloat(−d ∗ d_2, d ∗ d_2);
        MAV_i.y = h + randFloat(−5.0, 5.0);
        MAV_i.z = randFloat(−d ∗ d_2, d ∗ d_2);
    end for
end while
```

The two methods defined as "intervallic" launches indicate that the MAVs are sequentially deployed into the reference environment according to their ID[22]. In the

[20] In case of a 3D initial formation the distances are calculated considering the MAVs are all at the same altitude.

[21] Algorithms 3 and 4 refer to the 3D scenarios. The altitude (y axis) is simply set equal to h in case a 2D deployment method is selected.

[22] When the MAV objects are generated by the software, each of them receives an ID number starting from 1 and increasing sequentially. The upper boundary of the IDs range is N, which corresponds to the number of MAVs used.

2D scenario the aircraft just appear at the centre of the environment with a certain time delay between each of them. Instead - in the 3D scenario - the MAVs take off in series from the ground, then follow a fixed path which brings them to the desired altitude through progressive modifications of their pitch rate (see Figure 7.7). In both cases, the interval between each "launch" amounts to 500 time steps.

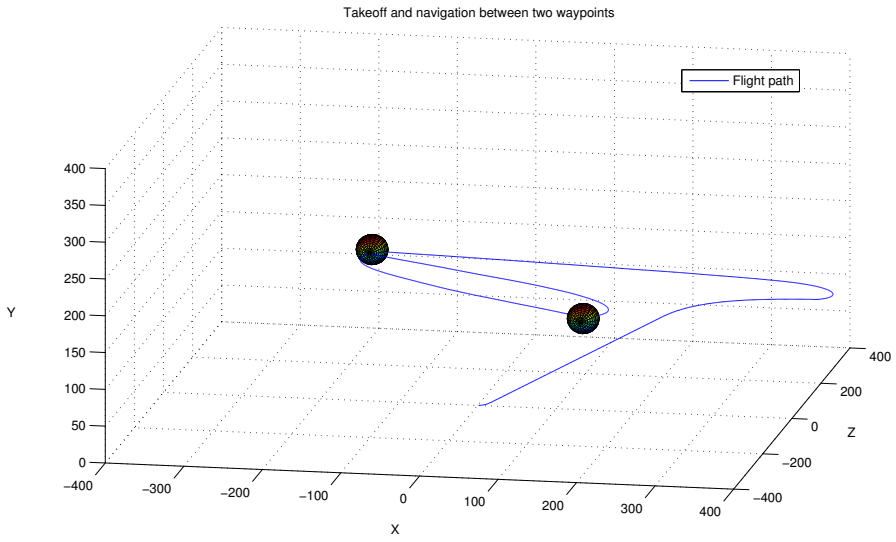


Figure 7.7: [
Flight path followed by a single MAV taking off from the ground]Flight path followed by a single MAV taking off from the ground and then navigating between two waypoints

7.2.2 Navigation algorithms

Concerning navigation, the MAVs - as mentioned before - can be driven by three different categories of algorithms: *a)* fly being attracted to the centre of the environment, *b)* navigate back and forth between different (fixed) waypoints, or *c)* follow a "leader" teammate.

To implement the attraction towards the centre of the environment the simple formula expressed in Equation 7.4 - which returns a steering request - has been used. The numerator of this formula computes the distance from the centre of the environment for the i-th MAV (in two-dimensions only, which is the reason for omitting the $MAV_i.y^2$ term). The denominator simply provides to calculate the diagonal length for the base of the reference environment, then divides the obtained value by 2. Gaussian noise is added to the resulting steering request.

$$steeringRequest = \frac{\sqrt{MAV_i.x^2 + MAV_i.z^2}}{(\sqrt{800^2 + 800^2})/2} + X \sim (0, 0.25) \tag{7.4}$$

Figure 7.8 shows, in two-dimensions, the flight paths that 4 aircraft have followed during a simulation they started deployed according to a random 2D formation. The MAVs have flown for about 10,000 time steps while being attracted towards the centre of the environment.

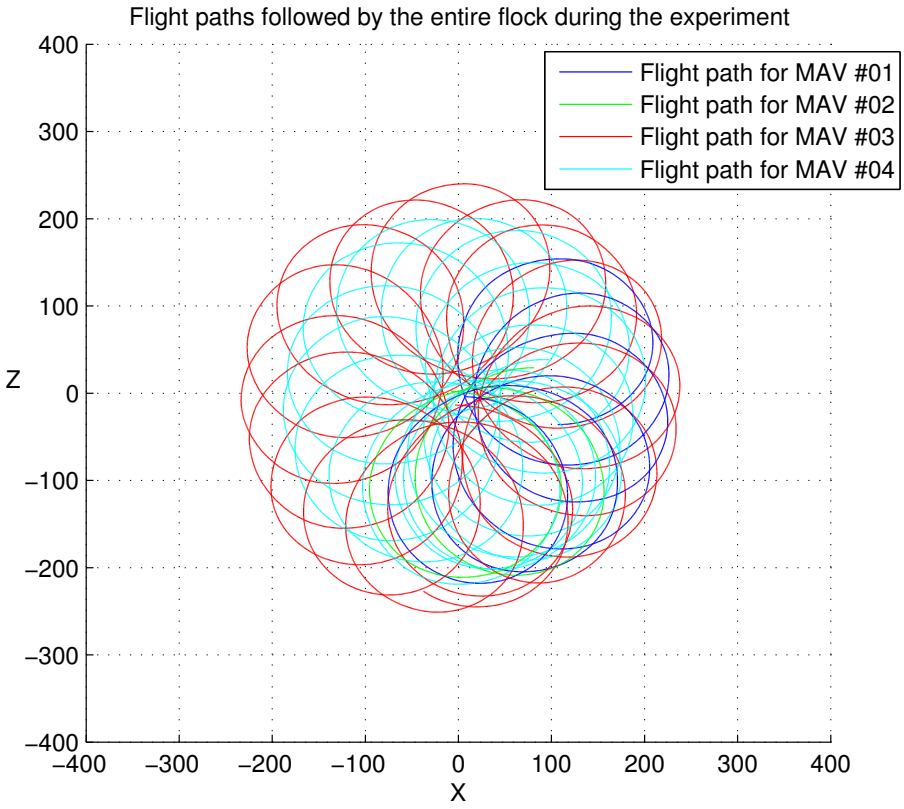


Figure 7.8: Flight paths followed by a team of four MAVs attracted to the centre of the reference environment. Their trajectories describe a series of circles - moving counterclockwise - passing through the central point

Waypoint navigation simply consists in the MAV flying between two fixed points in the space. The two waypoints used are respectively located at coordinates $(-165.0, h, 165.0)$ and $(165.0, h, -165.0)$, thus 300GU far from each other (see Equation D.1 for a reference on how the distance has been calculated). h, within this context, represents the altitude of the all MAVs or the altitude of the MAV with the lowest ID for 2D and 3D starting formations respectively. The steering request generated by the controller at any time step is calculated according to Equation 7.5.

In this equation the $\Delta\alpha$ symbol indicates the angle between the current waypoint and the heading of the MAV (assuming the aircraft being parallel to the ground, i.e. with a roll/bank angle equal to 0°); its value falls within the $[-180.0°, 180.0°]$ range ($[-180.0°, 0.0°]$ when on the relative "left" of the MAV, $[0.0°, 180.0°]$ when on its relative "right"). Again, normally distributed noise is added to the value generated by the equation to allow for some uncertainty in the outcome of the executed manoeuvre. A waypoint is considered reached when one of the MAVs gets closer than 30GU to it. The two waypoints, as well as the trajectory followed during a test by a MAV navigating between them (inclusive of the take off phase), can be seen in Figure 7.7.

$$steeringRequest = \frac{\Delta\alpha}{40} + X \sim (0, 0.25) \tag{7.5}$$

The simulator allows all of the MAVs to fly between waypoints/around the centre of the environment, or just one of them (the "leader") conforming with such a navigation task. If the latter feature has been activated, the non-leader MAVs will have (depending on the choice made by the user) two options available: not doing anything (*i.e.* flying in a straight line), or fly following the leader. When an algorithm (whether a navigation or, as we will see in the next section, a flocking one) involving a leader is selected, the one designated to assume that role is the MAV with the lowest ID.

Leader-following behaviour and waypoint navigation have been implemented in a very similar way. When one of the two navigation algorithms belonging to the leader-following category is selected, the leader either navigates attracted to the centre of the environment or flies back and forth between the two fixed waypoints. The followers are driven by steering requests generated through Equation 7.5, with the only difference consisting in the calculation of the $\Delta\alpha$ parameter which now represents the angle between the follower's heading and the position of the leader, rather than the angular difference between the heading direction of the follower and the current waypoint.

Two more things to consider in relation to this are:

- for 3D intervallic launches the navigation algorithm only activates once the MAV has reached the desired altitude;

- a common factor across all the navigation algorithms is the fact that the MAVs start every simulation flying at a speed equal to $12 GU/timestep$.

7.2.3 Flocking algorithms

The software simulator will not permit the assignation of any navigation task to the aircraft. In this case all the MAVs will simply follow a straight heading, unless a flocking algorithm, which from time to time forces them to steer, has been selected. Flocking algorithms are so labelled because they aim to make the MAV team behave like a flock. Various alternative strategies have been tested to obtain this outcome and a description of how they work is provided within the current section.

From a technical point of view, the flocking algorithms that generate steering requests[23] make the MAV perform, at any time step, a yaw rotation which is the sum (intended as sum of circular quantities) of two independent steering requests: one coming from the navigation algorithm (if enabled), the other one from the flocking rule.

The first option available to the MAVs is speed adjustment. According to this algorithm, which only works when waypoint navigation is used[24], the designated leader continuously broadcasts information about its coordinates and those of the waypoint it is currently aiming at. The other MAVs receive this information in real time[25] and interpret it according to Algorithm 5, where: *distance()* is a function which returns the distance between the two points specified as input parameters, *flockingDistance* is the desired distance at which the followers should keep from the leader, 0 is the ID of the flock leader, and $waypoint_j$ is the waypoint towards which the leader and/or all the MAVs is/are currently flying. As the algorithm shows, a

[23] Only one of them, namely *Speed adjustment*, does not generate any steering request at all.

[24] This algorithm could potentially also work with "attraction to the centre" as navigation task, but this functionality has not been implemented in the simulator.

[25] It should be noted that within the simulator all the MAVs have instantaneous access to all the information they need. As we will see in next sections, this is not true for real robots since all the required information must be exchanged between the robots, thus leading to some delay in communication and to the necessity of staying within a limited distance range.

2GU tolerance has been added to the calculus of the desired flocking distance.

Algorithm 5 Flocking simulator: speed adjustment for the i-th non-leader MAV

```
flockingDistance = 10;
if distance(MAV_i, waypoint) > distance(MAV_0, waypoint) then
  if distance(MAV_i, MAV_0) > (flockingDistance + 2) then
    MAV_i.speed++;
  else if distance(MAV_i, MAV_0) < (flockingDistance − 2) then
    MAV_i.speed−−;
  end if
else
  MAV_i.speed−−;
end if
```

This simple code allows the MAVs to infer their relative position compared to the leader (*i.e.*, whether they are in front of or behind it, "understood" exploiting the knowledge about the current waypoint coordinates) and then adjust their speed accordingly (increasing it if behind the leader and farther than the desired flocking distance, decreasing it if in front of the leader or too close to it). The intensity of the speed adjustment corresponds to a random float value drawn from a flat distribution ranging between 0 and 0.5.

An additional flocking algorithm implemented in the simulator is heading alignment. As the name suggests, this algorithm is used to modify the heading of each MAV in order to match it with a reference point. The references can be either the leader's heading (as usual the "leader" is the MAV that has been assigned the lowest ID, independently from the navigation algorithm in use[26]) or the average heading for all the MAVs within the neighbourhood.

Once the $\Delta\alpha$ between the current heading and the desired one has been calculated, the amount of steering to perform is calculated according to Equation 7.5. In algorithmic terms the entire procedure can be represented by the pseudocode in Algorithm 6, where: *calculateDeltaHeadingFromTo()* is a function which returns the angle between the heading of the object received in input (first parameter), and a different object for whom the coordinates are received by the function as a second parameter; *calculateAverageNeighboursHeading()* is a function calculating the average neighbours' heading for the MAV specified in input.

[26]In the case of this flocking algorithm being selected it will not have any impact on the leader,

Algorithm 6 Flocking simulator: heading alignment for the i-th MAV

$\Delta\alpha = 0$;
if $flockingAlgorithm = alignToTheLeader$ **then**
  if $i \neq 0$ **then**
    $\Delta\alpha = calculateDeltaHeadingFromTo(MAV_i, MAV_0)$;
  end if
else if $flockingAlgorithm = alignToTheNeighboursHeading$ **then**
  $avgHeading = calculateAverageNeighboursHeading(MAV_i)$;
  $\Delta\alpha = calculateDeltaHeadingFromTo(MAV_i, avgHeading)$;
end if
$performYaw(MAV_i, \Delta\alpha \div 40 + X \sim (0, 0.25))$;

The simulator also allows the use of mixed flocking algorithms in which speed matching and heading alignment (*i.e.* Algorithm 5 and 6) are used together. Furthermore the user can decide to employ the Reynolds' flocking algorithm for boids. What this algorithm is and how it works will be explained in detail in the next two sections.

Please consider that all of the flocking algorithms also generate an output steering manoeuvre which is affected by Gaussian distributed noise.

Reynolds' flocking algorithm for boids

With the term "Reynolds' flocking algorithm" we refer to the core of the software Craig Reynolds originally designed to implement automatic flocking behaviour among computer animated agents (or "boids", *i.e.* bird-like objects, according to his definition). His work - originally used in order to help the workers in computer graphics involved in designing the motion of large groups of entities - eventually led in 1987 to a seminal publication introduced at the annual edition of the SIGGRAPH[27] conference [311]. The work presented by Reynolds received a great deal of attention quickly becoming extremely popular within the computer science field[28].

The remarkable facet of Reynolds' work consists of the assumption upon which

since it is not supposed to perform any steering manoeuvre in order to match its own heading.

[27]SIGGRAPH (International Conference and Exhibition on Computer Graphics and Interactive Techniques, `http://www.siggraph.org`) is a traditional conference dedicated to the computer graphics community that in 2011 will be held for the 38th time.

[28]Scientists from apparently unrelated fields, as complexity science, also looked with interest at Reynolds' model, seeing in it an excellent demonstration of a complex collective behaviour emerging from the low-level interactions of a multitude of agents, each of them being aware of (as well as influencing) a narrow neighbourhood only. This is a classical example of what Murray has defined as the *molecular view* of complexity [273].

his model is based. Rather than elaborating complex rules to govern the behaviour of a flock considered as a whole, he proposed an approach based on every single individual obeying to a limited set of simple rules. As well as simple these rules are also local, in the sense that every boid is only aware of its local neighbourhood (*i.e.* the behaviour exhibited by the boids closer to it than a certain threshold) and does not have access to global information about the entire flock at all. This mechanism has been proven to work. Large groups of boids driven by Reynolds' algorithm are capable of showing a coherent flocking behaviour, as well as higher level properties such as collective obstacle avoidance[29]. Further improvements that have been made over the years on top of the original algorithm allow for a flock to be directed (even if that would mean losing one of the characteristic traits of the model, which is the absence of global information[30]) to replicate leader-following dynamics [150, 312], etc.. Modified versions of the algorithm have also been applied countless time to reproduce things such as the motion of animals as in schooling [199] or herding [165].

It is worth noting that, at the time he wrote his software, Reynolds was not interested in replicating real animal behaviour in computer animations, so he never claimed that his model faithfully recreates conducts that can be observed in nature. Furthermore, another assumption he implicitly made about all the members of a flock being peers (*i.e.*, no hierarchical structures existing within the group) is not necessarily true (and it has recently been challenged by Biro and colleagues [274, 41, 125] in their studies focusing on homing pigeons).

The set of rules governing the flocking behaviour can be summarised (in order of decreasing importance) as:

1. *collision avoidance*: avoid collisions with nearby flockmates;

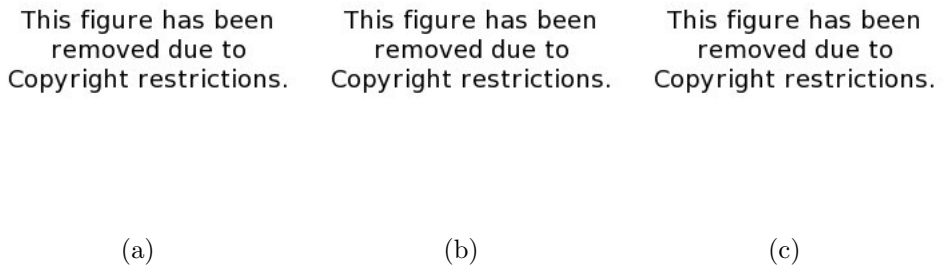
2. *velocity matching*: attempt to match velocity with nearby flockmates;

[29] An example of Reynolds-based flocking behaviour can be found in two videos made by the author: one demonstrating the basic behaviour (`http://www.youtube.com/watch?v=2aXMo3MFNsA`) and one in which the boids are attracted by a point moving across the space and at the same time need to avoid a fixed obstacle (`http://www.youtube.com/watch?v=GUkjC-69vaw`).

[30] Although it might be argued that not all of the boids need to be aware of this information. The modified algorithm would preserve Reynolds' assumption if only the boids in front of the group would be able to perceive the point of attraction and steer towards it, thus back-propagating a steering manoeuvre to the entire flock

3. *flock centering*: attempt to stay close to nearby flockmates.

To fully understand how these three rules work, it is fundamental to remember that Reynolds defined "velocity" as "*a vector quantity, referring to the combination of heading and speed.*" Rule number 2, velocity matching, therefore refers to both speed adjustment (match the speed of the boids inside the neighbourhood) and heading alignment. To clear up potential misunderstandings, in a later paper [312] Reynolds renamed the velocity matching rule as *alignment*. Collision avoidance and flock centering are instead two complementary rules that respectively provide to keep the boids at a "safe distance" between each other, but not too far away (that is: close enough to be considered a homogeneous group to the eyes of an external observer).



(a) (b) (c)

Figure 7.9: The three flocking rules elaborated by Reynolds: (a) separation; (b) alignment/velocity matching; (c) cohesion. Source: `http://www.red3d.com/cwr/boids/`

Every rule generates an independent request for a steering manoeuvre to be executed by the boid under examination. The entire model relies on vector geometry, implying that the steering requests generated by the different rules are expressed in terms of independent geometric vectors. But, considering that a boid can only perform one steering manoeuvre at a time, an issue is raised regarding how the boid is supposed to behave when facing rules that attempt to steer it towards opposite directions. To solve this potential source of troubles, Reynolds first assigned different weights to the three rules, thus attributing a different relative importance to each of them. Not only that the vectors generated by the different rules are attenu-

ated by a certain factor[31], but a "governing element", named *accumulator*, was also introduced. The working principle of the accumulator is fairly easy to understand. As Reynolds explained [311]:

> "*The acceleration [steering] requests are considered in priority order and added into an accumulator. The magnitude of each request is measured and added into another accumulator. This process continues until the sum of the accumulated magnitudes gets larger than the maximum acceleration [steering] value, which is a parameter of each boid. The last acceleration request is trimmed back to compensate for the excess of accumulated magnitude.*"

Despite the entire procedure appearing to be quite straightforward, Reynolds' original paper is lacking in terms of technical details. Making it difficult to replicate its work in a faithful way.

Customised (à la Parker) implementation of Reynolds' algorithm

As previously mentioned there is some degree of uncertainty about how Reynolds originally implemented his model. His original paper, although well written and capable of stimulating endless discussions, is quite minimalist in terms of technical details. It provides an overview of the general principles followed in order to achieve the flocking behaviour, but does not go particularly far in describing how exactly the entire procedure should be implemented in terms of computer code. For this reason, over the years many researchers have proposed their own implementations of Reynolds' algorithm. The approach we have decide to take inspiration from is the one recently suggested by Conrad Parker[32].

Parker's pseudocode for the separation and cohesion rules is shown in Algorithms 7 and 8 respectively. Consider that b_i and b_j represent respectively the i-th and the j-th boids belonging to the flock; while N can be interpreted either as the total amount of boids in the flock or just as the subset of those within i's neighbourhood.

We have decided to forgo using one of Reynolds' three original rules (namely

[31]The weights associated to the rules are fractional values included between 0 and 1.

[32]The pseudo-code written by Parker, as well as a detailed explanation about the various assumptions he made in writing it, can be found online at the URL: `http://www.kfish.org/boids/pseudocode.html`.

Algorithm 7 Parker's pseudocode for flocking: cohesion rule for boid i

```
vector c;
for j = 1 : N do
  if i ≠ j then
    c = c + b_j.position;
  end if
end for
c = c/(N − 1);
return (c − b_j.position)/100;
```

Algorithm 8 Parker's pseudocode for flocking: separation rule for boid i

```
vector s = 0;
for j = 1 : N do
  if i ≠ j then
    if |b_i.position − b_j.position| < 100 then
      c = c − (b_i.position − b_j.position);
    end if
  end if
end for
return s;
```

speed matching), thus just relying on two of them. The reason for this is because using a point of attraction (waypoints, in our case) valid for all the MAVs, would make redundant (and possibly counterproductive) to use the alignment/velocity-matching rule. Notwithstanding how many rules are employed, Algorithm 9 shows how any number of these can be assembled together in order to generate a single steering manoeuvre and the consequent movement of boid b to the desired location. In this algorithm $w_1, w_2, w_3, ...$ represent the weight factors applied to the vectors $v_1, v_2, v_3, ...$ generated by rules 1, 2, 3, ... respectively. As can be seen Parker has not implemented any sort of accumulator, but has simply relied upon a weighted sum of all the vectors created by the individual flocking rules.

Algorithm 9 Parker's pseudocode for flocking: assembling the rules together

```
vector v_1, v_2, v_3, ...; total = 0;
int w_1, w_2, w_3, ...;
for i = 1 : N do
  v_1 = w_1 · rule1(b_i);
  v_2 = w_2 · rule2(b_i);
  v_3 = w_3 · rule3(b_i);
  ...
  total = total + v_1 + v_2 + v_3 + ...;
  b_i.position = b_i.position + b_i.velocity;
end for
```

Because of the autopilot system described in Section 7.1 we only consider two dimensions (X and Y) while extrapolating the vectors from the flocking rules, assuming that the value of the altitude of the MAVs flight is constant.

Assembling the rules together has been done through the pseudocode shown in Algorithm 10, for which the parameters were fixed at the end of a trial-and-error procedure. *calculateCohesionVector(id)* and *calculateSeparationVector(id)* are two functions returning the vectors for the *id*-th MAV generated by the cohesion and the separation rules respectively (Algorithms 7 and 8). These two vectors are summed to the one representing the current position of MAV_{id} in order to obtain a vector which identifies the desired position towards which the MAV should aim. Since the modelled aircraft is a fixed wing one (and cannot therefore move to the destination point ignoring its current orientation), out of this data we must extrapolate the delta angle ($\Delta\alpha$) between the current heading and the desired vector. $\Delta\alpha$ is first divided by 40 - as for Equation 7.5 - then Gaussian distributed noise is added. On top of that, the resulting value (which is the amount of yaw steering the MAV has to perform) is further divided by three because of the observations extrapolated by our experiments. This seems to suggest how the flocking behaviour becomes more efficient when the steering value is attenuated.

Algorithm 10 Parker's-inspired pseudocode for flocking: assembling the rules together for boid i

$\vec{v_1} = \frac{1}{1} * calculateCohesionVector(MAV_i)$;
$\vec{v_2} = \frac{1}{50} * calculateSeparationVector(MAV_i)$;
$resulting\vec{V}ector = M\vec{A}V_i + \vec{v_1} + \vec{v_2}$
$\Delta\alpha = calculateDeltaHeadingToResultingVector(MAV_i)$;
$performYaw(MAV_i, (\Delta\alpha \div 40 + X \sim (0, 0.25)) \div 3)$;

7.3 Moving from simulations to reality: overcoming the reality gap

Moving from computer simulations to real robots is always a challenging task. We have already discussed the so-called "reality gap" in section 4.3.6. In this case, the job is made significantly easier by the robotics platform we are relying on. When

we refer to the platform as in this case, we do not generally intend to point our attention to its "physical" component, but rather to its software. We are looking in particular at the embedded autopilot system, which provides a very good level of flight stabilisation in addition to the possibility of receiving/executing flight instructions generated in real time by another piece of software running on the onboard PC104 computer. With such a system available, we can focus on the more practical aspects of our experiments, relying on the fact that the platform will react most of the time in the expected way to the commands issued.

Despite this, many other issues remain. Working with physical robots interacting in a real world environment means that the readings coming from the MAVs' sensors will often be inaccurate (when not completely absent) and that the communication exchanges between vehicles will sometimes fail, etc.. All of these issues have been taken into account when writing the code for the autonomous controllers.

Specifically looking at our experimental setup, we have found an additional problem as soon as we have moved towards real-world experiments. When it comes to implementing the flocking behaviours on real flying robots one of the first issues that arises concerns making sure that all the vehicles involved in the flocking might be close enough to each other in order to exchange information and move according to the algorithm designed. Although many alternative ways to approach this issue are available, the problem is not straightforward. The MAVs take off from the ground at different times and the autopilot system embedded on their fuselage makes them follow different flight paths to reach the desired altitude, thus they end up being scattered into the environment and are potentially very far (a few hundred meters) from each other.

The simple approach we have decided to implement consists in designating a leader MAV which takes off and once it has reached the desired altitude and it starts navigating back and forth between two fixed waypoints. The other MAVs, that start their tests some time after the leader, combine the aforementioned "follow the leader" navigation algorithm with the "speed adjustment" flocking algorithm. In this way, the followers get progressively closer to the leader until all of them are

next to its tail. When this stage is reached (which can be determined either using a threshold function or relying on the experimenter's intuition), the Boid-like flocking algorithm can be activated.

7.3.1 Test field and coordinate systems

The tests involving physical robots were carried out over a rural area within the village of Bioley-Orjulaz (VD, Switzerland) (see Figure 7.10). Authorisation was given by the Swiss Federal Office for Civil Aviation[33] to perform experiments below $150m$ of altitude in this area.

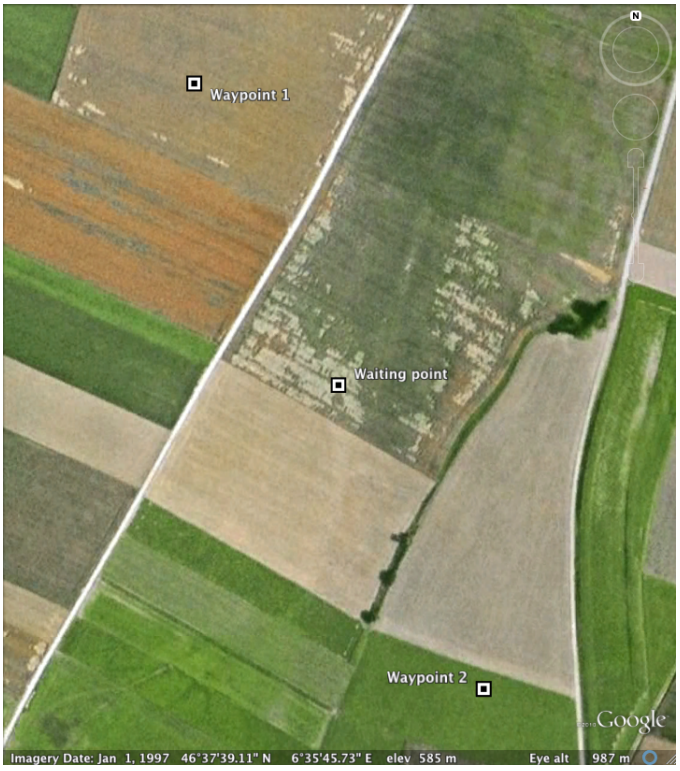


Figure 7.10: Satellite image of the test field. Designed using Google Earth (`http://earth.google.co.uk`)

In the computer simulator the MAVs were using a fixed Cartesian coordinate system, which is not available to the real robots. The MAVs can nonetheless access GPS information through their embedded receivers, and use this information to generate a virtual XY-like reference system more accurate than the GPS data in itself for navigation purposes. The method we have chosen to make the robots do this consists of translating the GPS coordinates into an ECEF (Earth-Centred, Earth-

[33] `http://www.bazl.admin.ch/index.html?lang=en`

Fixed) Cartesian system. ECEF, also known as "conventional terrestrial" system represents positions as X, Y, and Z coordinates, with $(0, 0, 0)$ being the mass centre of the Earth. The X axe passes through the equator at the prime meridian, the Z axe passes through the north pole, and the Y axe can be determined by the right-hand rule to be passing through the equator at 90° longitude[34] (see Figure 7.11). The conversion from GPS to ECEF is performed in real time by the controller software running on the MAVs, according to Equations D.5 and D.6[35].

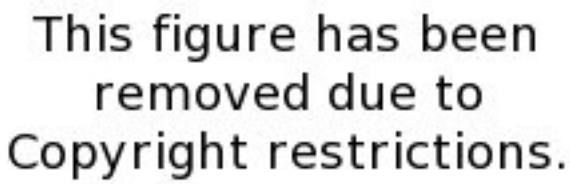


Figure 7.11: ECEF (Earth Centred, Earth Fixed coordinate system) reference frame. Source: `http://en.wikipedia.org/wiki/ECEF`

Metrics used

Four metrics have been elaborated in order to evaluate the performances generated by the navigation and flocking algorithms: 1) area covered by the flock; 2) average distance between the flock members; 3) mean heading; and 4) standard deviation from the mean heading. Assuming that N is the number of MAVs in the flock, h_i the absolute heading (within the range $[-180; 180]$) of the i-th MAV, x_i and y_i the x and y coordinates of the i-th MAV respectively.

The area covered by the flock at a certain time is calculated by taking into

[34] `http://en.wikipedia.org/wiki/ECEF`

[35] Although the process described in the Appendix does not match exactly with the one described by Drake, the reader can find more information about GPS data conversion looking at [97].

account the XY coordinates of all the MAVs (considered as vertices of a bounding polygon) according to Equation 7.6.

$$A = \frac{1}{2}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \begin{vmatrix} \dots & \dots \\ \dots & \dots \end{vmatrix} + \begin{vmatrix} x_N & x_1 \\ y_N & y_1 \end{vmatrix}\right) \tag{7.6}$$

The average distance $\overline{d}$ between flock members corresponds to the mean of all the $\binom{N}{2}$ intra-flock distances, individually calculated in accordance with Pythagoras' Theorem (see Equation 7.7).

$$\overline{d} = \frac{\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}{\binom{N}{2}} \tag{7.7}$$

The average heading $\overline{h}$ is calculated as a mean of circular quantities (see Appendix D.3). The computation is performed through the C++ *atan2()* function, fed in input with the sums of the sin and of the cosine for the headings of all the flock members, as expressed in Equation 7.8[36].

$$\overline{h} = atan2(\sum_{i=1}^{N}\sin h_i, \sum_{i=1}^{N}\cos h_i) \times \frac{180}{\pi} \tag{7.8}$$

Finally, the standard deviation from the average heading indicates how much each MAV deviates, on average, from the mean heading of the flock. The formula for the calculation is reported in Equation 7.9, where the difference $(h_i - \overline{h})$ is calculated as a subtraction between circular quantities.

$$\sigma_h = \sqrt{\frac{\sum_{i=1}^{N}(h_i - \overline{h})^2}{N}} \tag{7.9}$$

7.3.2 Leader following behaviour through explicit communication (rendezvous)

The first experiment described herein is focused on the approach described at the beginning of Section 7.3. A leader MAV is the first vehicle to take off from the ground and, once it reaches the desired altitude, it starts navigating between two waypoints.

[36] $atan2()$ returns a value in radians which in Equation 7.8 is converted to degrees.

The other MAVs sequentially take off later on, simply aiming to follow the leader and to adjust their cruise speed in order to get as close to it as possible. The combination of following a leader navigating through fixed waypoints with speed modifications has been labeled "rendezvous behaviour". Algorithm 5 shows, as pseudo-code, the rules the followers obey to modify their speed in order to approach the leader MAV.

Algorithm 11 Flocking simulator: speed adjustment for rendezvous behaviour (i-th follower)

```
if distance(MAV_i, waypoint) > distance(leader, waypoint) then
  if distance(MAV_i, leader) > (flockingdistance + 2) then
    increaseSpeed(MAV_i);
  else if distance(MAV_i, leader) < (flockingdistance − 2) then
    decreaseSpeed(MAV_i);
  end if
else
  decreaseSpeed(MAV_i);
end if
```

Waypoint navigation has been implemented in a slightly different way than what was described in Section 7.2.2. In this scenario, as the leader is the only MAV to navigate through waypoints, the active waypoint changes when the leader reaches it. If one of the following MAVs happen to reach the waypoint before the leader, the latter will keep flying towards that waypoint without modifying its behaviour.

Results on simulation

The rendezvous behaviour has been tested both in simulation and on physical robots. For what concerns the computer-based experiments, a few details have to be highlighted before looking at the results obtained.

First of all, the communication range for simulated MAVs is assumed to be infinite. This means that the followers can receive information about the leader position and the coordinates of the current waypoint regardless of how far they are from the leading aircraft. The two waypoints the leader cycles through are deployed in coordinates $(-165, 165, 165)$ and $(165, 165, -165)$ respectively. The second MAV takes off 500 time steps after the leader has started. The metrics shown in the following graphs are collected after the take off of the leader, but they only become meaningful after the follower is flying also.

Figure 7.12 shows the flight paths followed by the leader (marked in blue) and by one single follower MAV (green line). Both the aircraft take off from the point of coordinates $(0, 0, 0)$, and follow exactly the same path to reach the desired altitude (which corresponds to $165GU$). What can be seen is how the leader, once it has reached the proper flight altitude, immediately points to the waypoint in $(-165, 165, 165)$ and, once reached, starts going back and forth from the two waypoints. The follower flies according to a different path for the second part of its flight as it just goes after the leader. The follower reaches its desired altitude (thus starting its navigation algorithm) when the leader is already flying towards the waypoint in $(165, 165, -165)$. Very soon this waypoint is reached by the leader who then suddenly changes its aim towards $(-165, 165, 165)$, provoking the follower to perform a U-turn in order to approach it. It is during this phase that the follower, increasing its speed, eventually reaches the leader. The rendezvous is accomplished.

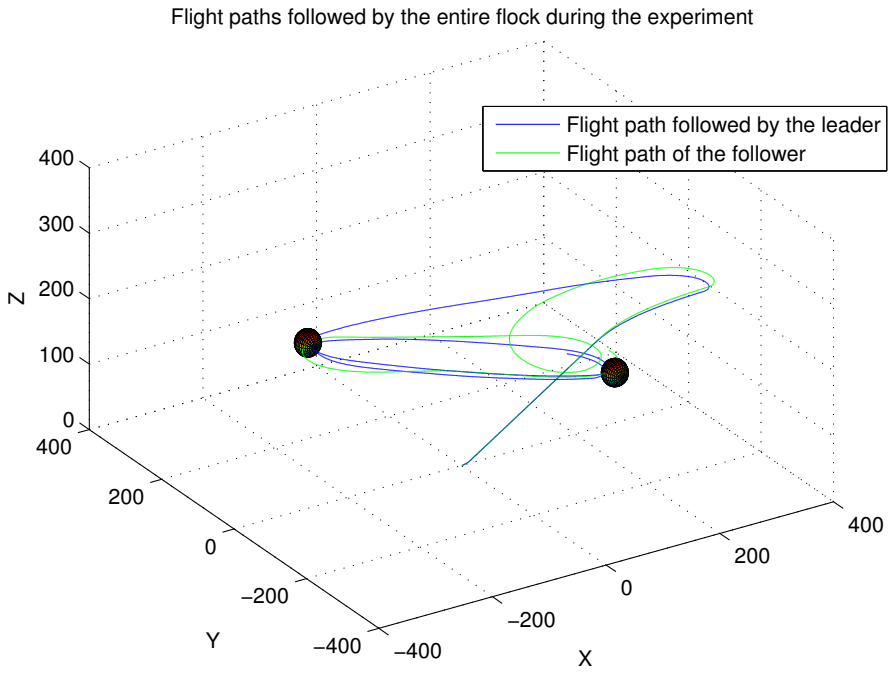


Figure 7.12: Rendezvous behaviour: flight paths followed in simulation by two MAVs, one of which being the leader and the other one the follower

Figure 7.13 displays the distance between the leader and the follower at the various time steps. When the follower takes off the distance between the two agents is about $400GU$. Since the takeoff path brings the MAV slightly far from the centre of the environment this distance quickly increases, reaching a maximum of about

600GU. Then the navigation algorithm comes into play and the follower quickly reduces its distance to the leader until being able to constantly keep it under the 100GU threshold.

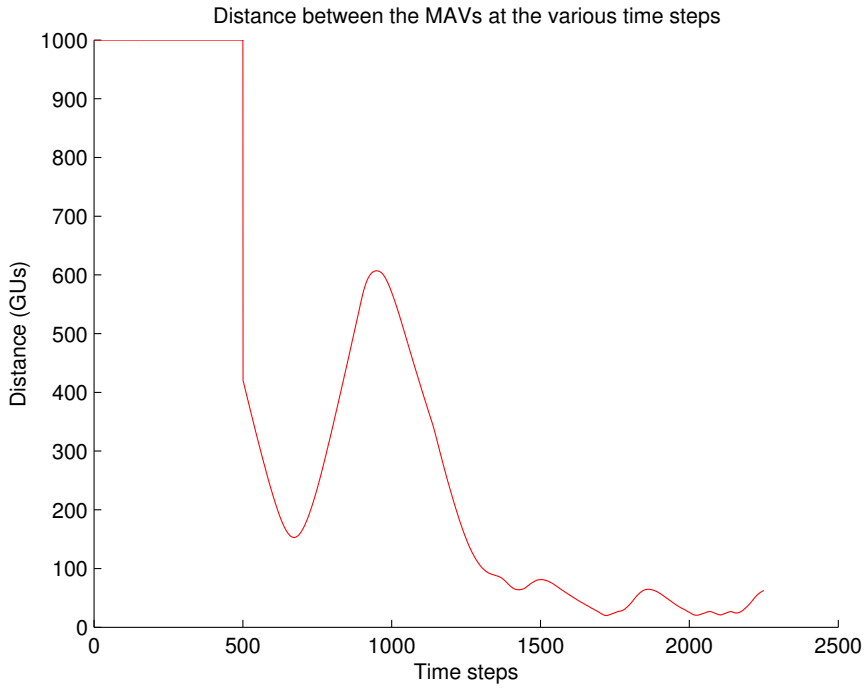


Figure 7.13: Rendezvous behaviour: distance in simulation between the leader and the follower obtained in simulation at the various time steps

Figure 7.14 shows the standard deviation in the heading of the two MAVs during the various time steps. As expected as a direct consequence of the behaviour described above, when the follower takes off from the ground the heading directions of the two agents are very different from each other, as any of them can follow different navigation rules (the leader already cycling between waypoints, the follower still busy attempting to reach the desired altitude). As soon as the follower switches to "follow the leader" mode the standard deviation begins to fall until reaching fairly low absolute values, well under 10. The pattern of this graph after 1,500 time steps presents continuous "jumps". These jumps correspond to the moments in which the leader quickly turns because it has reached the current waypoint. As the follower blindly reacts to what the leader does, it requires some time steps in order to regain the desired alignment. During these transitions, the headings of the two agents will be significantly misaligned, thus explaining the pattern shown in the graph.

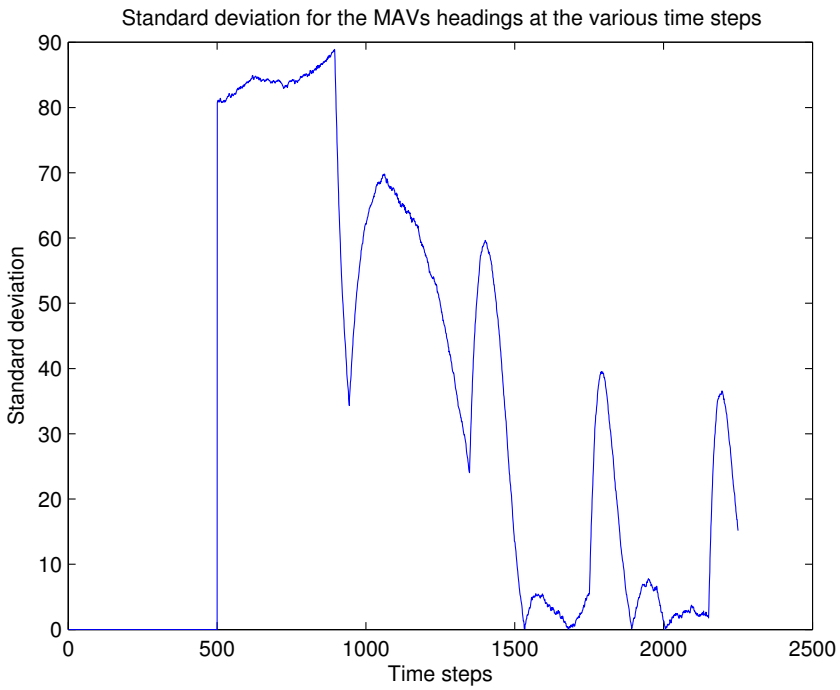


Figure 7.14: Rendezvous behaviour: standard deviation in simulation for the MAVs' heading obtained in simulation at the various time steps

Results on physical robots

A few modifications to the rendezvous algorithm have been required in order to make it work effectively on real robots. The main issue we have had to address is related to the fact that physical robots have a limited communication range. On top of that, wireless radio signals sent through the air often fail to reach the intended destination for a number of reasons. For example, the USB Wi-Fi dongle installed on the *swinglet*, as mentioned in section 7.1, has a theoretical maximum communication range of $500m$ along a free line-of-sight. In reality, a reliable communication between two devices like these requires them to be at a much lesser distance. As the MAVs used can easily fly for a few hundred meters in opposite directions, it is pretty much guaranteed that there will be, during any tests, moments in which a clear and working communication link between two or more aircraft is not available.

To tackle this problem we have introduced a new functionality in the controller driving the followers. Whenever one of the followers has spent the last 2.5 seconds without receiving any message coming from the leader, it automatically switches to waypoint navigation. This algorithm does not rely on the previously introduced

waypoints #1 and #2, rather on a third one labelled "waiting point". This waypoint is dislocated more or less midway between the other two, thus guaranteeing that flying around the area a communication link with the leader MAV, which regularly passes over that zone, will be eventually re-established.

The coordinates of the waypoints used for the experiments described here, together with the correspondent ECEF values, are reported in Table 7.1 and graphically shown in Figure 7.10. The two waypoints flown through by the leader are located 465.13 meters far from each other, with the waiting point being stationed roughly half-way (230m from each of those).

Table 7.1: Coordinates of the waypoints used in experiments with real robots

Waypoint	GPS longitude	GPS latitude	ECEF X	ECEF Y
Waypoint #1	6.5947686°	46.6293520°	504124.85	5190644.22
Waypoint #2	6.5973049°	46.6257091°	504352.67	5190238.71
Waiting point	6.5960367°	46.6275306°	504238.76	5190441.47

The communication amongst the MAVs is uni-directional, as it is the leader MAV only that broadcasts information. Information that has to be received, interpreted and elaborated accordingly by the followers. The structure of the messages sent by the leader is quite straightforward. Its components can be seen resumed in Table 7.2[37].

Table 7.2: Structure of the data messages broadcasted by the leader MAV

Variable name	Definition
*id**	identification code used by the leader
*lon**	GPS longitude
*lat**	GPS latitude
*goalAlt**	goal altitude
*metric**	set of internal autopilot parameters
currWP	identification code of the current waypoint followed
currWPLat	GPS longitude of the current waypoint followed
currWPLon	GPS latitude of the current waypoint followed

The most important parameters are the current GPS coordinates of the MAV

[37]The variables marked with the star are those that must be broadcast, whether they are used or not, because of how the autopilot system has been designed from a software point of view.

and those of the current waypoint. These two pieces of information are required by the followers both to determine the position of the leader (and modify their turn rate accordingly), and to check whether they are in front or behind it (in order to adjust the cruise speed). Some of the information transmitted is redundant. For example, since the waypoints that the leader flies between are fixed, in principle there is no need to transmit their coordinates attached to every message. Rather, hard-coding the GPS coordinates of the two waypoints inside the followers' controllers, would make it sufficient for the leader only to broadcast the ID of the current waypoint.

For what concerns the testing of this setup, one successful experiment has been carried out employing a flock made of four *swinglet* MAVs. The behaviour produced by the aircraft can be seen in a short movie available on the Internet[38], while Figure 7.15 shows the flight paths followed.

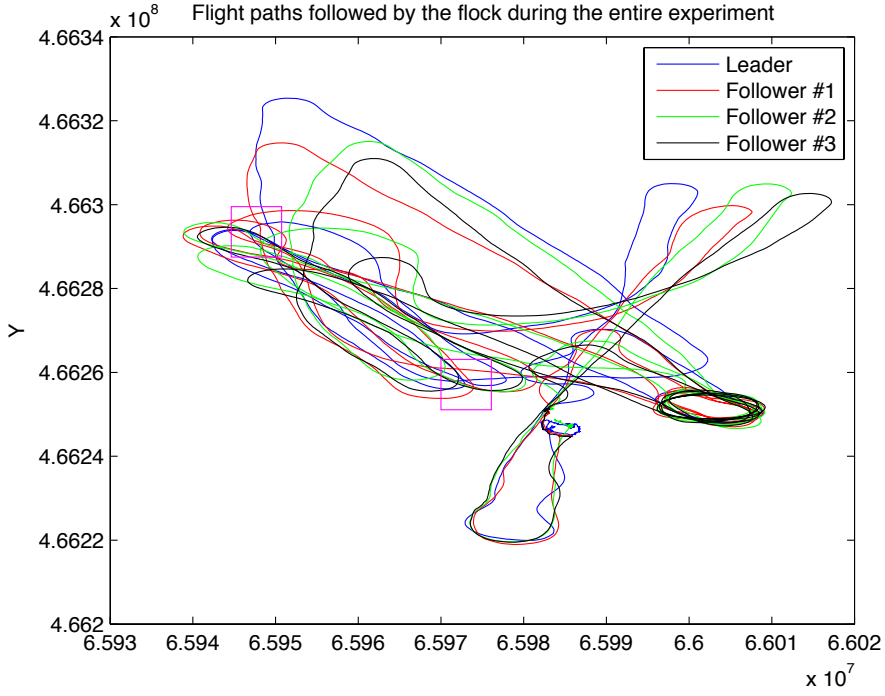


Figure 7.15: Rendezvous behaviour: flight paths followed in reality by four MAVs, one of which being the leader and the other three the followers

Looking at Figure 7.15 in more detail, we can identify a few key areas in it. First of all, the MAVs take off from the point with coordinates approximately $(6.5985, 46.625)$, perform a U-turn after less than $100m$ and start gaining elevation, progressively reaching their goal altitude, flying more or less straight (the deviations

[38] `http://www.fabioruini.eu/EPFL/rendezvous.mov`

from a straight line in this part of the graph are mostly due to physical factors such as wind, air resistance, etc.). As it can be seen in the top-right corner of the picture, all the MAVs used have followed quite different flight trajectories once they have reached the desired altitude. This is due to the previously discussed algorithm, which makes the leader aim towards the first waypoint and the followers come along after him. The two waypoints, whose coordinates have been reported in Table 7.1, are highlighted in the picture via two rectangles coloured in purple. The "waiting point" is not reported graphically, but stands more or less midway between the other two. When the test is considered completed, the MAVs go to the designated landing point (in coordinates that are approximately equal to $(6.607, 46.625)$), over which they fly in a circle decreasing their altitude until landing.

Unfortunately, no additional metrics are available at this stage because of some of the issues that have arisen during the only test the author has had the chance to supervise in person.

7.3.3 Flocking behaviour

As we have just seen, the rendezvous algorithm is used as a preliminary measure to ensure that all the MAVs involved in the experiment will get close enough to each other in order to be able to communicate. When this condition is satisfied, the flocking algorithm can be activated. The implementation of the Boid-like flocking behaviour used for the simulation experiments presented herein is resumed in Algorithm 12. The algorithm, obeyed by every aircraft, simply consists in two steps. The first one is the execution of the navigation algorithm, which provides to calculate and perform a desired turn manoeuvre for the MAV. The second one is the flocking part, implemented as one single manoeuvre which steers the MAV towards the optimal "flocking vector" (calculated as weighted sum of separation and cohesion vectors/rules).

Algorithm 12 Flocking simulator: navigation and flocking algorithm for boid i

% Navigation
$\Delta h = calculateHeadingDeviation(MAV_i, currentWaypoint)$;
$performYaw(MAV_i, \Delta h \div 4)$;

% Flocking
$\vec{v_1} = calculateSeparationVector(MAV_i)$;
$\vec{v_3} = calculateCohesionVector(MAV_i)$;
$\vec{total} = MAV_i.\vec{position} + \vec{v1} + (\vec{v3} \div 50)$;
$\Delta h = calculateHeadingDeviation(MAV_i, total)$;
$performYaw(MAV_i, \Delta h \div 4 \div 3)$;

Results on simulation

The flocking algorithm for which we have proposed the implementation has been tested on the computer simulator, where it has proven to be potentially successful.

In the experimental setup described in this section, a flock made of four MAVs is employed. They start from the middle of the environment, deployed in a V-formation. The basic navigation task is waypoint navigation. At any time, all the MAVs fly towards the same waypoint. Whenever one of the aircraft reaches the current waypoint (*e.g.* waypoint #1), the active waypoint switches to the other one (*e.g.* waypoint #2).

Figure 7.16 shows the flight paths followed by the MAVs. What we can immediately see when comparing this graph with Figure 7.12 is how the lines corresponding to the various flight paths do not tend to overlap with each other, but remain quite separated. This is exactly the sort of effect we would have expected by a flocking algorithm.

Figure 7.17 portraits the area covered by the flock during the simulation. What we are looking for in this graph is uniformity in the statistic measured across all the time steps. If the flocking algorithm works properly, the overall area covered by the flock as a whole should tend to be more or less the same at all time. Of course, in our scenario there is waypoint navigation involved. Due to this reason the flocking motion is often (whenever the flock has to perform a U-turn going from one waypoint to the other one) disturbed and struggles in reaching a stable value as it would happen instead in case of straight flight.

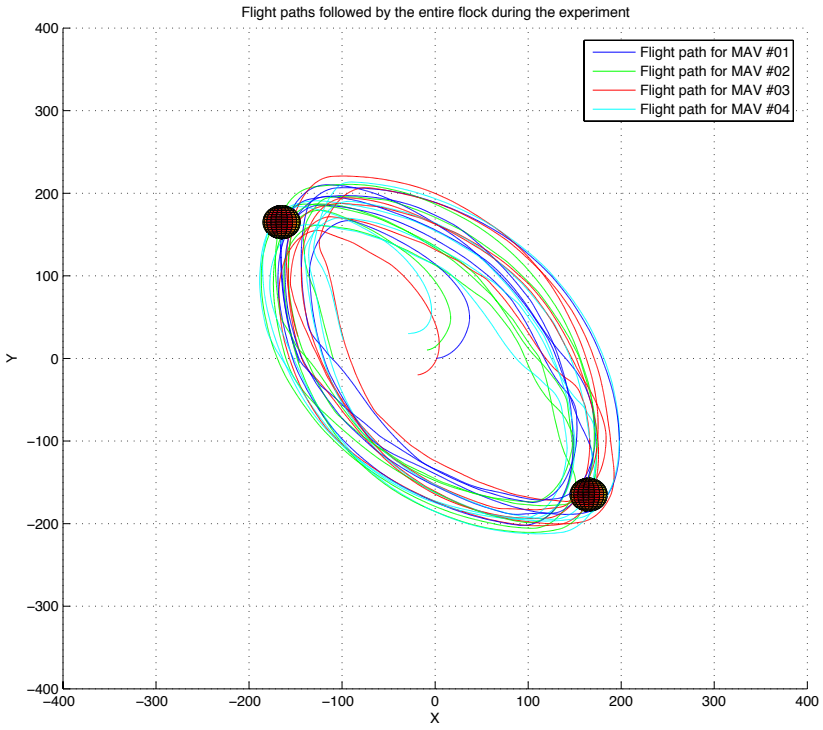


Figure 7.16: Flocking behaviour: flight paths followed in simulation by a flock of four MAVs

Figure 7.18 shows the average and minimum distance between the members of the flock at any time step. The average value of this statistic tends to vary quite frequently over time, while the minimum one remains much more stable instead. The horizontal green line plotted on the graph represents the threshold value ($25GU$) we have used for calculating the separation vector for the flocking. MAVs react to this rule rarely getting closer to each other than the threshold. When this happens they immediately respond and re-establish the desired flocking distance.

Finally, Figure 7.19 reports the standard deviation for the MAVs' headings. The value scored, most of the time ranging between 1 and 5, suggest how the headings tend to be aligned with each other.

Results on physical robots

Due to time constraints, the implementation of the flocking behaviour on teams of real robots has not been possible for the author. Rather, this part of the work has been carried out by other people working at the EPFL in Lausanne, namely Sabine

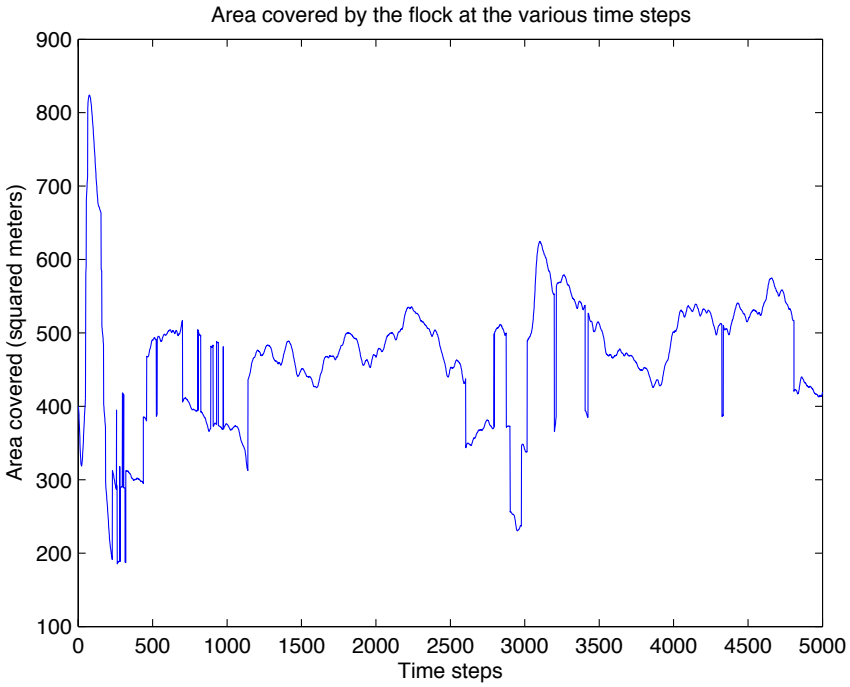


Figure 7.17: Flocking behaviour: area covered in simulation by a flock of four MAVs at the various time steps

Hauert, Severin Leven, and Maja Varga under the supervision of Jean-Cristophe Zufferey and Dario Floreano. This section is therefore based on the results they have obtained and published in a joint publication [157].

The focus of their work reflects their scientific interest in studying the impact of both motion constraints and communication range on the success of aerial flocking in reality.

A few modifications have been made on top of the work we have previously described herein. First of all, the robot motion has been analysed in simulation using a proper first order model (already validated in relation to the *swinglet* platform in [210]) rather than an incremental geometry approach. The model is described in Equation 7.10.

$$\begin{cases} x(t) = x(t - dt) + v \cdot cos(\omega \cdot dt) \cdot dt \\ y(t) = y(t - dt) + v \cdot sin(\omega \cdot dt) \cdot dt \\ z(t) = const \end{cases} \tag{7.10}$$

For what concerns the flocking behaviour, this has been implemented based

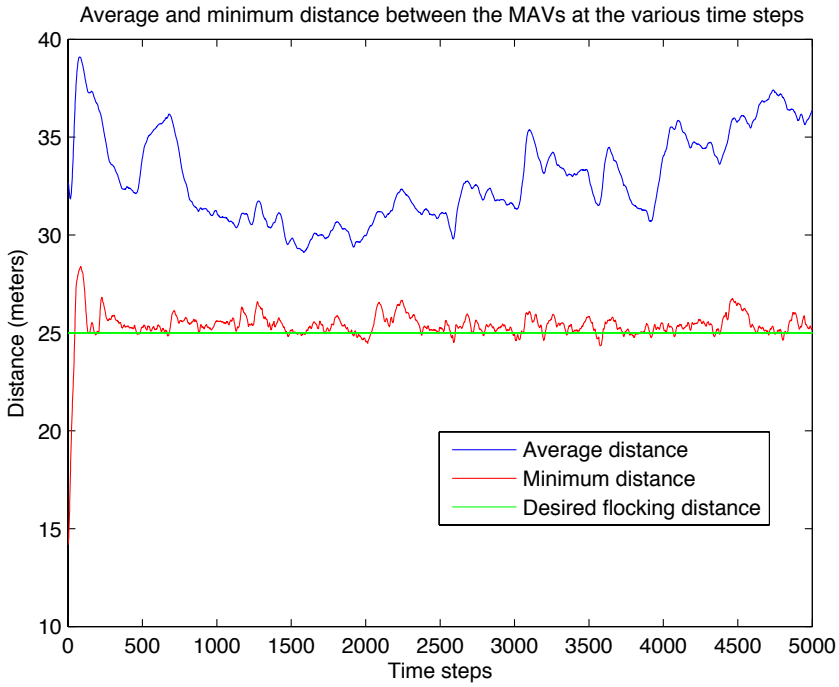


Figure 7.18: Flocking behaviour: average and minimum distance in simulation between members of the flock measured in simulation at the various time steps

on the entire set of rules elaborated by Reynolds, *i.e.* alignment, cohesion, and separation. In order to prevent robots from flying away during the test the migration rule (as theorised by Crowther [80]) has also been introduced.

In order to implement a realistic communication model in simulation, the authors have implemented Fenton's shadowing propagation model [106] to probabilistically determine the range of inter-robot transmissions. In the experiments involving physical robots, two different line-of-sight communication ranges of $50m$ and $300m$ respectively have been tested. The probability of having an effective communication in these two scenarios is shown in the graphics in Figure 7.20.

Flocks made of ten MAVs have been used. These teams have been evaluated for their capacity to flock coherently with varying communication ranges and maximum turn rates available to every aircraft. Figure 7.21 shows the flight paths followed during an entire trial performed on a flock made of nine *swinglet* robots.

Only two metrics have been employed to measure how well the flocking algorithm performs. The first one is the standard deviation on all robot headings (calculated as an average over the last minute of ten trials lasting 15 minutes each). The second one involves the elaboration of distance matrices storing all the inter-robots dis-

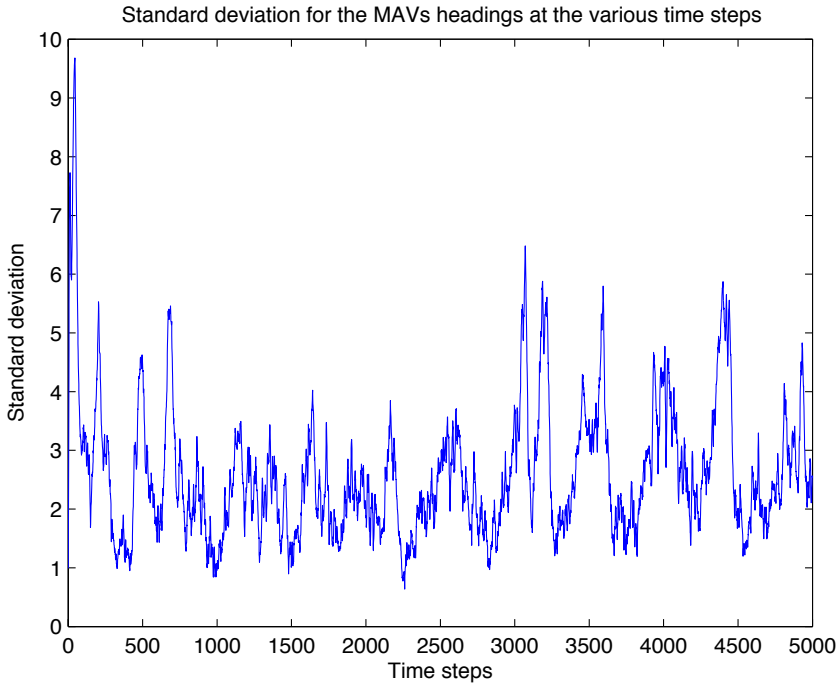


Figure 7.19: Flocking behaviour: standard deviation in simulation for the MAVs' heading obtained in simulation at the various time steps

tances within the flock. A distance matrix is calculated every second and compared with the one created one second before. At any given time, the absolute difference between the two matrices is computed and all the elements of the resulting matrix are summed together. This sum is then averaged over each second interval during the last minute of ten trials elapsed (each of them, again, lasting 15 minutes). Figure 7.22 displays the values measured for these two metrics, with Figure 7.22(a) focusing on the heading standard deviation and Figure 7.22(b) highlighting the intra-flock distance.

The results shown exhibit the expected tradeoff: the higher the turn rate and the communication range, the better the flocking behaviour emerged. Interestingly enough, the communication range seems to have a more significant impact than the turn rate in determining the ability of the MAVs to flock in a coherent way.

The configurations having obtained the best results in simulation have confirmed their validity on the experiments carried out on physical robots. By using a conservative parameter selection, with a communication range of $300m$ and a maximum turn rate of $0.7rad/sec$, coherent flocking on fixed-wing physical robots has been easily achieved.

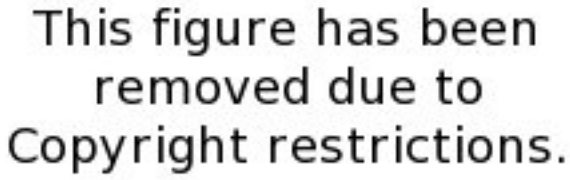


Figure 7.20: Inter-robot communication probability in function of the distance between the robots. The upper figure is related to Wi-Fi dongles set up for a $50m$ line-of-sight communication range, the lower one for a $300m$ range. The three lines in each plot respectively represent the lower quartile, the median, and the upper quartile. Source: [157]

7.4 Conclusions

The research presented in this chapter has covered two main aspects.

First of all, the work carried out on the new software simulator has demonstrated how the popular algorithm for Reynolds' flocking can be successfully applied to fixed-wing aircraft, notwithstanding their motion constraints. Basing our work on the implementation of the Reynolds' algorithm proposed by Parker, we have seen how teams of aircraft adhering to a subset of the classic flocking rules (separation and cohesion specifically) can form coherent flocks. Being able to flock gives to teams of MAVs several advantages. One of the most prominent consists in the fact that independently flying aircraft can gather together, create a wireless communication link between them, and share their resources in order to perform computationally intensive tasks (*e.g.* complex image processing) that would be otherwise impossible to be carried out in real-time by a single MAV. On top of that flocking behaviour we have also developed a "leader-following" algorithm, in which one of the aircraft takes the role of "leader" and drives the entire flock. The flocking behaviour has proven to be resilient, as the flock quickly reshapes itself in the event of sudden changes in

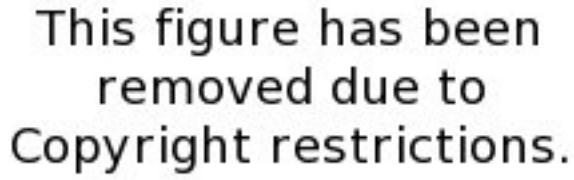
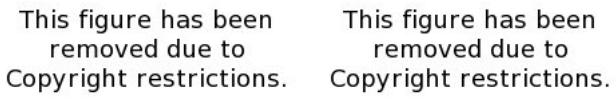


Figure 7.21: Flocking behaviour in reality: flight paths followed by a flock of nine MAVs employed during a trial. Source: [157]



(a) (b)

Figure 7.22: Graphical representations of how communication range and turn rate impact respectively on (a) the standard deviation in heading; (b) the intra-flock distance. Source: [157]

the direction of the leader that disrupt the flock. Also, a "rendezvous" algorithm has been proposed to help in creating a flock out of MAVs taking off independently from the ground at different times.

In the second part of the chapter we have seen the example of an autonomous controller implementing Reynolds' flocking and working on a real aircraft, specifically a SenseFly *swinglet* MAV. Despite the control algorithms eventually used on the aircraft being slightly different than those proposed in the first paragraphs, the underlying design principles are the same in terms of outputs they generate. This demonstrates how, once the compulsory fine-tuning of all the parameters involved is done successfully, the algorithm we have proposed in this chapter and, even most

importantly, those described in chapters 5 and 6, are expected to work effectively on a real robotics platform.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Many different topics have been touched upon in this thesis. The principal aim of the work presented herein was in extending the range of applicability of the Evolutionary Robotics approach to the design of autonomous controllers for fixed-wing aerial robots, a domain which did not receive a great deal of attention in the past from researchers in the ER field.

The 2D simulator presented in chapter 5 has provided a general framework for demonstrating how Evolutionary Robotics controllers can easily deal with the typical motion constraints of fixed-wing aircraft. In the various simulation experiments carried out fairly sophisticated behaviours have evolved, such as, apart from basic navigation towards a specific target area, obstacle avoidance (which is not by any mean an easy task when motion constraints as those characterising fixed-wing aircraft are involved), target tracking (in which the MAVs demonstrated to cope effectively with targets moving at different speeds), and cooperative/collective behaviour based upon implicit communication strategies. Concerning the last setup, from a theoretical point of view it has been shown how complex behaviours requiring a non-trivial degree of cooperation between the team members can be achieved through a very minimalist set of information (implicitly, *i.e.* not purposely) exchanged between the agents involved in the task. An interesting point is also in how

the cooperation emerges. No central controllers are used[1] in our experiments and the single individuals do not have any sort of knowledge about what the "global" task the team they are part of is expected to perform. Nonetheless, cooperative behaviours arise as a complex phenomenon from the simple interactions (amongst them and with the environment) of many individuals, each of them simply performing its own task alone. Individuals being evolved to maximise a fitness function which rewards the collective behaviour of the entire group, unconsciously making every MAV a gear in a bigger mechanism.

The experiments carried out on the two-dimensional simulator have led to several technical considerations. In the first experimental setup we have contrasted the performances of eight different neural network controller topologies, mostly focusing our attention on the different ways of encoding the input information (MAV-target distance and MAV-target relative angle) possible. While the sort of encoding used for the MAV-target distance did not seem to affect the performances of the controller very much, a more critical role to be played by the encoding of the MAV-target angle has resulted. Architectures encoding this information in a discrete way (with the angle divided in several "slices" each of them represented by a three-digit binary number via three input neurons) performed better than those relying on a continuous encoding. In the second experimental setup we have introduced obstacles inside the reference environment and evaluated different configurations of ultra-sonic sensors. The evolutionary process eventually managed to create a controller working with an accuracy of 100% for all the configurations tested, although the evolution went more smoothly for those relying on three sensors (one of which was facing forward). In the third experimental setup (where the target can spot an approaching MAV and move accordingly trying to escape from it) we have contrasted five scenarios in which we varied the speed of the target (we have used respectively one sixth, one fifth, one fourth, one third, and half the speed of the aircraft). For all the target's

[1] A critic might argue that the models we have proposed in this thesis are not examples of "absolute" distribute control, as they all rely on the existence of a "central" system knowing the location of the target and broadcasting this information in real-time to the MAVs. Nonetheless (and despite the fact that the "limitations" dictated by such an assumption could be overcame in several ways) we fully consider our models examples of distributed control, as several agents are coordinated, in some scenarios also to perform cooperatively, by individual and separated controllers exclusively.

moving speeds the evolutionary process led to controllers with a maximum success rate above the success threshold adopted (90%). Finally, also for what concerns the fourth experimental setup (in which the aircraft has to approach the target area in a coordinated fashion, thus making the task significantly more difficult to be dealt with for the MAVs' controllers), we successfully developed an autonomous controller working with over a 90% accuracy. Unfortunately, this was possible only with the target standing still in a certain position. Using a moving target has brought down the performances to slightly below the success threshold.

Chapter 5 has also presented some analysis carried out in order to test the validity of the simulation model we have created in terms of generalisation. A few modifications to the fitness function used for the genetic algorithm have allowed to evolve controllers coping effectively with a different environment than the one used in our simulations, at least for what concerns basic navigation and obstacle-avoidance.

The studies carried out using the 3D model and described in chapter 6 have replicated the experimental setups tested in the two-dimensions model, with the notable exception consisting in no obstacles being deployed into the environment anymore. The goal of these experiments was to evaluate whether controllers designed according to the Evolutionary Robotics principles could properly drive MAVs capable of manoeuvring over three dimensions (yaw, pitch, and roll) with levels of performance comparable to those achieved in the 2D simulations. Several interesting results have emerged from these experiments. The most prominent finding consists in an inverse correlation between success rate and degrees of freedom manoeuvrable by the controllers. This result was far from unexpected as, obviously, an autonomous controller struggles to control a device the more this device is complex. The controllers that emerged from the evolutionary process in the three-dimensional simulator have therefore proven to be highly efficient when the only degree of freedom available is yaw, less performant when both yaw and pitch are used, rarely good when also roll is introduced. We must nonetheless consider that the autopilot systems often being mentioned throughout this thesis can take care of flight stabilisation and low-level flight issues in general, thus working as a sort of middleware and relieving the

high-level controller (*i.e.*, in our case, the neural network) from having to deal with certain task. We can easily imagine, for example, that in real applications we do not need to put our controller in charge of the rotations around the roll axis.

Looking in more detail to the results obtained by these experiments, in the first set of simulations we aimed to identify the best neural network topology for basic navigation. Several architectures were contrasted (the main differences amongst them consisted in the degrees of freedom available, in the presence/absence of a layer of hidden units, and in being purely feed-forward rather than including memory components) and most of them exceeded the 90% success rate for the best individuals. Particularly poor performances have been nonetheless produced by networks implementing memory structures (Jordan and Elmann networks). One architecture managed anyway to exceed our success threshold while in control of all the rotation axes available. This architecture was then used as a basis (together with another topology, similar but including a hidden layer) for the following analysis. In the second set of experiments (involving a target able to move away from the approaching MAV, with the movement repository of the target depending upon the degrees of freedom available to the aircraft) the results showed that satisfying performances can be obtained by evolved controllers dealing with targets moving at one fifth and one fourth of the MAVs' speed. For targets moving at one third of the speed of their chasers, the performance stopped at 89%, still very close to our success threshold. In the third and final experimental setup, the one involving implicit cooperation, unfortunately the results obtained have not been favourable. Out of the twelve controller topologies tested (half of those designed for the first 3D experimental setup), only one architecture (working on a single DoF) managed to carry out the tests with a success rate sufficient to exceed the 90% success threshold we decided to use. Things have improved quite clearly adopting incremental evolution, but not enough to qualitatively modify the meaning of the results obtained in this scenario. Although it has been demonstrated that basic navigation, obstacle avoidance, and target tracking are tasks that an evolutionary controller can comfortably deal with both in two and three dimensional models, at the current stage more complex tasks

requiring cooperation seem not to be solvable by evolutionary controllers designed according to our methodology.

Chapter 7 has described an alternative approach to collective behaviour and distributed control that we have developed, based upon flocking algorithms. The experiments on flocking behaviour have an importance that goes well beyond the results presented in that chapter. What has been demonstrated is that, thanks to the recent developments in the field of unmanned aerial vehicles, autonomous controllers for fixed-wing MAVs can be transferred from simulations to physical robots in a relatively straightforward way. Again, this is mainly due to autopilot systems, as they can provide a simple interface to control an aircraft based on basic commands. An autonomous controller can indeed instruct the controlled aircraft to do anything that is within its behavioural repository, by simply producing a single value to be passed to the autopilot onboard. It is the latter, which takes care of all the low-level issues.

During the many informal discussions the author had with experts in the field at the beginning of his Ph.D. programme, the criticism that was raised most often concerned the alleged impossibility of designing autonomous controllers relying on software simulators that do not take into account all the aerodynamics effects existing in reality. What we have seen instead is that extremely basic computer models, as those presented here and based on incremental geometric flight, are more than adequate even for achieving fairly sophisticated behaviours. Low-level control issues (flight stabilisation, etc.) can be plainly ignored and the focus of the research shifted on the navigation part (or high-level control) only. In other words, designing autonomous controllers for fixed-wing aerial robots is not an unreasonably complex task anymore, and does not require in-depth knowledge of aerodynamics (as long as a "plausible" flight model is implemented). This opens the door for many innovative approaches to be employed with success on the domain. We have seen one of them in this thesis, but many others are possible.

8.2 Future work

There are a number of aspects of the present research that could be further addressed and extended. Amongst the possible future research directions that might be possible to investigate we can list those that are the most interesting from the author's point of view.

- *introduction of explicit communication*: good performances have been achieved in the cooperative tasks carried out in the 2D model using a simple form of implicit communication. This approach nonetheless has its limitations, as the array of tasks that autonomous robots can undertake based upon such a form of communication is necessarily restricted. Introducing the possibility of explicit communication, either relying on a vocabulary provided by the experimenter or using one developed by the robots themselves, should enable the extension of the range of cooperative tasks that the MAVs can successfully deal with;

- *obstacle avoidance in reality*: as we have seen, testing obstacle avoidance behaviours in reality when using fixed-wing aircraft is an extremely challenging task. This kind of aircraft must in fact be flying at a significant altitude, at which in general it is not possible to find any obstruction. A possible solution would be to carry out these tests within a urban area where tall buildings are present. Unfortunately, due to understandable safety reasons, it is not possible to obtain the required authorisation to perform experiments such as these. Furthermore, this kind of aerial platform tends to be relatively fragile and the potential risk of damaging the MAVs during the tests would be high. A potential answer to this problem could consist in "simulating" the obstacles via software. The computer onboard the MAV may for example associate certain specific coordinates to "obstacles", thus causing false sensor readings entering the control loop. The aircraft would therefore react as if it were in the presence of an obstacle. Analysing the log files at the end of the experimental session would enable an evaluation of the effectiveness of the obstacle avoidance behaviour;

- *target tracking in reality*: with the availability of the proper hardware, it would be possible to test in reality the target tracking ability of the MAVs. One possible scenario would consist in having a wheeled vehicle moving on the ground (either autonomously - following a pre-planned navigation path or moving in response to its sensorial input - or remotely controlled by a human operator) constantly emitting data signals. Equipping the aircraft with receivers capable of capturing that signal, its strength and direction could be used to infer its position, in turn making possible its tracking.

Bibliography

[1] AAström, K. J., and Hägglund, T. *PID Controllers: Theory, Design, and Tuning*, 2nd ed. Instrument Society of America, Research Triangle Park, NC, USA, 1995.

[2] Aberle, B. *Clean & Quiet: The Guide to Electric Powered Flight*. Covered Bridge Press, N. Attleboro, MA, USA, 1996.

[3] Ablavsky, V., Stouch, D., and Snorrason, M. Search Path Optimization for UAVs using Stochastic Sampling with Abstract Pattern Descriptors. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit* (2003).

[4] Abzug, M., and Larrabee, E. *Airplane Stability and Control: a History of the Technologies that Made Aviation Possible*. Cambridge University Press, Cambridge, MA, USA, 2002.

[5] Acerbi, A., and Parisi, D. Cultural Transmission Between and Within Generations. *Journal of Artificial Societies and Social Simulation 9*, 1 (2006).

[6] Afshar, S., Yousefi-Koma, A., Shahi, H., Mohammadshahi, D., and Maleki, H. Design and Fabrication of a Delta Wing Micro Aerial Vehicles. *International Journal of Mechanics 1*, 4 (2007), 51–58.

[7] Ali, K., and Carter, L. Miniature-Autopilot Evaluation System. *Journal of Computer Science 4*, 1 (2008), 30–35.

[8] Allen, J., and Walsh, B. Enhanced Oil Spill Surveillance, Detection and Monitoring through the Applied Technology of Unmanned Air Systems. In *Proceedings of the 2008 International Oil Spill conference* (2008), pp. 113–120.

[9] Allred, J., Hasan, A., Panichsakul, S., Pisano, W., Gray, P., Huang, J., Han, R., Lawrence, D., and Mohseni, K. SensorFlock: An Airborne Wireless Sensor Network of Micro-Air Vehicles. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems* (2007), pp. 117–129.

[10] Almeida, P., Bencatel, R., Goncalves, G., Sousa, J., and Ruetz, C. Experimental Results on Command and Control of Unmanned Air Vehicle Systems. In *Proceedings of IAV'07, the 6th IFAC Symposium on Intelligent Autonomous Vehicles* (2007).

[11] Alvis, W., Murthy, S., Valavanis, K., Moreno, W., and Katkoori, S. FPGA Based Flexible Autopilot Platform for Unmanned Systems. In

Proceedings of the 15th Mediterranean Conference on Control & Automation (2007), pp. T34–005.

[12] Amidi, O., Kanade, T., and Miller, R. Vision-Based Autonomous Helicopter Research at Carnegie Mellon Robotics Institute 1991-1997. *CMU Robotics Institute*, Paper 21 (1998).

[13] Anton, S., Erturk, A., and Inman, D. Energy Harvesting from Small Unmanned Air Vehicles. In *Proceedings of the 17th International Symposium on Application of Ferroelectrics, 3rd Annual Energy Harvesting Workshop* (2008).

[14] Antsaklis, P. Defining Intelligent Control. Report of the Task Force on Intelligent Control. Tech. Rep. December, IEEE Task Force on Intelligent Control, 1993.

[15] Antsaklis, P. J., Passino, K. M., and Wang, S. An Introduction to Autonomous Control Systems. *IEEE Control Systems* (1991), 5–13.

[16] Arkin, R. C. Governing Lethal Behavior: Embedding Ethics in a Hybrid Deliberative/Reactive Robot Architecture - PART I: Motivation and Philosophy. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction* (2008), pp. 121–128.

[17] Asada, M., MacDorman, K., Ishiguro, H., and Kuniyoshi, Y. Cognitive Developmental Robotics as a New Paradigm for the Design of Humanoid Robots. *Robotics and Autonomous Systems 37* (2001), 185–193.

[18] Astrom, K., and Wittenmark, B. *Computer Controlled Systems: Theory and Design*. Prentice Hall Professional, New York, NY, USA, 1984.

[19] Back, T. Evolutionary algorithms. *AGM SIGBIO Newsletter - Special Edition on Biologically Motivated Computing 12*, 2 (1992), 26–31.

[20] Back, T., Hoffmeister, F., and Schwefel, H. P. A Survey of Evolution Strategies. In *Proceedings of ICGA 1991, the Fourth International Conference on Genetic Algorithms* (1991).

[21] Back, T., Rudolph, G., and Schwefel, H.-P. Evolutionary Programming and Evolution Strategies: Similarities and Differences. *Proceedings of the 2nd Annual Conference on Evolutionary Programming* (1993).

[22] Baker, J. Adaptive Selection Methods for Genetic Algorithms. In *Proceedings of ICGA 1985, the First International Conference on Genetic Algorithms* (1985), pp. 101–111.

[23] Baker, J. Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of ICGA 1987, the Second International Conference on Genetic Algorithms and their Application* (1987), pp. 14–21.

[24] Balch, T., and Arckin, R. C. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transactions on Robotics And Automation 14* (1999), 926–939.

[25] Bamberger, R., Watson, D., Scheidt, D., and Moore, K. Flight Demonstrations of Unmanned Aerial Vehicle Swarming Concepts. *John Hopkins APL Technical Digest 27*, 1 (2006), 41–55.

[26] Barlow, G., and Oh, C. Evolved Navigation Control for Unmanned Aerial Vehicles. *Frontiers in Evolutionary Robotics* (2008), 596–621.

[27] Barlow, G., Oh, C., and Grant, E. Incremental Evolution of Autonomous Controllers for Unmanned Aerial Vehicles using Multi-objective Genetic Programming. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems* (2004), pp. 689–694.

[28] Barshan, B., and Durrant-Whyte, H. Inertial Navigation Systems for Mobile Robots. *IEEE Transactions on Robotics and Automation 11*, 3 (1995), 328–342.

[29] Baumgartner, F. R., and Jones, B. D. *Agendas and Instability in American Politics*, 2nd ed. The University of Chicago Press, Chicago, IL, USA, 2009.

[30] Bayindir, L., and Sahin, E. A Review of Studies in Swarm Robotics. *Turkish Journal of Electrical Engineering 15*, 2 (2007), 115–147.

[31] Beard, R., McLain, T., Nelson, D., Kingston, D., and Johanson, D. Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs. In *Proceedings of the IEEE* (2006), vol. 94, pp. 1306–1324.

[32] Beer, R. A Dynamical Systems Perspective on Agent-Environment Interaction. *Artificial Intelligence 72* (1995), 173–215.

[33] Bekey, G. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press, Cambridge, MA, USA, 2005.

[34] Bell, A., Pace, A., and Santos, R. Evolutions of Communication. `http://www.cs.swarthmore.edu/~meeden/cs81/s09/finals/AlexAndrewRaul.pdf`, 2009.

[35] Berthouze, L., and Metta, G. Epigenetic Robotics: Modelling Cognitive Development in Robotic Systems. *Cognitive Systems Research*, 6 (2005), 189–192.

[36] Bertuccelli, L., and How, J. UAV Search for Dynamic Targets with Uncertain Motion Models. In *Proceedings of the 45th IEEE Conference on Decision & Control* (2006), pp. 5941–5946.

[37] Betts, R. K. Analysis, War, and Decision: Why Intelligence Failures Are inevitable. *World Politics 31*, 1 (1978), 61–89.

[38] Beyeler, A., Magnenat, S., and Habersaat, A. Ishtar: a Flexible and Lightweight Software for Remote Data Access. In *Proceedings of EMAV08, the 2008 European Micro Air Vehicle Conference* (2008).

[39] Beyer, H.-G. Evolution strategies. `http://www.scholarpedia.org/wiki/index.php?title=Evolution_strategies`, 2007.

[40] Beyer, H.-G., and Schwefel, H.-P. Evolution strategies. A Comprehensive Introduction. *Natural Computing 1* (2002), 3–52.

[41] Biro, D., Sumpter, D., Meade, J., and Guilford, T. From Compromise to Leadership in Pigeon Homing. *Current Biology*, 16 (2006), 2123–2128.

[42] Blackman, S., and Popoli, R. *Design and Analysis of Modern Tracking Systems*. Artech House, Boston, MA, USA, 1999.

[43] Bolkcom, C. Homeland Security: Unmanned Aerial Vehicles and Border Surveillance. CSR Report for Congress RS21698, U.S. Congress, 2004.

[44] Bonabeau, E. Agent-based Modeling: Methods and Techniques for Simulating Human Systems. *PNAS 99*, 3 (2002), 7280–7287.

[45] Bongard, J., and Paul, C. Making Evolution an Offer It Can't Refuse: Morphology and the Extradimensional Bypass. In *Proceedings of ECAL 2001, the Sixth European Conference on Artificial Life* (2001), pp. 401–412.

[46] Borenstein, J. The Ethics of Autonomous Military Robots. *Studies in Ethics, Law, and Technology 2*, 1 (2008).

[47] Borenstein, J., and Koren, Y. Obstacle Avoidance with Ultrasonic Sensors. *IEEE Journal of Robotics and Automation 4*, 2 (1988), 213–218.

[48] Boschetti, F., Prokopenko, M., Macreadie, I., and Grisogono, A.-M. Defining and Detecting Emergence in Complex Networks. In *Proceedings of KES 2005, the 9th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems* (2005), pp. 573–580.

[49] Braitenberg, V. *Vehicles. Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, USA, 1984.

[50] Brooks, R. A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics and Automation 2*, 1 (1986), 14–23.

[51] Brooks, R. A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network. *Neural Computation 1*, 2 (1989), 253–262.

[52] Brooks, R. Intelligence Without Representation. *Artificial Intelligence*, 47 (1991), 139–159.

[53] Brooks, R. Artificial Life and Real Robots. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (Cambridge, MA, USA, 1992), F. J. Varela and P. Bourgine, Eds., MIT Press, pp. 3–10.

[54] Brooks, R. *Flesh and Machines. How Robots will Change Us*. Pantheon, New York, NY, USA, 2002.

[55] Bryson, A., and Ho, Y.-C. *Applied Optimal Control. Optimization, Estimation, and Control*. Taylor & Francis, London, UK, 1975.

[56] Burdakov, O., Doherty, P., Holmberg, K., Kvarnstrom, J., and Olsson, P.-M. Positioning Unmanned Aerial Vehicles as Communication Relays for Surveillance Tasks. In *Proceedings of Robotics: Science and Systems Conference* (2009).

[57] Buskey, G., Roberts, J., Corke, P., Dunbabin, M., and Wyeth, G. The CSIRO Autonomous Helicopter Project. In *Proceedings of the International Symposium on Experimental Robotics* (2002).

[58] Buskey, G., Roberts, J., Corke, P., Ridley, P., and Wyeth, G. Sensing and Control for a Small-Size Helicopter. In *Experimental Robotics VIII*, B. Siciliano and P. Dario, Eds. Springer-Verlag, Berlin Heideberg, Germany, 2003, pp. 476–486.

[59] Buskey, G., Wyeth, G., and Roberts, J. Autonomous Helicopter Hover Using an Artificial Neural Network. In *Proceedings ICRA 2001, the IEEE International Conference on Robotics and Automation* (2001), vol. 2, pp. 1635–1640.

[60] Cai, G., Cai, A., Chen, B., and Lee, T. Construction, Modeling and Control of a Mini Autonomous UAV Helicopter. In *Proceedings of ICAL 2008, the IEEE International Conference on Automation and Logistics* (2008), pp. 449–454.

[61] Cai, G., Chen, B., and Lee, T. An Overview on Development of Miniature Unmanned Rotorcraft Systems. *Frontiers of Electrical and Electronic Engineering in China 5*, 1 (2010), 1–14.

[62] Cambone, S., Krieg, K., Pace, P., and II, L. W. Unmanned Aircraft Systems Roadmap 2005-2030. Tech. rep., U.S. Department of Defense, Office of the Secretary of Defense, 2005.

[63] Canamero, L. Emotion Understanding From the Perspective of Autonomous Robots Search. *Neural Networks*, 18 (2005), 445–455.

[64] Canning, J. S. A Concept of Operations for Armed Autonomous Systems. `http://www.dtic.mil/ndia/2006disruptive\_tech/canning.pdf`, 2006.

[65] Cannon, H. G. What Lamarck really said. *Proceedings of the Linnean Society of London 168*, 1-2 (1957), 70–87.

[66] Chao, H., Cao, Y., and Chen, Y. Autopilots for Small Fixed-Wing Unmanned Air Vehicles: A Survey. *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation* (2007), 3144–3149.

[67] Chew, A. After London: The Role of the Micro Air Vehicle (MAV) in Counter-Terrorism. `http://www.isn.ethz.ch/isn/Digital-Library/Publications/Detail/?ots591=0c54e3b3-1e9c-be1e-2c24-a6a8c7060233&lng=en&id=13374`, 2005.

[68] Childress, J. F. Just-War Theories: The Bases, Interrelations, Priorities, and Functions of Their Criteria. *Theological Studies 39*, 3 (1978), 427–445.

[69] Christensen, A. L., and Dorigo, M. Incremental Evolution of Robot Controllers for a Highly Integrated Task. *From Animals to Animats 9. Proceedings of SAB 2006, the 9th International Conference on Simulation of Adaptive Behavior* (2006), 473–484.

[70] Clapper, J., Young, J., Cartwright, J., and Grimes, J. Unmanned Systems Roadmap 2007-2032. Tech. rep., U.S. Department of Defense, 2007.

[71] Clark, A., and Grush, R. Towards a Cognitive Robotics. *Adaptive Behaviour 7*, 1 (1999), 5–16.

[72] Clarke, R. A., and Knake, R. *Cyber War. The Next Threat to National Security and What to Do About It.* Ecco Press, Manhattan, NY, 2010.

[73] Cliff, D., Harvey, I., and Husbands, P. Explorations in Evolutionary Robotics. *Adaptive Behavior 2*, 1 (1993), 73–110.

[74] Cohen, C., McNally, B., and Parkinson, B. Flight Tests of Attitude Determination Using GPS Compared Against an Inertial Navigation Unit. In *Proceedings of the ION National Technical Meeting* (1993).

[75] Coifman, B., McCord, M., Mishalani, R., and Redmill, K. Surface Transportation Surveillance from Unmanned Aerial Vehicles. In *Proceedings of the 83rd Annual Meeting of the Transportation Research Board* (2004).

[76] Cook, M. *Flight Dynamics Principles.* Elsevier: Butterworth-Heinemann, Waltan, MA, USA, 2007.

[77] Corner, J. J., and Lamont, G. B. Parallel Simulation of UAV Swarm Scenarios. In *Proceedings of the 2004 Winter Simulation Conference* (2004), pp. 355–363.

[78] Craighead, J., Murphy, R., Burke, J., and Goldiez, B. A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In *Proceedings of ICRA 2007, the IEEE International Conference on Robotics and Automation* (2007), pp. 852 – 857.

[79] Crowther, B., and Rivier, X. Flocking of Autonomous Unmanned Air Vehicles. In *Proceedings of the 17th Bristol UAV Conference* (2002).

[80] Crowther, W. Rule-Based Guidance for Flight Vehicle Flocking. In *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* (2004), vol. 218, pp. 111–124.

[81] Cutler, M., McLain, T., Beard, R., and Capozzi, B. Energy Harvesting and Mission Effectiveness for Small Unmanned Aircraft. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2010).

[82] Dalamagkidis, K., Valavanis, K., and Piegl, L. Evaluating the Risk of Unmanned Aircraft Ground Impacts. In *Proceedings of the 16th Mediterranean Conference on Control and Automation* (2008), pp. 709–716.

[83] Darwin, C. *On the Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life.* John Murray, London, UK, 1859.

[84] Dasgupta, P. A Multiagent Swarming System for Distributed Automatic Target Recognition Using Unmanned Aerial Vehicles. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 38*, 3 (2008), 549–563.

[85] Davenport, W. W. *Gyro!: The Life and Times of Lawrence Sperry*. Scribner Publishing, New York, NY, USA, 1978.

[86] Davis, L. Military Significance of Draper's Work for the Air Force. In *Air Space and Instruments*, S. Lees, Ed. McGraw-Hill, New York, NY, USA, 1963.

[87] De Jong, K. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, MI, USA, 1975.

[88] De Marqui, C., Erturk, A., and Inman, D. Finite Element Analysis of a UAV Wing Spar with Piezoceramics for Vibration Energy Harvesting. In *Proceedings of the 50th AIAA/ASME/ASCE/AHS/ASC Structure, Structural Dynamics, and Materials Conference* (2009).

[89] De Nardi, R., Holland, O., Woods, J., and Clark, A. SwarMAV: A Swarm of Miniature Aerial Vehicles. In *Proceedings of the 21st Bristol UAV Systems Conference* (2006).

[90] DeLuca, A. Experimental Investigation into the Aerodynamic Performance of both Rigid and Flexible Wing Structured Micro-Air-Vehicles. Master's thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, OH, USA, 2004.

[91] Deming, R., Perlovsky, L., and Brockett, R. Sensor Fusion for Swarms of Unmanned Aerial Vehicles using Modeling Field Theory. In *Proceedings of KIMAS 2005, the IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems* (2005), pp. 122–127.

[92] Deng, X., Schenato, L., and Sastry, S. Model Identification and Attitude Control for a Micromechanical Flying Insect Including Thorax and Sensor Models. In *Proceedings of ICRA 2003, the IEEE International Conference on Robotics & Automation* (2003), pp. 1152–1157.

[93] Dillingham, G. L. Unmanned Aircraft Systems. Federal Actions Needed to Ensure Safety and Expand Their Potential Uses within the National Airspace System. Tech. rep., GAO, United States Government Accountability Office, 2008.

[94] Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E., and Wiklund, J. The WITAS Unmanned Aerial Vehicle Project. In *Proceedings of ECAI 2000, the 14th European Conference on Artificial Intelligence* (2000), pp. 747–755.

[95] Dong, M., and Sun, Z. A Behavior-based Architecture for Unmanned Aerial Vehicles. In *Proceedings of the 2004 IEEE International Symposium on Intelligent Control* (2004), pp. 149–155.

[96] Doyle, J. C., Francis, B. A., and Tannenbaum, A. R. *Feedback Control Theory*. Dover Publications, New York, NY, USA, 2009.

[97] Drake, S. Converting GPS Coordinates to Navigation Coordinates (ENU). http://dspace.dsto.defence.gov.au/dspace/bitstream/1947/3538/1/DSTO-TN-0432.pdf, 2002.

[98] Duranti, S., Conte, G., Lundstrom, D., Rudol, P., Wzorek, M., and Doherty, P. LinkMAV, a Prototype Rotary Wing Micro Aerial Vehicle. In *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace* (2007).

[99] Eiben, A., Haasdijk, E., and Bredeche, N. Embodied, On-line, On-board Evolution for Autonomous Robotics. In *Symbiotic Multi-Robot Organisms*, P. Levi and S. Kernbach, Eds. Springer-Verlag, Berlin Heideberg, Germany, 2010, pp. 367–388.

[100] Eisenbeiss, H. A Mini Unmanned Aerial Vehicle (UAV): System Overview and Image Acquisition. *International Archives of Photogrammetry. Remote Sensing and Spatial Information Sciences 36*, 5/W1 (2004).

[101] Elman, J. Finding Structure in Time. *Cognitive Science 14* (1990), 179–211.

[102] Eng, P., Mejias, L., Walker, R., and Fitzgerald, D. Simulation of a Fixed-wing UAV Forced Landing with Dynamic Path Planning. In *Proceedings of the Australasian Conference on Robotics and Automation* (2007).

[103] Epstein, A. Millimeter-Scale MEMS Gas Turbine Engines. In *Proceedings of ASME Turbo Expo 2003, Power for Land, Sea, and Air* (2003).

[104] Etkin, B., and Reid, L. *Dynamics of Flight. Stability and Control*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[105] Everaerts, J. The Use of Unmanned Aerial Vehicles (UAVs) for Remote Sensing and Mapping. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2008), 1187–1192.

[106] Fenton, L. The Sum of Log-Normal Probability Distributions in Scatter Transmission Systems. *IRE Transactions on Communication Systems* (1960), 57–67.

[107] Ficici, S., Watson, R., and Pollack, J. Embodied Evolution: A Response to Challenges in Evolutionary Robotics. In *Proceedings of the Eighth European Workshop on Learning Robots* (1999), pp. 14–22.

[108] Fikes, R., and Nilsson, N. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2*, 3-4 (1971), 189–208.

[109] Filliat, D., Kodjabachian, J., and Meyer, J.-A. Incremental Evolution of Neural Controllers for Navigation in a 6-legged Robot. *Proceedings of the Fourth International Symposium on Artificial Life and Robotics* (1999).

[110] Fletcher, B. Autonomous Vehicles and the Net-Centric Battlespace. *Proceedings of the 2000 International UUV Symposium* (2000).

[111] Floreano, D., Hauert, S., Leven, S., and Zufferey, J.-C. Evolutionary Swarms of Flying Robots. In *Proceedings of the International Symposium on Flying Insects and Robots* (2007).

[112] Floreano, D., Husbands, P., and Nolfi, S. Evolutionary Robotics. In *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer-Verlag, Berlin, Germany, 2008.

[113] Floreano, D., and Keller, L. Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection. *PLoS Biology 8*, 1 (2010), 1–8.

[114] Foch, R. SENDER - A Low Cost Airobotic Platform. In *Proceedings of the AUVSI 2006 conference* (2006).

[115] Foch, R., and Ailinger, K. Low Reynolds Number Long Endurance Airplane Design. AIAA Paper No. 92-1263, American Institute of Aeronautics and Astronautics, 1992.

[116] Foch, R., Dahlburg, J., McMains, J., Bovais, C., and Carruthers, S. Dragon Eye, an Airborne Sensor System for Small Units. In *Proceedings of the Unmanned Systems conference* (2000).

[117] Fogel, D. On the Philosophical Differences between Evolutionary Algorithms and Genetic Algorithms. In *Proceedings of the 2nd Annual Conference on Evolutionary Programming* (1993), pp. 23–29.

[118] Fogel, D., and Fogel, L. An Introduction to Evolutionary Programming. In *Proceedings of AE'95, Artificial Evolution: European Conference* (1996).

[119] Fogel, D. B. Review of Computational Intelligence: Imitating Life. *Proceedings of the IEEE 83*, 11 (1995), 1588–1592.

[120] Fogel, L. J., Owens, A. J., and Walsh, M. J. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, NY, USA, 1966.

[121] Forrest, S. Scaling Fitnesses in the Genetic Algorithm. Documentation for PRISONERS DILEMMA and NORMS Programs that Use the Genetic Algorithm (unpublished manuscript), 1985.

[122] Freed, M., Fitzgerald, W., and Harris, R. Intelligent Autonomous Surveillance of Many Targets with Few UAVs. *IEEE Transactions on Control Systems Technology* (2005).

[123] Freed, M., Harris, R., and Shafto, M. Human vs. Autonomous Control of UAV Surveillance. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2004).

[124] Freeman, J., and Skapura, D. *Neural Networks. Algorithms, Applications, and Programming Techniques*. Computation and Neural Systems. Addison-Wesley Publishing Company, Boston, MA, USA, 1991.

[125] Freeman, R., and Biro, D. Modelling Group Navigation: Dominance and Democracy in Homing Pigeons. *The Journal of Navigation*, 62 (2009), 33–40.

[126] Frew, E., McGee, T., Kim, Z., Xiao, X., Jackson, S., Morimoto, M., Rathinam, S., Padial, J., and Sengupta, R. Vision-Based Road-Following Using a Small Autonomous Aircraft. In *Proceedings of the 2004 IEEE Aerospace Conference* (2004), pp. 3006–3015.

[127] Gacy, M., and Dahn, D. Commonality of Control Paradigms for Unmanned Systems. In *Proceedings of HRI'06, the 1st Annual Conference on Human-Robot Interaction* (2006), pp. 339–340.

[128] Gancet, J., Hattenberger, G., Alami, R., and Lacroix, S. Task Planning and Control for a Multi-UAV System: Architecture and Algorithms. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems* (2005), pp. 1017–1022.

[129] Garfinkel, S. L., Juels, A., and Pappu, R. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security & Privacy* (2005), 34–43.

[130] Gaudiano, P., Bonabeau, E., and Shargel, B. Evolving Behaviors for a Swarm of Unmanned Air Vehicles. In *Proceedings of the IEEE Swarm Intelligence Symposium* (2005), pp. 317–324.

[131] Gautier, F. Evolving Neural Network Using Genetic Algorithm In A Prey-Predator Scenario: Optimization By Threads. Master's thesis, University of Plymouth, Plymouth, UK, 2008.

[132] Gebre-Egziabher, D., Elkaim, G., Powell, J., and Parkinson, B. A Gyro-Free Quaternion-Based Attitude Determination System Suitable for Implementation Using Low Cost Sensors. In *Proceedings of the 2000 IEEE Position Location and Navigation Symposium* (2000), pp. 185–192.

[133] Geyer-Schulz, A. Holland Classifier Systems. In *Proceedings of APL '95, the International Conference on Applied Programming Languages* (1995), pp. 43–55.

[134] Ghosh, C. Application of Unmanned Combat Aerial Vehicles in Future Battles of the Subcontinent. *Strategic Analysis 25*, 4 (2001), 599–611.

[135] Gibson, J. *The Ecological Approach to Visual Perception.* Houghton, Mifflin and Company, Boston, MA, USA, 1979.

[136] Gilbert, E. Gray Codes and Paths on the n-Cube. *The Bell System Technical Journal* (1958), 815–826.

[137] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Boston, MA, USA, 1989.

[138] Goldberg, D. E. Genetic and Evolutionary Algorithms Come of Age. *Communications of the ACM 37*, 3 (1994), 113–119.

[139] Goldberg, D. E., and Deb, K. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms*, B. Spatz, Ed. Morgan Kaufmann Publishers, Inc., Burlington, MA, USA, 1991, pp. 69–93.

[140] Gomez, F., and Miikkulainen, R. Incremental Evolution of Complex General Behavior. *Adaptive Behavior*, 5 (1997), 317–342.

[141] Gomi, T., and Griffith, A. Evolutionary Robotics - An Overview. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation* (1996), pp. 40–49.

[142] Gould, S. J. *The Structure of Evolutionary Theory.* Belknap Press, Cambridge, MA, USA, 2002.

[143] Gould, S. J., and Eldredge, N. Punctuated Equilibria: the Tempo and Mode of Evolution Reconsidered. *Paleobiology 3*, 2 (1977), 115–151.

[144] Granlund, G., Nordberg, K., Wilklund, J., Doherty, P., Skarman, E., and Sandewall, E. WITAS: An Intelligent Autonomous Aircraft Using Active Vision. In *Proceedings of the UAV 2000 International Technical Conference and Exhibition* (2000).

[145] Grasmeyer, J., and Keennon, M. Development of the Black Widow Micro Air Vehicle. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2001), pp. AIAA Paper 2001–0127.

[146] Grefenstette, J. J. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics SMC-16*, 1 (1986), 122–128.

[147] Gurney, K. *An Introduction to Neural Networks.* UCL Press, London, UK, 1997.

[148] Hancock, P. J. B. An Empirical Comparison of Selection Methods in Evolutionary Algorithms. In *Evolutionary Computing: AISB Workshop*, T. Fogarty, Ed. Springer-Verlag, Berlin, Germany, 1994.

[149] Harman, L., Shama, U., Dand, K., and Kidwell, B. Remote Sensing and Spatial Information for Transportation Demand Management (TDM) Assessment. In *Proceedings of Pecora 15/Land Satellite Information IV/ISPRS Commission I/FIEOS 2002 Conference* (2002).

[150] Hartman, C., and Benes, B. Autonomous Boids. *Computer Animation and Virtual Worlds 17* (2006), 199–206.

[151] Harvey, I. Artificial Evolution: A Continuing SAGA. In *ER '01: Proceedings of the International Symposium on Evolutionary Robotics. From Intelligent Robotics to Artificial Life* (2001), vol. 2217, pp. 94–109.

[152] Harvey, I., Husbands, P., and Cliff, D. Issues in Evolutionary Robotics. In *From Animals to Animats 2. Proceedings of SAB92, the Second International Conference on Simulation of Adaptive Behavior* (1993).

[153] Harvey, I., Husbands, P., and Cliff, D. Seeing the Light: Artificial Evolution, Real Vision. In *From Animals to Animats 3. Proceedings of SAB94, the 3rd International Conference on Simulation of Adaptive Behaviour* (1994), pp. 392–401.

[154] Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. Evolutionary Robotics: the Sussex Approach. *Robotics and Autonomous Systems 20* (1996), 205–224.

[155] Harvey, I., Paolo, E. D., Tuci, E., Wood, R., and Quinn, M. Evolutionary Robotics: A New Scientific Tool for Studying Cognition. *Artificial Life 11*, 1-2 (2005), 79–98.

[156] Hauert, S. *Evolutionary Synthesis of Communication-based Aerial Swarms.* PhD thesis, EPFL, Lausanne, Switzerland, 2010.

[157] Hauert, S., Leven, S., Varga, M., Ruini, F., Cangelosi, A., Zufferey, J.-C., and Floreano, D. Reynolds Flocking in Reality with Fixed-Wing Robots: Communication Range vs. Flight Dynamics. In *Proceedings of IROS 2011, the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2011), pp. 5015–5020.

[158] Hauert, S., Leven, S., Zufferey, J.-C., and Floreano, D. Communication-based Swarming for Flying Robots. In *Proceedings of ICRA 2010, the IEEE International Conference on Robotics and Automation* (2010).

[159] Hauert, S., Zufferey, J.-C., and Floreano, D. Evolved Swarming without Positioning Information: an Application in Aerial Communication Relay. *Autonomous Robots 26*, 1 (2009), 21–32.

[160] Hauert, S., Zufferey, J.-C., and Floreano, D. Reverse-Engineering of Artificially Evolved Controllers for Swarms of Robots. In *Proceedings of CEC 2009, the IEEE Congress on Evolutionary Computation* (2009), pp. 55–61.

[161] Hebb, D. *The Organization Of Behavior.* Wiley & Sons, Hoboken, NJ, USA, 1949.

[162] Hermans, D., and Decuypere, R. A Challenge for Micro and Mini UAV: The Sensor Problem. In *Advanced Sensory Payloads for UAV, Meeting Proceedings RTO-MP-SET-092* (2005), pp. 13.1–13.8.

[163] Herwitz, S., Johnson, L., Arvesen, J., Higgins, R., Leung, J., and Dunagan, S. Precision Agriculture as a Commercial Application for Solar-Powered Unmanned Aerial Vehicles. In *Proceedings of AIAA's 1st Technical Conference and Workshop on Unmanned Aerospace Vehicles, Systems, Technologies, and Operations* (2002).

[164] Hing, J., and Oh, P. Development of an Unmanned Aerial Vehicle Piloting System with Integrated Motion Cueing for Training and Pilot Evaluation. *Journal of Intelligent & Robotic Systems 54*, 1-3 (2009), 3–19.

[165] Hodgins, J., and Brogan, D. Robot Herds: Group Behaviors for Systems with Significant Dynamics. In *Proceedings of ALIFE IV, the 4th International Workshop on the Synthesis and Simulation of Living Systems* (1994), pp. 319–324.

[166] Holland, J. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[167] Hopfield, J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *Proceedings of the National Academy of Sciences* (1982), vol. 79, pp. 2554–2558.

[168] Horan, B. The Statistical Character of Evolutionary Theory. *Philosophy of Science 61*, 1 (1994), 76–95.

[169] How, J., King, E., and Kuwata, Y. Flight Demonstrations of Cooperative Control for UAV Teams. In *Proceedings of the AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit* (2004).

[170] Howard, R., and Kaminer, I. Survey of Unmanned Air Vehicles. In *Proceedings of the American Control Conference* (1995), pp. 2950–2953.

[171] Hui, S., and Zak, S. The Widrow-Hoff Algorithm For McCulloch-Pitts Type Neurons. *IEEE Transactions on Neural Networks 5*, 6 (1994), 924–929.

[172] Hundley, R., and Gritton, E. Future Technology-Driven Revolutions in Military Operations. Results of a Workshop. RAND Documented Briefing No. DB-1100ARPA, RAND Corporation, 1994.

[173] Husbands, P., and Harvey, I. Evolution versus Design: Controlling Autonomous Robots. In *Proceedings of the 3rd Annual Conference on Artificial Intelligence, Simulation and Planning* (1992), pp. 139–146.

[174] Hussain, T., Montana, D., and Vidaver, G. Evolution-Based Deliberative Planning for Cooperating Unmanned Ground Vehicles in a Dynamic Environment. In *Proceedings of GECCO 2004, the Genetic and Evolutionary Computation Conference* (2004), pp. 1185–1196.

[175] Ifju, P., Jenkins, D., Ettinger, S., Lian, Y., Shyy, W., and Waszak, M. Flexible-Wing-Based Micro Air Vehicles. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2002).

[176] Ishii, K., Fujii, T., and Ura, T. Neural Network System for Online Controller Adaptation and its Application to Underwater Robot. In *Proceedings of ICRA 1998, the IEEE International Conference on Robotics & Automation* (1998), pp. 756–761.

[177] Jacobs, O. L. R. *Introduction to Control Theory.* Oxford University Press, Oxford, UK, 1993.

[178] Jager, R. Flight Test and Evaluation of the Piccolo II Autopilot System for Use on a One-Third Scale Yak-54. Master's thesis, Auburn University, AL, USA, 2008.

[179] Jakobi, N. *Minimal Simulations For Evolutionary Robotics.* PhD thesis, University of Sussex, UK, 1998.

[180] Jakobi, N., Husbands, P., and Harvey, I. Noise and the Reality Gap: the Use of Simulation in Evolutionary Robotics. In *Advances in Artificial Life: Proceedings of the 3rd European Conference on Artificial Life* (1995), pp. 704–720.

[181] Jang, J., and Liccardo, D. Small UAV Automation Using MEMS. *IEEE A&E Systems Magazine* (2007), 30–34.

[182] Jordan, M. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (1986), pp. 531–546.

[183] Juan, L., Zixing, C., and Jianqin, L. Premature Convergence in Genetic Algorithm: Analysis and Prevention Based on Chaos Operator. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation* (2000), pp. 495–499.

[184] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering 82*, Series D (1960), 35–45.

[185] Kaplan, F., and Oudeyer, P.-Y. Trends in Epigenetic Robotics: Atlas 2006. In *Proceedings of the Sixth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems* (2006).

[186] Kellog, J., Bovais, C., Dahlburg, J., Foch, R., Gardner, J., Gordon, D., Hartley, R., Kamgar-Parsi, B., McFarlane, H., Pipitone, F., Ramamurti, R., Sciambi, A., Spears, W., Srull, D., and Sullivan, C. The NRL MITE Air Vehicle. In *Proceedings of the 16th International Conference on Unmanned Air Vehicle Systems* (2001), pp. 25.1–25.14.

[187] Kenneth, P. *The Evolution of the Cruise Missile.* Air University Press, Maxwell, AL, USA, 1985.

[188] Khatib, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Proceedings of ICRA 1985, the IEEE International Conference on Robotics and Automation* (1985), pp. 500–505.

[189] Kim, J.-H., Wishart, S., and Sukkarieh, S. Real-time Navigation, Guidance, and Control of a UAV using Low-cost Sensors. In *Proceedings of FSR 2003, the International Conference on Field and Service Robotics* (2003), pp. 95–100.

[190] King, E., Kuwata, Y., Alighanbari, M., and Bertuccelli, L. J. Coordination and Control Experiments on a Multi-vehicle Testbed. In *Proceedings of the 2004 American Control Conference* (2004), pp. 5315–5320.

[191] Kingston, D., and Beard, R. Real-Time Attitude and Position Estimation for Small UAVs Using Low-Cost Sensors. In *Proceedings of the AIAA 3rd Unmanned Unlimited Systems Conference and Workshop* (2004).

[192] Koestler, A., and Braunl, T. Mobile Robot Simulation with Realistic Error Models. In *Proceedings of the 2nd International Conference on Autonomous Robots and Agents* (2004), pp. 46–51.

[193] Kontitsis, M., Valavanis, K., and Tsourveloudis, N. A UAV Vision System for Airborne Surveillance. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation* (2004), pp. 77–83.

[194] Koos, S., Mouret, J.-B., and Doncieux, S. Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers. *Proceedings of GECCO '12, the 12th annual Conference on Genetic and Evolutionary Computation* (2010), 119–126.

[195] Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[196] Koza, J. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA, 1994.

[197] Koza, J. R., and Poli, R. Genetic Programming. In *Search Methodologies*, E. K. Burke and G. Kendall, Eds. Springer US, New York, NY, USA, 2005, pp. 127–164.

[198] Kube, C. R. Task Modelling in Collective Robotics. *Autonomous Robots 4* (1997), 53–72. 10.1023/A:1008859119831.

[199] Kunz, H., and Hemelrjik, C. Artificial Fish Schools: Collective Effects of School Size, Body Size and Body Form. *Artificial Life 9*, 3 (2003), 237–253.

[200] Kutschera, U., and Niklas, K. The Modern Theory of Biological Evolution: an Expanded Synthesis. *Naturwissenschaften 91*, 6 (2004), 255–276.

[201] Langton, C. Studying Artificial Life with Cellular Automata. *Physica D: Nonlinear Phenomena 22*, 1 (1986), 120–149.

[202] Langton, C. *Artificial life: An overview*. MIT Press, Cambridge, MA, USA, 1997.

[203] Latchman, H., Wong, T., and Courage, K. Statement of Work for Airborne Traffic Surveillance Systems. Proof of Concept Study for Florida Department of Transportation, University of Florida, FL, USA, 2002.

[204] Lawrence, D., Donahue, R., Mohseni, K., and Han, R. Information Energy for Sensor-Reactive UAV Flock Control. In *Proceedings of the 3rd AIAA "Unmanned Unlimited" Technical Conference, Workshop and Exhibit* (2004).

[205] Lax, M., and Sutherland, B. An Extended Role for Unmanned Aerial Vehicles in the Royal Australian Air Force. Tech. Rep. Paper 46, Royal Australian Air Force, Air Power Studies Centre, 1996.

[206] Lee, C. Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I. *IEEE Transactions on Systems, Man, and Cybernetics 20*, 2 (1990), 404–418.

[207] Lee, C. Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part II. *IEEE Transactions on Systems, Man, and Cybernetics 20*, 2 (1990), 419–435.

[208] Lee, J., Huang, R., Vaughn, A., Xiao, X., Hedrick, J., Zennaro, M., and Sengupta, R. Strategies of Path-Planning for a UAV to Track a Ground Vehicle. In *Proceedings of AINS 2003, Second Annual Symposium on Autonomous Intelligent Networks and Systems* (2003).

[209] Lee-Johnson, C., and Carnegie, D. Mobile Robot Navigation Modulated by Artificial Emotions. *IEEE Transaction on Systems, Man, and Cybernetics 40*, 2 (2010), 469–480.

[210] Leven, S. *Enabling Large-Scale Collective Systems in Outdoor Aerial Robotics*. PhD thesis, EPFL, Lausanne, Switzerland, 2011.

[211] Leven, S., Zufferey, J.-C., and Floreano, D. A Simple and Robust Fixed-Wing Platform for Outdoor Flying Robot Experiments. In *Proceedings of the International Symposium on Flying Insects and Robots* (2007), pp. 69–70.

[212] Leven, S., Zufferey, J.-C., and Floreano, D. A Minimalist Control Strategy for Small UAVs. In *Proceedings of IROS 2009, the IEEE/RSJ International Conference on Intelligent RObots and Systems* (2009), pp. 2873–2878.

[213] Leven, S., Zufferey, J.-C., and Floreano, D. Dealing with Midair Collisions in Dense Collective Aerial Systems. *Journal of Field Robotics 28*, 3 (2011), 405–423.

[214] Lewis, F., Jagannathan, S., and Yesildirek, A. *Neural Network Control of Robot Manipulators and Nonlinear Systems*. London, UK. Taylor & Francis, 1999.

[215] Lima, J., Gracias, N., Pereira, H., and Rosa, A. Fitness Function Design for Genetic Algorithms in Cost Evaluation Based Problems. In *Proceedings of the 19996 IEEE International Conference on Evolutionary Computation* (2002), pp. 207–212.

[216] Lin, P.-H., and Lee, C.-S. The Eyewall-Penetration Reconnaissance Observation of Typhoon Longwang (2005) with Unmanned Aerial Vehicle, Aerosonde. *Journal of Atmospheric and Oceanic Technology 25*, 1 (2008), 15–25.

[217] Lin, P.-H., Lee, C.-S., Yen, T.-C., and Lee, H.-C. Fly into Typhoon Hayan with UAV Aerosonde. In *Proceedings of the 2000 American Meteorological Society Conference* (2000), vol. 52113.

[218] Lindgren, K. Evolutionary Phenomena in Simple Dynamics. In *Artificial Life II*, C. Langton, C. Taylor, D. Farmer, and S. Rasmussen, Eds. Addison-Wesley, Boston, MA, USA, 1991, pp. 295–312.

[219] Lipson, J. Evolutionary Robotics: Emergence of Communication. *Current Biology 17*, 9 (2007), R330–R332.

[220] Lobb, A., and Bangay, S. Realistic Autonomous Fish for Virtual Reality. In *Proceedings of AFRIGRAPH '03, the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa* (2002), pp. 167–170.

[221] Loch, C., and Huberman, B. A Punctuated-Equilibrium Model of Technology Diffusion. *Management Science 45*, 2 (1999), 160–177.

[222] Loh, R., Bian, Y., and Roe, T. UAVs in Civil Airspace: Safety Requirements. *IEEE Aerospace and Electronic Systems Magazine 24*, 1 (2009), 5–17.

[223] Lovtrup, S. The Four Theories of Evolution. Epilogue. *Rivista di Biologia / Biology Forum*, 92 (1982), 478–480.

[224] Lund, H. H., and Miglino, O. From Simulated to Real Robots. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation* (1996), pp. 362–365.

[225] Lyon, D. A Military Perspective on Small Unmanned Aerial Vehicles. *IEEE Instrumentation & Measurement Magazine*, September (2004), 27–31.

[226] Malthus, T. *An Essay on the Principle of Population. An Essay on the Principle of Population, as it Affects the Future Improvement of Society with Remarks on the Speculations of Mr. Godwin, M. Condorcet, and Other Writers.* J. Johnson, London, UK, 1798.

[227] Manai, M., Desbiens, A., and Gagnon, E. Identification of a UAV and Design of a Hardware-in-the-Loop System for Nonlinear Control Purposes. In *Proceedings of the 2005 AIAA Guidance, Navigation, and Control Conference and Exhibit* (2005).

[228] Mark, A., Polani, D., and Uthmann, T. A Framework for Sensor Evolution in a Population of Braitenberg Vehicle-like Agents. In *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life* (1998), pp. 428–432.

[229] Marocco, D. *Intelligenza Artificiale. Introduzione ai nuovi modelli.* Bonanno Editore, Acireale, Italy, 2006.

[230] Marocco, D., Cangelosi, A., and Nolfi, S. The Emergence of Communication in Evolutionary Robots. *Philosophical transactions Series A, Mathematical, physical, and engineering sciences 361*, 1811 (2003), 2397–2421.

[231] Maroney, D., Bolling, R., Heffron, M., and Flathers, G. Experimental Platforms for Evaluating Sensor Technology for UAS Collision Avoidance. In *Proceedings of the 26th Digital Avionics Systems Conference* (2007), pp. 5.C.1–1 – 5.C.1–9.

[232] Mataric, M. Issues and Approaches in Design of Collective Autonomous Agents. *Robotics and Autonomous Systems 16* (1994), 321–331.

[233] Mataric, M. *The Robotics Primer.* MIT Press, Cambridge, MA, USA, 2007.

[234] Maza, I., Caballero, F., Molina, R., Pena, N., and Oliero, A. Multimodal Interface Technologies for UAV Ground Control Stations. *Journal of Intelligent & Robotic Systems 57*, 1-4 (2010).

[235] McCarley, J., and Wickens, C. Human Factors Concerns in UAV flight. `http://www.hf.faa.gov/docs/508/docs/uavFY04Planrpt.pdf`, 2004.

[236] McCarthy, J. Artificial Intelligence, Logic and Formalizing Common Sense. *Philosophical Logic and Artificial Intelligence* (1989), 161–189.

[237] McClelland, J. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, 2nd ed. `http://www.stanford.edu/group/pdplab/pdphandbook/`, 2011.

[238] McCormack, E. The Use of Small Unmanned Aircraft by the Washington State Department of Transportations. TRAC Research Report - Agreement T4118, Task 04, Unmanned Aerial Vehicles, Washington State Transportation Center (TRAC), Washington, WA, USA, 2008.

[239] McCulloch, W., and Pitts, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–133.

[240] McGeer, T. LAIMA: The first Atlantic Crossing by Unmanned Aircraft. `http://www.aerovelco.com/papers/LaimaStory.pdf`, 1999.

[241] McLain, T. Coordinated Control of Unmanned Air Vehicles. Tech. rep., Wright-Patterson Air Force Base, OH, USA, 1999.

[242] McLean, D. *Automatic Flight Control Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 1990.

[243] Medler, D. A Brief History of Connectionism. *Neural Computing Surveys*, 1 (1998), 61–101.

[244] Melhuish, C., Welsby, J., and Greenway, P. Gradient Ascent with a group of Minimalist Real Robots: Implementing Secondary Swarming. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (2002), vol. 2, pp. 509–514.

[245] Mendes, M. D. B. 45 years of evolution strategies: Hans-Paul Schwefel interviewed for the genetic argonaut blog. `http://delivery.acm.org/10.1145/1820000/1810133/p2-mendes.pdf?ip=141.163.186.62&acc=ACTIVE%20SERVICE&CFID=53031044&CFTOKEN=85679215&__acm__=1320944646_f3279097809eef76c43911bb8f37e247`, 2009.

[246] Merino, L., Caballero, F., de Dios, J. M., Ferruz, J., and Ollero, A. A Cooperative Perception System for Multiple UAVs: Application to Automatic Detection of Forest Fires. *Journal of Field Robotics 23*, 3-4 (2006), 165–184.

[247] Metni, N., and Hamel, T. A UAV for Bridge Inspection: Visual Servoing Control Law with Orientation Limits. *Automation in Construction 17* (2007), 3–10.

[248] Metz, C. Use of Fiber Optic Data Links in Marine Corps Unmanned Vehicle. *Unmanned Systems 6*, 4 (1988), 13–17.

[249] Michelson, R. The Entomopter. In *Neurotechnology for Biomimetic Robots*, J. Ayers, J. Davis, and A. Rudolph, Eds. MIT Press, Cambridge, MA, USA, 2002, pp. 481–509.

[250] Michelson, R. Novel Approaches to Miniature Flight Platforms. In *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* (2004), vol. 18, pp. 363–373.

[251] Michelson, R. Test and Evaluation of Fully Autonomous Micro Air Vehicles. *ITEA Journal*, 29 (2008), 367–374.

[252] Michelson, R. Overview of Micro Air Vehicle System Design and Integration Issues. In *Encyclopedia of Aerospace Engineering*, R. Blockley and W. Shyy, Eds. John Wiley & Sons, New York, NY, USA, 2010.

[253] Miglino, O., Lund, H., and Nolfi, S. Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life 2*, 4 (1995), 417–434.

[254] Miller, J., Minear, P., Niessner, A., DeLullo, A., Geiger, B., Long, L., and Horn, J. Intelligent Unmanned Air Vehicle Flight Systems. In *Proceedings of the 2005 AIAA InfoTech@Aerospace Conference* (2005).

[255] Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry*, 2nd ed. MIT Press, Cambridge, MA, USA, 1972.

[256] Mirolli, M., and Parisi, D. Talking to Oneself as a Selective Pressure for the Emergence of Language. In *The Evolution of Language: Proceedings of the 6th International Conference on the Evolution of Language* (2006), pp. 214–221.

[257] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[258] Mitchell, M., Forrest, S., and Holland, J. The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (1992), pp. 242–254.

[259] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. The e-puck, a Robot Designed for Education in Engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions* (2009), pp. 59–65.

[260] Mondada, F., Franzi, E., and Guignard, A. The Development of Khepera. In *Proceedings of the 1st International Khepera workshop* (1999), pp. 7–13.

[261] Montambault, S., Beaudry, J., Toussaint, K., and Pouliout, N. On the Application of VTOL UAVs to the Inspections of Power Utility Assets. In *Proceedings of the 1st International Conference on Applied Robotics for the Power Industry* (2010).

[262] Montana, D., and Davis, L. Training Feedforward Neural Networks Using Genetic Algorithms. In *Proceedings of IJCAI'89, the 11th international joint conference on Artificial intelligence* (1989).

[263] Moore, F. A Methodology for Missile Countermeasures Optimization under Uncertainty. *Evolutionary Computation 10*, 2 (2002), 129–149.

[264] Mouret, J.-B., and Doncieux, S. Overcoming the Bootstrap Problem in Evolutionary Robotics Using Behavioral Diversity. In *Proceedings of CEC '09, the IEEE Congress on Evolutionary Computation* (2009), pp. 1161–1168.

[265] Mouret, J.-B., Doncieux, S., and Meyer, J.-A. Incremental Evolution of Target-Following Neuro-controllers for Flapping-Wing Animats. In *From Animals to Animats 9. Proceedings of SAB 2006, the 9th International Conference on Simulation of Adaptive Behavior* (2006), pp. 606–618.

[266] Mueller, T. Aerodynamic Measurements at Low Reynolds Numbers for Fixed Wing Micro-Air Vehicles. Tech. rep., University of Notre Dame, Notre Dame, IN, USA, 1999.

[267] Mueller, T. On the Birth of Micro Air Vehicles. *International Journal of Micro Air Vehicles 1*, 1 (2009), 1–12.

[268] Mueller, T., and DeLaurier, D. Aerodynamics of Small Vehicles. *Annual Review of Fluid Mechanics 35* (2003), 89–111.

[269] Muhlenbein, H. Evolution in Time and Space - The Parallel Genetic Algorithm. In *Foundations of Genetic Algorithms* (1991), pp. 316–337.

[270] Mullens, K., Pacis, E., Stancliff, S., Burmeister, A., Denewiller, T., Bruch, M., and Everett, H. An Automated UAV Mission System. In *Proceedings of USIS 03, the AUVSI conference on Unmanned Systems in International Security* (2003).

[271] Mullin, J. Tier II UAV. `http://hqinet001.hqmc.usmc.mil/pp&o/pog/GCE_Conf_Briefs/1-05/UAV%2520Tier%2520II%2520Brief%2520to%2520GCE%2520Conf%2520-%252022%2520Apr%252005.ppt`, 2005.

[272] Murphy, D., Bott, J., Bryan, W., Coleman, J., Gage, D., Nguyen, H., and Cheatham, M. MSSMP: No Place to Hide. In *Proceedings of the AUVSI'97 Conference* (1997).

[273] Murray, P. So What's New About Complexity? *The Journal of Systems Research and Behavioral Science 20*, 5 (2003), 409–417.

[274] Nagy, M., Akos, Z., Biro, D., and Vicsek, T. Hierarchical Group Dynamics in Pigeon Flocks. *Nature 464*, 7290 (2010), 890–893.

[275] Negenborn, R. *Robot Localization and Kalman Filters. On finding your position in a noisy world.* PhD thesis, Utrecht University, The Netherlands, 2003.

[276] Nelson, A. L., Barlow, G. J., and Doitsidis, L. Fitness Functions in Evolutionary Robotics: A Survey and Analysis. *Robotics and Autonomous Systems 57*, 4 (2009), 345–370.

[277] Nelson, A. L., Grant, E., Galeotti, J. M., and Rhody, S. Maze Exploration Behaviors Using an Integrated Evolutionary Robotics Environment. *Robotics and Autonomous Systems 46*, 3 (2004), 159–173.

[278] Nillson, N. Shakey the Robot. SRI International - Technical Note 323, Stanford Research Institute, Stanford, CA, USA, 1984.

[279] Nitschke, G. Emergence of Cooperation: State of the Art. *Artificial Life 11*, 3 (2005).

[280] Nolfi, S., and Floreano, D. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines*, 2nd ed. MIT Press, Cambridge, MA, USA, 2001.

[281] Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics. In *Artificial Life IV. Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (1994), pp. 190–197.

[282] Nolfi, S., and Parisi, D. Exploiting the Power of Sensory-Motor Coordination. In *Proceedings of ECAL'99, the 5th European Conference on Advances in Artificial Life* (1999), pp. 173–182.

[283] Ollero, A., and Merino, L. Control and Perception Techniques for Aerial Robotics. *Annual Reviews in Control 28* (2004), 167–178.

[284] Padfield, G., and Lawrence, B. The Birth of Flight Control: An Engineering Analysis of the Wright Brothers' 1902 Glider. *The Aeronautical Journal*, December (2003), 697–718.

[285] Pagello, E., D'Angelo, A., Montesello, F., Garelli, F., and Ferrari, C. Cooperative Behaviors in Multi-Robot Systems through Implicit Communication. *Robotics and Autonomous Systems 29* (1999), 65–77.

[286] Palmer, R., Arthur, W., Holland, J., LeBaron, B., and Tayler, P. Artificial Economic Life: A Simple Model of a Stockmarket. *Physica D: Nonlinear Phenomena 75*, 1-3 (1994), 264–274.

[287] Pamphile, T., and Lin, K.-C. Behavior-Based Control Hierarchy of Unmanned Aerial Vehicle Swarming. In *Proceedings of CTS 2006, the International Symposium on Collaborative Technologies and Systems* (2006), pp. 349–355.

[288] Parisi, D. *Mente: i nuovi modelli della Vita Artificiale*. Il Mulino, Bologna, Italy, 1999.

[289] Pashilkar, A., Sundararajan, N., and Saratchandran, P. A Fault-Tolerant Neural Aided Controller for Aircraft Auto-Landing. *Aerospace Science and Technology*, 10 (2006), 49–61.

[290] Patcher, M., and Chandler, P. Challenges of Autonomous Control. *IEEE Control Systems* (1998), 92–97.

[291] Patterson, M., Mulligan, A., Robinson, D., Wardell, L., and Pallister, J. Volcano Surveillance by ACR Silver Fox. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2005).

[292] Pavlov, I. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press, Oxford, UK, 1927.

[293] Peniak, M., Bentley, B., Marocco, D., Cangelosi, A., Ampatzis, C., Izzo, D., and Biscani, F. An Evolutionary Approach to Designing Autonomous Planetary Rovers. In *Proceedings of TAROS 2010, the 11th Conference Towards Autonomous Robotic Systems* (2010), pp. 198–204.

[294] Perlovsky, L. *Neural Networks and Intellect: using model based concepts*. Oxford University Press, Oxford, UK, 2001.

[295] Petrovic, P. Overview of Incremental Approaches to Evolutionary Robotics. In *Proceedings of the 1999 Norwegian Conference on Computer Science* (1999), pp. 151–162.

[296] Petrovic, P. A Step towards Incremental On-Board Evolutionary Robotics. In *Proceedings of SCAI'01, the Seventh Scandinavian Conference on Artificial Intelligence* (2001), pp. 3–12.

[297] Pfeifer, R., and Scheier, C. *Understanding intelligence*. MIT Press Cambridge, MA, USA, 1999.

[298] Pirjanian, P., and Mataric, M. Multi-Robot Target Acquisition Using Multiple Objective Behavior Coordination. In *Proceedings of ICRA 2000, the IEEE International Conference on Robotics and Automation* (2000), vol. 3, pp. 2696–2702.

[299] Purcell, E. Life at low Reynolds number. *American Journal of Physics 45*, 1 (1977), 3–11.

[300] Puri, A. A Survey of Unmanned Aerial Vehicles (UAVs) for Traffic Surveillance. Tech. rep., University of South Florida, Tampa, FL, USA, 2005.

[301] Qu, Y.-H., Pan, Q., and Yan, J.-G. Flight Path Planning of UAV Based on Heuristically Search and Genetic Algorithms. In *Proceedings of IECON 2005, the 31st Annual Conference of IEEE Industrial Electronics Society* (2005), pp. 45–49.

[302] Quigley, M., Goodrich, M., and Beard, R. Semi-Autonomous Human-UAV Interfaces for Fixed-Wing Mini-UAVs. In *Proceedings of ICIRS 2004, the IEEE/RSJ International Conference on Intelligent Robots and Systems* (2004), pp. 2457–2462.

[303] Quigley, M., Goodrich, M., Griffiths, S., Eldredge, A., and Beard, R. Target Acquisition, Localization, and Surveillance Using a Fixed-Wing Mini-UAV and Gimbaled Camera. In *Proceedings of ICRA 2005, the IEEE International Conference on Robotics and Automation* (2005), pp. 2600–2605.

[304] Rango, A., Laliberte, A., Steele, C., Herrick, J., Bestelmeyer, B., Schmugge, T., Roanhorse, A., and Jenkins, V. Using Unmanned Aerial Vehicles for Rangelands: Current Applications and Future Potentials. *Environmental Practice 8* (2006), 159–168.

[305] Rathbun, D., Kragelund, S., Pongpunwattana, A., and Capozzi, B. An Evolution Based Path Planning Algorithm for Autonomous Motion of a UAV through Uncertain Environments. In *Proceedings of the AIAA Digital Avionics Systems Conference* (2002), pp. 608–619.

[306] Rathinam, S., Zennaro, M., Mak, T., and Sengupta, R. An Architecture for UAV Team Control. In *Proceedings of IAV 2004, the 5th IFAC Symposium on Intelligent Autonomous Vehicles* (2004), pp. 1721–1728.

[307] Ray, T. An Evolutionary Approach to Synthetic Biology: Zen and the Art of Creating Life. *Artificial Life 1*, 1/2 (1994), 195–226.

[308] Rechenberg, I. Cybernetic Solution Path of an Experimental Problem. In *Evolutionary Computation. The Fossil Record*, D. Fogel, Ed. IEEE Press, Piscataway, NJ, USA, 1965.

[309] Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, Germany, 1973.

[310] Reil, T., and Husbands, P. Evolution of Central Pattern Generators for Bipedal Walking in a Real-Time Physics Environment. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 159–168.

[311] Reynolds, C. Flocks, Herds, and Schools: A Distributed Behavioral Model. In *Proceedings of ACM SIGGRAPH '87* (1987), vol. 21, pp. 25–34.

[312] Reynolds, C. Steering Behaviors for Autonomous Characters. In *Proceedings of the Game Developers Conference* (1999), pp. 763–782.

[313] Richards, M., Whitley, D., and Beveridge, J. Evolving Cooperative Strategies for UAV Teams. In *Proceedings of GECCO 2005, Genetic and Evolutionary Computation Conference* (2005), pp. 332–339.

[314] Rocha, M., and Neves, J. Preventing Premature Convergence to Local Optima in Genetic Algorithms via Random Offspring Generation. *Lecture Notes in Computer Science 1611* (1999), 127–136.

[315] Romeo, G., Frulla, G., and Cestino, E. Design of a High-Altitude Long-Endurance Solar-Powered Unmanned Air Vehicle for Multi-Payload and Operations. In *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* (2007), vol. 221, pp. 199–216.

[316] Rosenblatt, F. The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review 65*, 6 (1958), 386–408.

[317] Rosenblatt, F. Perceptron Simulation Experiments. In *Proceedings of the IRE* (1960), pp. 301–309.

[318] Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, vol. 115. Spartan Books, Washington, WA, USA, 1962.

[319] Ruini, F., and Cangelosi, A. Distributed Control in Multi-Agent Systems: A Preliminary Model of Autonomous MAV Swarms. In *Proceedings of FUSION 2008, the International Conference on Information Fusion* (2008), pp. 1043–1050.

[320] Ruini, F., and Cangelosi, A. Evolutionary Algorithm Based Neural Network Controllers: an Application to MAV Swarms. In *Proceedings of WIVACE 2008, Italian Conference on Artificial Life and Evolutionary Computation* (2008).

[321] Ruini, F., and Cangelosi, A. Extending the Evolutionary Robotics Approach to Flying Machines: An Application to MAV Teams. *Neural Networks*, 22 (2009), 812–821.

[322] Ruini, F., and Cangelosi, A. Un Modello 3D di Robotica Evolutiva per lo Sviluppo di Controller Autonomi per Robot Volanti. In *Modelli, Sistemi e Applicazioni di Vita Artificiale e Computazione Evolutiva. Atti del VI Workshop Italiano di Vita Artificiale e Computazione Evolutiva (WIVACE 2009)* (Naples, Italy, 2009), O. Miglino, M. Ponticorvo, A. Rega, and F. Rubinacci, Eds., FEU - Fridericiana Edizioni Universitarie, pp. 177–185.

[323] Ruini, F., and Cangelosi, A. An Evolutionary Robotics 3D Model for Autonomous MAVs Navigation, Target Tracking and Group Coordination. In *Proceedings of IJCNN 2010, the International Joint Conference on Neural Networks* (2010), pp. 760–767.

[324] Ruini, F., and Cangelosi, A. An Incremental Approach to the Evolutionary Design of Autonomous Controllers for Micro-unmanned Aerial Vehicles. In *Proceedings of TAROS 2010, the 11th Conference Towards Autonomous Robotic Systems* (2010), pp. 239–246.

[325] Ruini, F., and Cangelosi, A. Intelligent Autonomous Controllers Based on Genetically Evolved Neural Networks for Flying Robots: Experiments in Two and Three Dimensions. In *Proceedings of PCCAT 2010, the Postgraduate Conference for Computing: Application and Theory* (2010).

[326] Ruini, F., Cangelosi, A., and Zetule, F. Individual and Cooperative Tasks Performed by Autonomous MAV Teams Driven by Embodied Neural Network Controllers. In *Proceedings of IJCNN 2009, the International Joint Conference on Neural Networks* (2009), pp. 2717–2724.

[327] Ruini, F., Petrosino, G., Saglimbeni, F., and Parisi, D. The Strategic Level and the Tactical Level of Behaviour. In *Advances in Cognitive Systems*, J. Gray and S. Nefti-Meziani, Eds. IET Publisher, Herts, UK, 2010, pp. 271–299.

[328] Rumelhart, D., Hinton, G., and Williams, R. Learning Representations by Back-Propagating Errors. *Nature 323* (1986), 533–536.

[329] Rundle, J., Klein, W., and Turcotte, D., Eds. *Reduction and Predictability of Natural Disasters*. Santa Fe Institute Series. Westview Press, Boulder, CO, 1996.

[330] Ryan, A., Xiao, X., Rahinam, S., Tisdale, J., Zennaro, M., Caveney, D., Sengupta, R., and Hedrick, J. A Modular Software Infrastructure for Distributed Control of Collaborating UAVs. In *Proceedings of the AIAA Conference on Guidance, Navigation, and Control* (2006).

[331] Salomon, R. Evolving and Optimizing Braitenberg Vehicles by Means of Evolution Strategies. *International Journal of Smart Engineering Systems Design 2* (1999), 69–77.

[332] Sandini, G., Metta, G., and Vernon, D. RobotCub: An Open Framework for Research in Embodied Cognition. In *Proceedings of the 4th IEEE/RAS International Conference on Humanoid robots* (2004), pp. 13–32.

[333] Sandini, G., Metta, G., and Vernon, D. The iCub Cognitive Humanoid Robot: an Open-System Research Platform for Enactive Cognition. *Lectures Notes in Computer Science 4850* (2007), 358–369.

[334] Schaffer, J., Whitley, D., and Eshelman, L. Combinations of Genetic Algorithms and Neural Networks: a Survey of the State of the Art. In *Proceedings of COGANN-92, the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (1992).

[335] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In *Proceedings of the Third International Conference on Genetic Algorithms* (Burlington, MA, USA, 1989), Morgan Kaufmann Publishers, Inc., pp. 51–60.

[336] Scheck, W. Lawrence Sperry: Genius on Autopilot. *Aviation History Magazine* (November 2004).

[337] Scherer, S., Singh, S., Chamberlain, L., and Elgersma, M. Flying Fast and Low Among Obstacles: Methodology and Experiments. *The International Journal of Robotics Research 27*, 5 (2008), 549–574.

[338] Schlect, J., Altenburg, K., Ahmed, B., and Nygard, K. Decentralized Search by Unmanned Air Vehicles using Local Communication. In *Proceedings of IC-AI 2003, the International Conference on Artificial Intelligence* (2003), pp. 757–762.

[339] Schlesinger, M., and Parisi, D. The Agent-Based Approach: A New Direction for Computational Models of Development. *Developmental Review 21*, 1 (2001), 121–146.

[340] Schlesinger, M., and Parisi, D. Connectionism in an Artificial Life Perspective: Simulating Motor, Cognitive, and Language Development. In *Neuroconstructivism: Perspectives and Prospects*, D. Mareschal, S. Sirois, and G. Westermann, Eds. Oxford University Press, Oxford, UK, 2007.

[341] Schwefel, H. *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie*. Birkhäuser Basel, Stuttgart, Germany, 1977.

[342] Schwefel, H. P. *Evolutionsstrategie und Numerische Optimierung*. PhD thesis, Technical University of Berlin, Berlin, Germany, 1975.

[343] Serra, R., Carletti, T., and Poli, I. Synchronization Phenomena in Surface-Reaction Models of Protocells. *Artificial Life 13*, 2 (2007), 123–138.

[344] Servan-Schreiber, D., Cleeremans, A., and McClelland, J. Graded State Machines: The Representation of Temporal Contingencies in Simple Recurrent Networks. *Machine Learning*, 7 (1991), 161–193.

[345] Sharkey, N. Automated Killers and the Computing Profession. *IEEE Computer* (Nov. 2007), 122–124.

[346] Sharkey, N. Grounds for Discrimination: Autonomous Robot Weapons. *RUSI Defence Systems* (Oct. 2008), 86–89.

[347] Sharkey, N. Weapons of Indiscriminate Lethality. *FIfF-Kommunikation*, 1 (2009), 26–29.

[348] Sharkey, N. Saying 'No!' to Lethal Autonomous Targeting. *Journal of Military Ethics 9*, 4 (2010), 369–383.

[349] Shumaker, J., Ali, K., and Carter, L. A Gimbaled Platform for MAV Autopilot Simulation and Calibration. In *Proceedings of DASC 2008, the 27th IEEE/AIAA Digital Avionics Systems Conference* (2008), vol. 4.C4-1–4.C.4-10.

[350] Sinsley, G., Miller, J., Long, L., Geiger, B., Niessner, A., and Horn, J. An Intelligent Controller for Collaborative Unmanned Air Vehicles. In *Proceedings of CISDA 2007, the IEEE Symposium on Computational Intelligence in Security and Defense Applications* (2007), pp. 139–144.

[351] Song, Y.-E., and Yoon, K.-J. Development of a Small Air Robot with Fixed Wing and Semi-Autopilot System. In *Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics* (2007), pp. 1821–1826.

[352] Sparrow, R. Killer Robots. *Journal of Applied Philosophy 24*, 1 (2007), 62–77.

[353] Spencer, H. *The Principles of Biology*. D. Appleton and Company, New York, NY, USA, 1898.

[354] Srinivas, M., and Patnaik, L. Genetic Algorithms: a Survey. *Computer 27*, 6 (1994), 17–26.

[355] Stanley, K., and Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[356] Steele Jr., G. L. *Common LISP. The Language*, 2nd ed. Digital Press, Bournemouth, UK, 1990.

[357] Steels, L. The Artificial Life Roots of Artificial Intelligence. *Artificial Life Journal 1*, 1-2 (1994), 75–110.

[358] Sullivan, J. Impediments to and Incentives for Automation in the Air Force. In *Proceedings of the ISTAS 2005, the International Symposium on Technology and Society. Weapons and Wires: Prevention and Safety in a Time of Fear* (2005), pp. 102–110.

[359] Sullivan, J. Evolution or Revolution? The Rise of UAVs. *IEEE Technology and Society Magazine 25*, 3 (2006), 43–49.

[360] Sun, D., Wu, H., Zhu, R., and Hung, L. Development of Micro Air Vehicle Based on Aerodynamic Modeling Analysis in Tunnel Tests. In *Proceedings of ICRA 2005, the IEEE International Conference on Robotics and Automation* (2005), pp. 2235–2240.

[361] Taha, Z., Tang, Y., and Yap, K. Development of an Onboard System for Flight Data Collection of a Small-Scale UAV Helicopter. *Mechatronics 21*, 1 (2011), 132–144.

[362] Tennekes, H. *The Simple Science of Flight: from Insects to Jumbo Jets*. MIT Press, Cambridge, MA, USA, 2009.

[363] Theodore, C., and Tischler, M. Precision Autonomous Landing Adaptive Control Experiment (PALACE). In *Proceedings of the 25th Army Science Conference, "Transformational Army Science and Technology - Charting the Future of S&T for the Soldier"* (2006).

[364] Tikhanoff, V. *Development of Cognitive Capabilities in Humanoid Robots*. PhD thesis, University of Plymouth, UK, 2009.

[365] Togelius, J. Evolution of a Subsumption Architecture Neurocontroller. *Journal of Intelligent and Fuzzy Systems 15*, 1 (2004), 15–20.

[366] Togelius, J., and Lucas, S. M. Evolving Controllers for Simulated Car Racing. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), vol. 2, pp. 1906–1913.

[367] Togelius, J., and Lucas, S. M. Evolving Robust and Specialized Car Racing Skills. In *Proceedings of the 2006 IEEE International Congress on Evolutionary Computation* (2006), pp. 1187–1194.

[368] Tomko, N., and Harvey, I. Do Not Disturb: Recommendations for Incremental Evolution. In *Proceedings of ALIFE XII, the 12th International Conference on the Synthesis and Simulation of Living Systems* (2010).

[369] Trianni, V., Nolfi, S., and Dorigo, M. Evolution, Self-organization and Swarm Robotics. In *Swarm Intelligence*, C. Blum and D. Merkle, Eds., Natural Computing Series. Springer-Berlag Berlin, Heideberg, Germany, 2008, pp. 163–191.

[370] Tu, H., and Du, X. The Design of Small UAV Autopilot Hardware System Based on DSP. In *Proceedings of the 2010 IEEE International Conference on Intelligent Computation Technology and Automation* (2010), pp. 780–783.

[371] Tuci, E., Quinn, M., and Harvey, I. An Evolutionary Ecological Approach to the Study of Learning Behavior Using a Robot-Based Model. *Adaptive Behavior 10*, 3-4 (2002), 201–221.

[372] Turing, A. Computing Machinery and Intelligence. *Mind 59* (1950), 433–460.

[373] Urzelai, J., and Floreano, D. Incremental Evolution with Minimal Resources. In *Proceedings of IKW'99, the First International Khephera Workshop* (1999), pp. 796–803.

[374] U.S. Department of Defense. Dictionary of Military and Associated Terms - Joint Publication 1-02. `http://www.dtic.mil/doctrine/jel/doddict`.

[375] U.S. Federal Aviation Administration. *Instrument Flying Handbook (FAA-H-8083-15A)*. Superintendent of Documents, United States Government Printing Office (GPO), 2008, ch. Airplane Attitude Instrument Flight.

[376] Usui, M., Simpson, A., Smith, S., and Jacob, J. Development and Flight Testing of a UAV with Inflatable-Rigidizable Wings. In *Proceedings of the American Institute of Aeronautics and Astronautics* (2004).

[377] Valpolini, P. ISR in Afghanistan: SR Easier than I. *armada International*, 2 (2010), 46–50.

[378] Valsalam, V. K., Hiller, J., MacCurdy, R., Lipson, H., and Miikkulainen, R. Constructing Controllers for Physical Multilegged Robots using the ENSO Neuroevolution Approach. *Evolutionary Intelligence 5*, 1 (2012), 45–56.

[379] Varela, F., Thompson, E., and Rosch, E. *The Embodied Mind: Cognitive Science and Human Experience*. MIT Press, Cambridge, MA, USA, 1991.

[380] Vavak, F., and Fogarty, T. A Comparative Study of Steady State and Generational Genetic Algorithms for Use in Nonstationary Environments. *Evolutionary Computing 1143* (1996), 297–304.

[381] Venugopal, K. P., Sudhakar, R., and Pandya, A. S. On-line Learning Control of Autonomous Underwater Vehicles Using Feedforward Neural Networks. *IEEE Journal of Oceanic Engineering 17*, 4 (1992), 308–319.

[382] Vidal, R., Shakernia, O., Kim, H., Shim, H., and Sastry, S. Multi-Agent Probabilistic Pursuit-Evasion Games with Unmanned Ground and Aerial Vehicles. *IEEE Transactions on Robotics And Automation 18*, 5 (2002), 662–669.

[383] Viieru, D., Tang, J., Lian, Y., Liu, H., and Shyy, W. Flapping and Flexible Wing Aerodynamics of Low Reynolds Number Flight Vehicles. In *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit* (2006).

[384] Vincent, P., and Rubin, I. A Framework and Analysis for Cooperative Search Using UAV Swarms. In *Proceedings of the 2004 ACM Symposium on Applied Computing* (2003), pp. 79–86.

[385] Walker, M. Comparing the Performance of Incremental Evolution to Direct Evolution. In *Proceedings of the 2nd International Conference on Autonomous Robots and Agents* (2004), pp. 119–124.

[386] Watkins, S., and Vino, G. The Turbulent Wind Environment of Birds, Insects and MAVs. In *Proceedings of the 15th Australasian Fluid Mechanics Conference* (2004).

[387] Watson, R., Ficici, S., and Pollack, J. Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots. In *Proceedings of CEC 99, the 1999 Congress on Evolutionary Computation* (1999), pp. 335–342.

[388] Watson, R., Ficici, S., and Pollack, J. Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots. *Robotics and Autonomous Systems*, 39 (2002), 1–18.

[389] Weiss, G. *MultiAgent Systems. A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 2000.

[390] Weisstein, E. W. Moore Neighborhood (from MathWorld - A Wolfram Web Resource). `http://mathworld.wolfram.com/MooreNeighborhood.html`.

[391] Welch, G., and Bishop, G. An Introduction to the Kalman Filter. `http://www.cs.unc.edu/~welch/media/pdf/kalman\_intro.pdf`, 1995.

[392] Weng, J. Developmental Robotics: Theory and Experiments. *International Journal of Humanoid Robotics 1*, 2 (2004), 199–236.

[393] Widrow, B., and Jr, M. H. Adaptive Switching Circuits. In *IRE WESCON Convention Record* (1960), vol. 4, pp. 96–104.

[394] Williams, W., and Harris, M. The Challenges of Flight-Testing Unmanned Air Vehicles. In *Proceedings of SETE 2002, the Systems Engineering, Test & Evaluation Conference* (2002).

[395] Wolfram, S., and Gad-el Hak, M. A New Kind of Science. *Applied Mechanics Reviews 56* (2003).

[396] Wood, R., and Di Paolo, E. New Models for Old Questions: Evolutionary Robotics and the 'A not B' Error. In *Proceedings of the 9th European Conference on Advances in Artificial Life* (2007), pp. 1141–1150.

[397] Wooldridge, M. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Inc., New Yotk, NY, USA, 2002.

[398] Wright, S. The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution. *Proceedings of the Sixth International Congress of Genetics 1* (1932), 356–366.

[399] Wu, A. S., Schultz, A. C., and Agah, A. Evolving Control for Distributed Micro Air Vehicles. In *Proceedings of CIRA '99, the IEEE International Symposium on Computational Intelligence in Robotics and Automation* (1999), pp. 174 – 179.

[400] Wu, H., Sun, D., and Zhou, Z. Micro Air Vehicle: Configuration, Analysis, Fabrication, and Test. *IEEE/ASME Transactions on Mechatronics 9*, 1 (2004), 108–117.

[401] Wu, H., Sun, D., Zhu, H., and Zhou, Z. An Autonomous Flight Control Strategy Study of a Small-Sized Unmanned Aerial Vehicle. *IEEE Transactions on Electronics E88-C*, 10 (2005), 2028–2036.

[402] Wu, H.-y., Sun, D., Zhou, Z.-y., Xiong, S.-s., and Wang, X.-h. Micro Air Vehicle: Architecture and Implementation. In *Proceedings of ICRA 2003, the IEEE International Conference on Robotics & Automation* (2003), pp. 534–539.

[403] Yenne, B. *Attack of the Drones: a History of Unmanned Aerial Combat*. Zenith Press, Minneapolis, MN, USA, 2004.

[404] Yuh, J. A Neural Net Controller For Underwater Robotic Vehicles. *IEEE Journal of Oceanic Engineering 15*, 3 (1990), 161–166.

[405] Zagal, J., del Solar, J. R., and Vallejos, P. Back to Reality: Crossing the Reality Gap in Evolutionary Robotics. In *Proceedings of IAV 2004, the 5th IFAC Symposium on Intelligent Autonomous Vehicles* (2004).

[406] Zetule, F. Multi-Agent Systems in Search and Destroy Scenario Evolved Using Genetic Algorithm: Research on Selection Algorithm. Master's thesis, University of Plymouth, UK, 2008.

[407] Zhang, Y., Hearn, G. E., and Sen, P. A Neural Network Approach to Ship Track-Keeping Control. *IEEE Journal of Oceanic Engineering 21*, 4 (1996), 513–527.

[408] Ziemke, T. On the Role of Robot Simulations in Embodied Cognitive Science. *AISB Journal 1*, 4 (2003), 389–399.

[409] Ziemke, T., Bergfeldt, N., Buason, G., Susi, T., and Svensson, H. Evolving Cognitive Scaffolding and Environment Adaptation: A New Research Direction for Evolutionary Robotics. *Connection Science 16*, 4 (2004), 339–350.

[410] Zufferey, J.-C., Guanella, A., Beyeler, A., and Floreano, D. Flying Over the Reality Gap: from Simulated to Real Indoor Airships. *Autonomous Robots 21*, 3 (2006), 243–254.

[411] Zufferey, J.-C., Hauert, S., Stirling, T., Leven, S., Roberts, J., and Floreano, D. Aerial Collective Systems. In *Handbook of Collective Robotics: Fundamentals and Challenges*, S. Kernbach, Ed. CRC Press, London, UK, 2012.

Appendix A

Micro-unmanned Aerial Vehicles: a review

This appendix reviews some of the most popular MAV platforms available either on the market or developed for military or scientific purposes.

A.1 AeroVironment Inc.

One of the major players in MAVs arena is certainly AeroVironment Inc. (AV)[1]. The California-based company, founded by Paul MacCready in 1971, was the developer of the Black Widow MAV mentioned in previous section. Taking inspiration from the creation of that early model, the American company developed two new families of Nano-UAVs: the Hornet[2] and the Wasp[3] (with the latest that started as a N-UAV, but modified during the design process and ended up being a M-UAV).

The Hornet (see Figure A.1(a)) made what it is believed to be the world's first successful flight (21st March 2003) of a miniature air vehicle powered entirely by a hydrogen fuel cell. The Hornet's design is characterised by a straight rectangular wing of $38cm$ span, $180g$ of takeoff weight and a frontal propeller. The fuel cells are built into the top of the wing, where they combine oxygen in the ambient air with hydrogen produced internally by the N-UAV through reaction of a hybrid material

[1] `http://www.avinc.com/`
[2] `http://www.avinc.com/uas/adc/hornet/`
[3] `http://www.avinc.com/uas/small_uas/wasp/`

with water.

The Wasp family is the outcome of a multi-year effort between AV and DARPA, which evolved following a three-step (Block I, Block II, and Block III) path to reach the configuration eventually available on the market. The earliest design, which was assigned the name of Wasp Block I[4] (see Figure A.1(b)), is a N-UAV with a flying wing configuration in which the wing is in the form of a rectangle with a slightly swept leading edge. The wingspan is $33cm$, while the weight equals is just $210g$. As it is the case in all of the sub-sequential Wasp models, the Block I version mounts an electric fuelled propeller. The design of the Wasp Block II (see Figure A.1(c)) is very similar to the one of its predecessor, just slightly bigger because of a $41cm$ wingspan and a takeoff weight of $275g$, that combined together make this aircraft fall into the M-UAV category rather than in the N-UAV one. Wasp Block III[5] (see Figure A.1(d)) can be considered the final product of the AeroVironment efforts and was selected as the reference platform for the USAF BATMAV program in 2007[6]. Wasp III can be manually operated from a ground control station or pre-programmed for GPS-based autonomous navigation. With a $72cm$ wingspan and a weight of $430g$[7] the Wasp III is the smallest UAV system currently sold by AeroVironment.

More in general, for what concerns Mini-UAVs AeroVironment has a significant history to exhibit. The Dragon Eye[8] (see Figure A.2(a)) is a back-packable $1.1m$ wingspan and $2.7kg$ heavy aircraft falling into the M-UAV class[9], the design of which dates back to 2001. The aircraft mounts two forward-facing twin propellers on the main wing, powered by an electric battery that offers (in single use battery configuration) between 45 and 60 minutes of flight endurance at $35km/h$. The payload installed by default is quite rich, consisting in a dual forward and side-look EO camera, a forward and side-look low light camera, and a side-look IR camera, all of them installed on the nose of the aircraft. The ground control station that comes with this MAV guarantees an operation range of up to $5km$. What is interesting

[4] http://www.designation-systems.net/dusrm/app4/index.html
[5] http://www.avinc.com/uas/small_uas/wasp/
[6] http://www.avinc.com/resources/press_release/u.s._air_force_takes_delivery_of_batmav_micro_unmanned_aircraft/
[7] http://www.avinc.com/downloads/Wasp_III.pdf
[8] http://www.avinc.com/uas/adc/dragoneye/
[9] http://www.avinc.com/downloads/Dragon_Eye_AV_datasheet.pdf

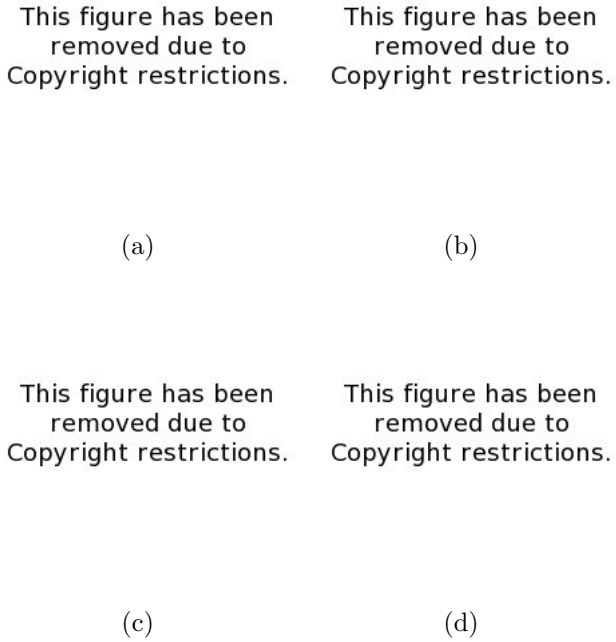


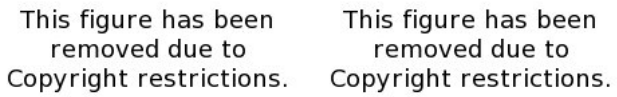
Figure A.1: (a) AV Hornet; (b) AV Wasp Block I; (c) AV Wasp Block II; (d) AV Wasp Block III / BATMAV. Sources: (a) `http://www.avinc.com/uas/adc/hornet/`; (b-c) `http://www.designation-systems.net/dusrm/app4/wasp.html`; (d) `http://www.af.mil/information/factsheets/factsheet.asp?id=10469`

about this model is the fact that the operator can control the Dragon Eye wearing a pair of properly designed video goggles. Differently from most of the products designed by AeroVironment, the Dragon Eye is bungee-launched[10]. This makes it slightly more difficult to deploy this platform on ones own. The MAV is recoverable, as it comprises features for conventional horizontal landing.

The U.S. Marine Corps, in 2003, ordered 1,000 Dragon Eye UAVs to be integrated within its forces, before switching to the Raven model when this was made available. The Raven is a family of Small UAVs for which the latest update consists in the RQ11 model[11] (see Figure A.2(b)). As it was the case with its predecessor, also the Raven B System[12] is offered in three different configurations according to the selling target: international markets[13], domestic (US) market[14] (the code name is Raven

[10]The launch system is similar to aircraft carriers catapults.
[11]`http://www.avinc.com/uas/adc/raven/`
[12]`http://www.avinc.com/uas/small_uas/raven/`
[13]`http://www.avinc.com/downloads/Raven_INTL_1210.pdf`
[14]`http://www.avinc.com/downloads/Raven_Domestic_1210.pdf`

RQ11-A for both these two models), and US Air Force[15] (RQ-11B model).



(a) (b)

Figure A.2: (a) AV Dragon Eye; (b) AV Raven RQ-11A. Sources: (a) `http://www.avinc.com/uas/small_uas/dragon_eye/`; (b) `http://www.avinc.com/glossary/raven`

All of these three setups are pretty similar to each other and characterised by a $1.4m$ wingspan and a weight of $1.9kg$. The Raven can fly at speeds between 32 and $81km/h$, at an altitude between a minimum of $30m$ and a maximum of about $150m$. This MAV is hand-launched and installs a backward facing propeller activated by an electric battery. The autonomy provided by the use of rechargeable lithium ion batteries has been estimated at 60-90 minutes, while single use batteries can provide 80-110 minutes of continuous flight. The standard payload (weighing less than $190g$) consists of two forward and side-look EO cameras (with electronic stabilised pan-tilt-zoom) installed on the MAV nose, as well as a forward and side-look IR camera. The ground control station offered with the MAV is a lightweight ruggedised one, running a control software developed by AeroVironment that combined with the hardware provided will guarantee a $10km$ operational range.

The differences between the three versions are relatively minor. The specifications mentioned above refer to the "international" model. The one targeted to the US market provides additional communication features, as for example the possibility of deconflicting occupied frequencies, and a stronger default data encryption on data transmitted between the MAV and the ground station. The military version of the Raven offers a slightly larger control range (up to $12km$) instead and more sophisticated payloads that can be installed on the aircraft (as day/night camera and thermal imager).

[15] `http://www.avinc.com/downloads/USAF_Raven_FactSheet.pdf`

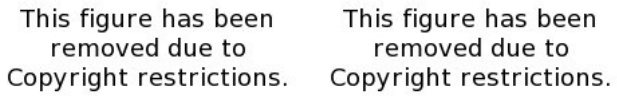
AeroVironment also has few Small-UAVs on its catalogue. The (supposedly) earliest S-UAV produced by the Californian company is the Pointer FQM-151A MAV[16] (see Figure A.3(a)). Its development started in 1986. It took about four years before both the U.S. Army and the U.S. Marine Corps decided to adopt it, buying about 50 models that were employed during the first Gulf War also. The FQM-151A is characterised by a simple single-boom parasol sailplane configuration. It is hand-launched and powered by a 300W electric motor, which uses either Li/SO2 primary or Ni/Cd rechargeable batteries to operate a backward-facing propeller installed at the middle of the main wing. The design specifications[17] mention a $2.74m$ wingspan, $4.3kg$ of takeoff weight, a maximum speed of $80km/h$, and 20/60 minutes of autonomy depending on the employment of rechargeable rather than single use batteries. The payload this MAV can transport is quite limited if compared to today's standard as it only supports a colour or a night vision camera. The ground control station used to operate the Pointer consists of two units, the pilot having a display and control box to fly the vehicle using the video from the camera (the maximum operation range guaranteed is 5km); the second operator having a hand-held display and a VCR unit with microphone to record commentary on the observed video picture.

The Puma AE (All Environment)[18] (see Figure A.3(b)) is an evolution of the earlier Puma[19] model, in turn a technological improvement over the Pointer FQM-151A[20]. With regard to its technical specifications[21], the wingspan of the Puma AE S-UAV is exactly twice as much as Raven's one, *i.e.* $2.8m$. The takeoff weight is also significantly higher, as it amounts on this model to $5.9kg$. The MAV is hand-launched and it flies with a frontal propeller controlled by an electric battery. The flight autonomy has been estimated at two hours with cruise speed of between $37km/h$ and $83km/h$, at a typical operating altitude of $152m$. The payload is hosted inside a gimballed slot and consists of an IR illuminator and a EO/IR

[16] http://www.avinc.com/uas/adc/pointer/
[17] http://www.designation-systems.net/dusrm/m-151.html
[18] http://www.avinc.com/uas/small_uas/puma/
[19] http://www.avinc.com/uas/adc/puma/
[20] http://www.defenseindustrydaily.com/Puma-AE-An-All-Environment-Mini-UAV-04962/
[21] http://www.avinc.com/downloads/PumaAE_0910.pdf

stabilised camera, capable of 360 degrees continuous pan and +10/-90 degrees tilt. One of the most interesting features of the Puma AE consists in the fact that its structure is completely waterproof, allowing the MAV to be employed for both land based and maritime operations. Despite the fact that this aircraft shares the same ground control station as the Raven and the Wasp, a relatively high level of autonomous control is provided, including the possibility for the UAV to perform an autonomous deep-stall landing. Furthermore, the more advanced communication hardware installed on the Puma AE allows up to $15km$ of operational range.

On the basis of the experience acquired with the design of the Hornet, the Puma has been develop in a fuel cell version as well[22].



(a) (b)

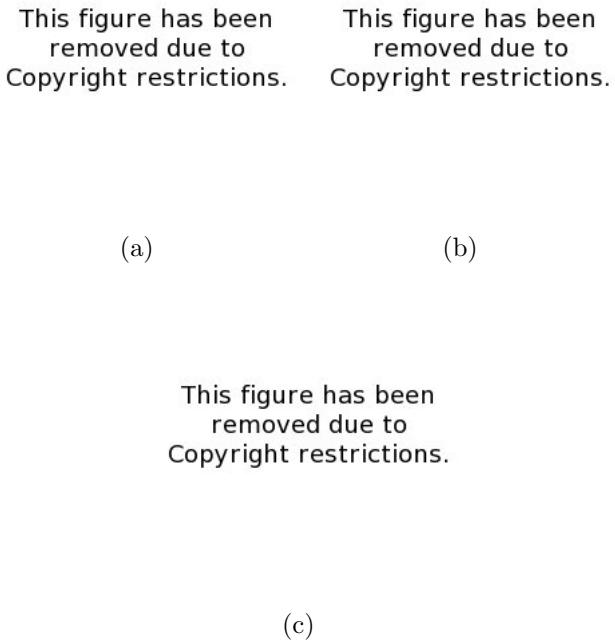
Figure A.3: (a) AV Pointer FQM-151A; (b) AV Puma AE (All Environment). Sources: (a) `http://www.avinc.com/uas/adc/pointer/`; (b) `http://www.avinc.com/uas/small_uas/puma/`

More recently, AeroVironment has started to work on two additional models: the Switchblade[23] (see Figure A.4(a)), a fixed wing aircraft that follows the history of the company in manufacturing M-UAVs, and the bio-inspired Nano Hummingbird[24] (see Figure A.4(b)), a new (at least for AeroVironment) flapping-bird UAV concept.

Closing this parenthesis dedicated to AV, it is worth mentioning the Hawk-Eye project[25]. HawkEye was the name of an interesting concept design (see Figure A.4(c)) for a non-powered Small UAV presented in 2007. Its tandem wing glider design was aimed at covertly delivering critical payloads to ground personnel with high precision. Experiments carried out on early prototypes proved that

[22] `http://www.avinc.com/uas/adc/fuel_cell_puma/`
[23] `http://www.avinc.com/uas/adc/switchblade/`
[24] `http://www.avinc.com/nano`
[25] `http://www.avinc.com/uas/adc/hawkeye/`

the HawkEye could deliver payloads as large as $25kg$[26]. The HawkEye has been conceptualised to be capable of flying up to $80km$ far from the deployment point, fully autonomously or remotely controlled. The possibility of incorporating optional propulsion systems into the MAV in order to extend endurance and range (although compromising the possibility of carrying out absolutely silent operations, virtually unnoticeable during night hours) has also been investigated.



(a) (b)

(c)

Figure A.4: (a) AV Switchblade; (b) AV Nano Hummingbird; (c) AV HawkEye. Sources: (a) `http://www.avinc.com/uas/adc/switchblade/`; (b) `http://www.avinc.com/nano/`; (c) `http://www.avinc.com/uas/adc/hawkeye`

A.2 Lockheed Martin

When discussing military technologies, Lockheed Martin[27] must be mentioned, as it is the largest provider of IT services, systems integration, and training to the U.S. Government. Principally engaged in the research, design, development, manufacture, integration and sustainment of advanced technology systems, products, and services, Lockheed Martin has one M-UAV platform on its catalogue, specifically

[26] `http://defense-update.com/features/du-1-07/aerialdelivery6-ulav.htm`
[27] `http://www.lockheedmartin.com/`

the Desert Hawk III[28] (DH III, see Figure A.5).

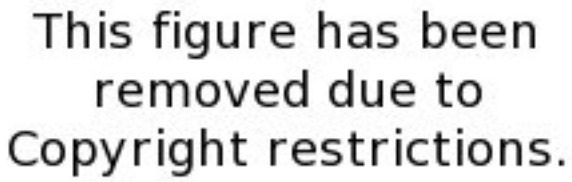


Figure A.5: Lockheed Martin Desert Hawk MAV. Source: `http://www.army.mod.uk/equipment/aircraft/1535.aspx`

The technical specifications[29] mention a $1.37m$ wingspan, an empty weight of $2.7kg$ and up to $900g$ of transportable extra payload. The flight endurance is estimated at 90 min, and the platform can cope with winds as strong as $25kts$. The operational range is up to $15km$ from the ground control station. The DH III has a 360 degree turret which can host a colour EO (capable of $10x$ optical zoom) or a long-wave IR imagers, as well as a RF signal geolocation module. Thanks to this configuration, the UAV can guarantee a continuous coverage of targets without the need to manoeuvre the aircraft.

A.3 AAI Corporation

Another significant role in the small UAVs arena is played by the AAI Corporation[30] which sells three families of unmanned aircraft systems: the Shadow, the Orbiter, and, through the controlled company Aerosonde, the Mark. The Shadow family is

[28] `http://www.lockheedmartin.com/products/DesertHawk/`
[29] `http://www.lockheedmartin.com/data/assets/ms2/pdf/Desert_Hawk_III_brochure.pdf`
[30] `http://www.aaicorp.com/`

a group of "proper" UAVs comprising the Shadow 200[31], the Shadow 400[32], and the Shadow 600[33], while the Orbiter and the Mark falls into the Small-UAV (S-UAV) category.

The Orbiter[34] (see Figure A.6), developed in collaboration with the Israel-based company Aeronautics Ltd.[35], was introduced in 2009. Its main technical specifications highlight a $220cm$ wingspan and a takeoff weight of $6.5kg$. The Orbiter flies thanks to a battery-powered backward-facing propeller, which guarantees between 2 and 3 hours of endurance at a speed in the $45-140km/h$ range. The launch method employed by this S-UAV is by catapult. The landing/recovery method is unusual also, as it consists of switching off the engine followed by the automatic deployment of a parachute and the inflation of an air bag located on the belly of the aircraft. The company does not provide any information about the default payload installed on the Orbiter. The only information available mentions a "gyro-stabilised zoom payload", and a day and night operational capability.

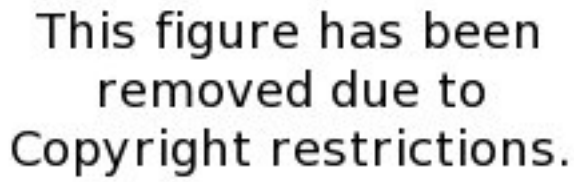


Figure A.6: AAI Orbiter Mini UAS. Source: `http://www.defenseindustrydaily.com/Mexico-Adds-More-Israeli-Surveillance-Platforms-05291/`

The Orbiter can be remotely controlled in camera-guided flight mode through the so called One System Ground Control Station (OSGCS). It is pretty interesting

[31] `http://www.aaicorp.com/pdfs/shadow_200.pdf`
[32] `http://www.aaicorp.com/pdfs/shadow400_12-18-09bfinal.pdf`
[33] `http://www.aaicorp.com/pdfs/shadow600_12-18-09bfinal.pdf`
[34] `http://www.aaicorp.com/pdfs/uas_orbiter07-20-09.pdf`
[35] `http://www.aeronautics-sys.com/`

to look at the remote control device developed by AAI Corporation. As it can be seen in Figure A.7, the remote control is built around a flat screen with about a 10" diagonal, with the two handles that allow the end user to operate it incorporating the two joysticks used to control the flight.

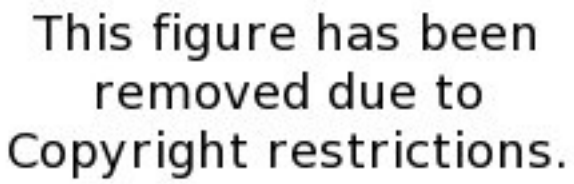
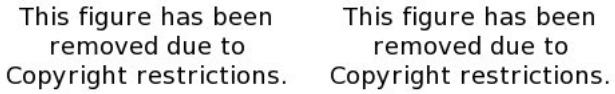


Figure A.7: One System Ground Control Station (OSGCS). Source: `http://www.aaicorp.com/pdfs/uas_orbiter07-20-09.pdf`

The Orbiter is available in two different configurations depending on the characteristics of the operations to be carried out. The default setup allows for an operational range of $15km$, while an extended version is available permitting to extend this range to $50km$.

For what concerns the AAI-Aerosonde Mark 4.7[36] (see Figure A.8(a)) this is an exponent of the Aerosonde Mark 4 UAS modular fleet. The Mark 4.7 uses a propulsion system which is relatively unusual for small UAVs, consisting in a J-type, four-stroke, $24cc$ electronic fuel injection (EFI) combustion engine. The use of such a propulsion system (which can also be upgraded to a K-twin, dual cylinder, four-stroke, EFI engine) makes the Mark 4.7 significantly heavier than miniature UAV systems of comparable sizes. With a wingspan of $360cm$, the weight of the aircraft amounts to either 17.5 or $25kg$ depending on the engine installed. The cruise speed guaranteed is about $90 - 110km/h$ with dash speeds of $115 - 150km/h$ at sea level.

[36] `http://www.aerosonde.com/pdfs/aerosonde-mark-47.pdf`



(a) (b)

Figure A.8: (a) AAI-Aerosonde Mark 4.7; (b) details of the combined launch/recovery system used by the Mark 4.7. Sources: (a) `http://www.azorobotics.com/News.aspx?newsID=2187`; (b) `http://www.defenseindustrydaily.com/From-Dolphins-to-Destroyers-The-ScanEagle-UAV-04933/`

The Mark 4.7 takes off with the aid of a rail launcher, while the the recovery is provided by a net. AAI offers a trailer-mounted, combined launch/recovery system, which allows for deployment within confined-area and maritime operations (see Figure A.8(b)). Again, no extensive details are provided about the payload installed as standard. The specifications just mention in fact a "combined electro-optic (EO), infrared (IR) and laser pointer (LP) payload".

A.4 Israel Aerospace Industries (IAI)

In addition to conventional UAVs, another company, namely IAI (Israel Aerospace Industries)[37], has on its catalogue three different miniature unmanned aircraft.

First of all we have the Mosquito[38] (see Figure A.9), a Nano-UAV (N-UAV) with a wingspan of $35cm$ and a maximum takeoff weight (including a $150g$ payload) equal to $500g$. The Mosquito is moved by a front-facing propeller, fuelled by an electric battery that guarantees about 40 minutes of endurance at the cruise speed of $60km/h$ (the Mosquito can allegedly reach a maximum speed of $110km/h$). The only payload carried by this aircraft consists of a miniature colour video camera. The Mosquito can be hand or bungee launched and lands on its belly.

The BirdEye 650[39] (see Figure A.10) is a significantly larger model, given its $3m$

[37] `http://www.iai.co.il`
[38] `http://www.iai.co.il/sip_storage/FILES/4/38204.pdf`
[39] `http://www.iai.co.il/sip_storage/FILES/3/38203.pdf`

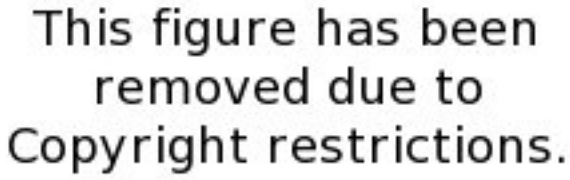


Figure A.9: IAI Mosquito Micro-UAV. Source: http://www.iai.co.il/sip_storage/FILES/4/38204.pdf

wingspan and the $11kg$ of takeoff weight ($1.2kg$ of which are due to the payload) this makes it fall into the S-UAV category. The size imposes constraints on the launch method, as the takeoff that can only take place through a rail launcher. A single backward-facing propeller keeps the BirdEye in the air. The aircraft can install both standard and fuel cell batteries, providing endurance for 3 and 7 hours respectively at a cruise speed of $75km/h$ (the maximum speed of this aircraft is about $120km/h$).

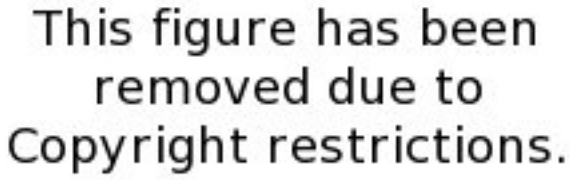


Figure A.10: IAI Mini UAS BirdEye 650. Source: http://www.flightglobal.com/articles/2010/02/10/338193/iai-unveils-bird-eye-650-uav.html

Finally IAI sells the Mini Panther[40] (see Figure A.11), an electrically propelled Small UAV with a $2.5m$ wingspan, and a maximum takeoff weight of $12kg$ ($2kg$ for the payload). The main characteristics of the Mini Panther consists in the possibility for the back propeller of tilting (this UAV uses three propellers, two forward-facing and the other one oriented backward), thus making the aircraft capable of automatic vertical takeoff and landing (AVTOL). The operation range is about $20km$ and the integrated battery guarantees up to 1.5 hours of endurance at a cruise speed between 55 and $75km/h$. The default payload includes EO and IR cameras.

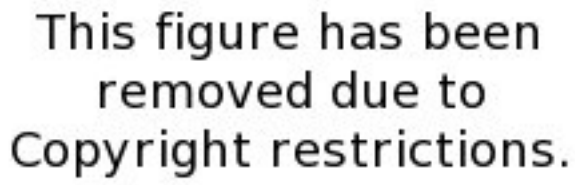


Figure A.11: IAI Mini Panther. Source: `http://www.iai.co.il/sip_storage/FILES/9/38199.pdf`

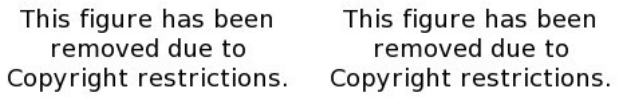
A.5 Insitu

Among the companies actively involved in miniature UAVs production we find the Washington-based Insitu[41] also, which has on its catalogue two different models.

The ScanEagle[42] (see Figure A.12(a)), developed in collaboration with Boeing, is a $3.11m$ wingspan Small UAV supporting a maximum takeoff weight of $20kg$. The aircraft is powered by a $1.4kw$ 2-stroke engine, fed using either gasoline or

[40] `http://www.iai.co.il/sip_storage/FILES/9/38199.pdf`
[41] `http://www.insitu.com`
[42] `http://www.insitu.com/documents/InsituWebsite/MarketingCollateral/ScanEagleFolderInsert.pdf`

heavy fuel. The endurance level guaranteed by such propulsion system is extremely high and has been estimated at 24 hours or more at a cruise speed of $90km/h$ (the maximum horizontal speed this aircraft can reach is slightly lower than $150km/h$). The standard payload consists of a high-resolution electro-optic camera or an IR one, installed on an inertially stabilised turret system. The vehicle can communicate with a ground station located up to $100km$ from it. The launch and recovery methods of the ScanEagle are quite unusual as they consist in a pneumatic catapult launcher for the takeoff, and in a "SkyHook wingtip capture[43]" system for the recovery part (see Figure A.12(b)).



(a) (b)

Figure A.12: (a) Insitu ScanEagle; (b) the SkyHook device used for the recovery. Source: `http://www.insitu.com/index.cfm?navid=422`

The ScanEagle is also available in a "dual bay" version[44] which provides additional room for payload to be installed.

The NightEagle[45] (see Figure A.13) is the other S-UAV model produced by Insitu. Slightly bigger than the ScanEagle (from which it is derived), thanks to its $3.19m$ wingspan and its maximum takeoff weight of $22kg$, the NightEagle differentiates from its predecessor mainly because it carries a mid-wave infrared (MWIR) imager payload onboard.

[43] `http://www.roboticstrends.com/security_defense_robotics/article/insitu_demonstrates_one_launch_and_recovery_system`

[44] `http://www.insitu.com/documents/InsituWebsite/MarketingCollateral/ScanEagleDualBayFolderInsert.pdf`

[45] `http://www.insitu.com/documents/InsituWebsite/MarketingCollateral/NightEagleFolderInsert.pdf`

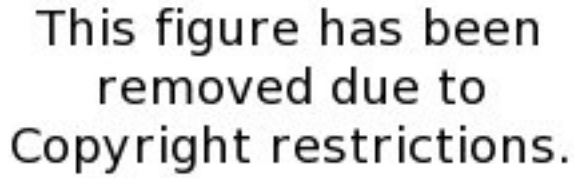


Figure A.13: Insitu NightEagle. Source: `http://www.insitu.com/documents/InsituWebsite/MarketingCollateral/NightEagleFolderInsert.pdf`

A.6 Elbit Systems

The last company we have decided to include in this brief review is Elbit Systems[46], which produces two families of UAV systems: the Hermes as full-sized UAVs, and the Skylark for what concerns miniature systems. The Skylark comprises two models: the Skylark I-LE[47], and the Skylark II[48]. The latter has been excluded from this review as its size does not match the criterion we have established for categorising small UAVs.

The Skylark I-LE (see Figure A.14) is a $2.9m$ wingspan S-UAV having a maximum takeoff weight of $6.5kg$. With an operation radius of $15km$ and an endurance estimated in 3 hours, the Skylark counts on its flexibility rather than on its technical specifications. The entire system (UAV and ground station) can be easily backpacked and quickly deployed thanks to the possibility of launching it by hand (although through a bungee-assisted system). The recovery method consists of making the UAV perform an automatic deep-stall followed by the inflation of an air-bag located under its belly. The default payload consists of gimbaled and stabilised EO or IR cameras. An interesting feature offered by this system is the simulator

[46] `http://www.elbitsystems.com`
[47] `http://elbitsystems.com/Elbitmain/files/Skylark_1_LE.pdf`
[48] `http://elbitsystems.com/Elbitmain/files/Skylark_2_hr.pdf`

environment embedded in the ground station, which allows the simulation of flight paths that can be followed before actually going through them.

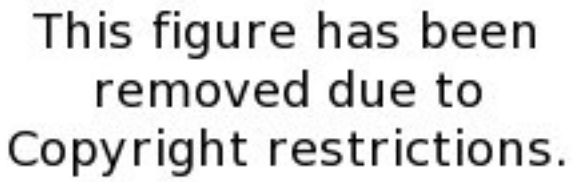
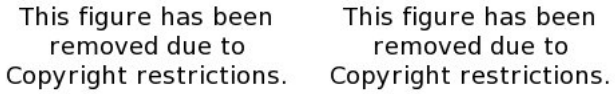


Figure A.14: Elbit Systems Skylark I-LE. Source: `http://defense-update.com/products/s/skylark1-uav.htm`

A.7 AerialRobotics

For what concerns the amateur/civilian usage of small UAVs, AerialRobotics[49] - a little company located in Poland - has gained a strong reputation as provider of affordable and robust solutions for aerial photography. AerialRobotics started producing the EasyUAV[50], a UAV developed on the basis of the EasyStar ARF Electric RC Airplane[51] (see Figure A.15(a)) with the integration of a self-developed autopilot system (the FlexiPilot, that will be reviewed in one of the next sections), and photographic equipment. The EasyStar is an electrically propelled aircraft, which mounts a single backward facing propeller. Its wingspan amounts to $1.38m$ and its takeoff weight to $680g$. As the original design has been made with hobbyists in mind, the endurance offered by the Permax 400 engine just reaches the 12-18 minutes range.

[49] `http://www.aerialrobotics.eu`
[50] `http://www.aerialrobotics.eu/easyuav/easyuav-manual-en.pdf`
[51] `http://www.hobby-lobby.com/easystar.htm`



(a) (b)

Figure A.15: (a) AerialRobotics EasyUAV (modified EasyStar); (b) AerialRobotics Pteryx. Sources: (a) `http://www.diydrones.com/profile/RobertDrone`; (b) `http://www.troybuiltmodels.com/items/PTERYX-UAV.html`

As a side note, it is interesting to report something that appears in the EasyUAV user's guide and which justifies the usage of autonomous UAVs (in place of standard remote-controlled aerial vehicles) for tasks such as aerial photography:

> "*The popular perception of UAV (Unmanned Aircraft Vehicle) blends with RC flying models. While both can look the same, the actual style of operation and goals differ significantly. A flying RC model is used under manual control all the time and requires constant visual supervision of the operator in order to keep the plane inside comfortable visual range, which is around 300-500m. The manoeuvers of such planes are very violent despite best efforts of their operators, aerial video obtained this way is notoriously unstable and the photos are either blurred or badly positioned due to large effort of combining photo shooting with navigation and maintaining level flight. The use of wireless camera and goggles allows for very popular hobby called FPV (First Person View) flying. While such systems usually display GPS position, flying smoothly along waypoint still requires constant supervision and is usually not feasible with precision better than 50m while keeping the flying style smooth, what is a requirement for good quality video and sharp aerial photography. Another limitation is the weight of the wireless video gear that occupies the payload for a better camera.*"

The EasyUAV was recently updated, resulting in the production of the Pteryx S-UAV[52] (see Figure A.15(b)). The design of this new model is similar to that of its predecessor, with the main difference consisting in the position of the propeller, which is now facing forward rather than backward. With a wingspan of $2.4m$, a maximum takeoff weight of $5kg$ (up to $1kg$ of which consists in the payload,) and a lithium-ion polymer battery operating a brushless DC electric motor which

[52] `http://www.trigger.pl/pteryx/Pteryx-UAV.php`

guarantees $1h$ endurance with $1kg$ payload (up to $2h$ with $250g$ payload), the Pteryx presents itself as a more solid and flexible platform compared with the EasyUAV. Most of the advantages provided by this aircraft come from the fact that it can be easily assembled (the process takes only five minutes according to the company), and does not necessarily require a ground station to fly. The data collected by the embedded digital camera can be used for generating digital elevation models using external photogrammetric software and orthorectification procedure, obtaining surface maps for precision agriculture using mosaicking software, and performing site and long range linear mapping through georeferencing of the data obtained.

A.8 Miscellaneous

The Airborne S-UAV platform [254] (see Figure A.16), developed by Miller and colleagues at the Pennsylvania State University on the basis of the commercially available SIG Kadet Senior RC plane[53], is a $2m$ wingspan aircraft with a takeoff weight just below the 3kg built with the intention of studying intelligent control. The Airborne is powered by a 4-stroke 0.91 cube inches engine. In order to be made autonomous, the aircraft has been further equipped with a NiMH battery pack that provides energy to the servomotors operating the flight control surfaces. The main physical structure has been also strengthened in order to support the additional weight installed. An autopilot (specifically a no longer available Piccolo Plus[54]), an onboard computer (namely an Ampro ReadyBoard 800 Single Board Computer, SBC[55]), and several sensors (as, among others, a GPS receiver) have been incorporated into the structure as well. An autonomous intelligent controller (IC, according to the definition used by the authors) is responsible of the high level behaviours.

Wu and colleagues [401, 400], in several works carried out mainly at the Tsinghua University in Beijing, have developed several prototypes of miniature UAVs, both in

[53] http://www.sigmfg.com/IndexText/SIGRC58ARFB.html

[54] http://www.cloudcaptech.com/SalesandMarketingDocuments/PiccoloComparisonTable.pdf

[55] http://www.ampro.com/Products/ReadyBoard/readyboard_800/

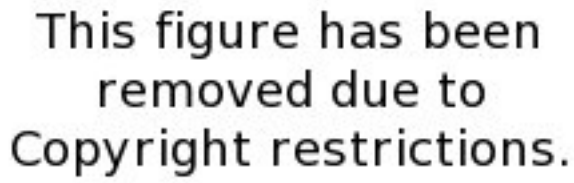


Figure A.16: Airborne platform. Source: [254]

triangle and square wing planforms, focusing on the study of aerodynamics at low Reynolds numbers, through wind and water tunnels [360]. Two of the prototypes they have built have been labelled TH360 and TH380 respectively (see Figure A.17 for an example).

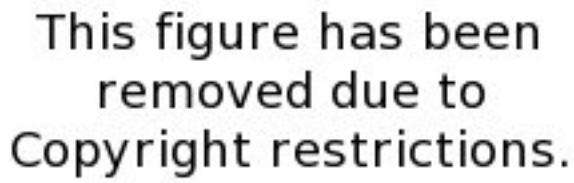
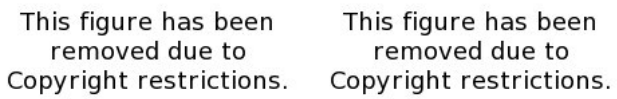


Figure A.17: One of the research platforms developed by Wu and colleagues. Source: [400]

We can now close this review mentioning a very popular product that hit the shelves a few months ago. Other than be used for military operations, intelligence purposes, law enforcement or various categories of civilian applications, small RC

air vehicles can be considered entertainment tools also. Even if it is not a fixed-wing aircraft, which is the case for the AR.Drone[56], a quad-rotor configuration system produced by Parrot[57] (see Figure A.18(a)). Although this platform has not been thought to work autonomously, it is so robust and easily modifiable that on the Internet several projects already appeared that made it possible to interact with the outer navigation loop (while leaving the AR.Drone taking care of the inner stabilisation loop), thus making the aircraft a UAV by all means[58].



(a) (b)

Figure A.18: (a) Parrot AR.Drone; (b) screenshot of the AR.FlyingAce application. Sources: (a) `http://www.itfgaming.com/tech-review/parrot-ar-drone-review`; (b) `http://ardrone.parrot.com/parrot-ar-drone/en/ar-games/ar-flyingace`

By default the AR.Drone can be remotely controlled by a dedicated app running on an iPhone via a wireless link. Other than just being remotely controlled, additional applications are available allowing for more intensive (and entertaining) interactions, such as simulating dogfights through augmented reality (see for example the AR.FlyingAce application[59], for which a screenshot can be found in Figure A.18(b)).

A.9 Classification

Just to summarise and conclude this section, we present here a table (Table A.1) that resumes the main characteristics (wingspan and takeoff weight) of the aircraft described in this section. This data is what has been used to plot Figure 3.3, which displays both the boundaries between the categories (Small, Mini, and Micro UAVs)

[56] `http://ardrone.parrot.com/parrot-ar-drone/usa/`
[57] `http://www.parrot.com/usa/`
[58] `http://diydrones.com/profiles/blogs/turning-the-parrot-ardrone`
[59] `http://itunes.apple.com/us/app/ar-flyingace/id422272353?mt=8&ls=1`

we have identified in chapter 3 and where the Miniature UAVs presented herein falls according to this classification system.

Table A.1: Classification of the miniature UAVs reviewed in the appendix, done accordingly to the categories outlined in Chapter 3

Manufacturer	Model	Wingspan (cm)	Weight (g)	Category
AeroVironment	Black Widow	15.2	56	Nano
AeroVironment	Hornet	38	180	Nano
AeroVironment	Wasp Block I	33	210	Nano
IAI	Mosquito	35	500	Nano
AerialRobotics	EasyUAV	138	680	Mini
AeroVironment	Wasp Block II	41	275	Mini
AeroVironment	Wasp Block III (BATMAV)	72	430	Mini
AeroVironment	Raven RQ-11B	140	1900	Mini
AeroVironment	Dragon Eye	110	2700	Mini
Lockheed Martin	Desert Hawk III	137	3600	Mini
senseFly	Swinglet	80	420	Mini
AAI	Mark 4.7	360	25000	Small
AAI	Orbiter	220	6500	Small
AerialRobotics	Pteryx	240	5000	Small
AeroVironment	Puma AE	280	5900	Small
AeroVironment	Pointer FQM-151A	274	4300	Small
Elbit Systems	Skylark I-LE	290	6500	Small
IAI	BirdEye 650	300	11000	Small
IAI	Mini Panther	250	12000	Small
Insitu	ScanEagle	311	20000	Small
Insitu	NightEagle	319	22000	Small

Appendix B

Autopilot systems

This appendix describes the most common autopilot systems available on the market, presenting the most relevant technical details as well as referring to some scientific publications in which they have been employed.

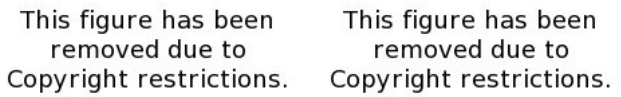
B.1 SBP400/MNAV

Crossbow[1] is one of the world leading suppliers in "smart-sensors technologies" aimed at military programs and high-value, asset-tracking operations. One common solution to the design of autopilot systems for MAVs consists in the combined use of two products developed by Crosswbow, namely a Stargate board computer and a MNAV Inertial Measurement Unit.

The SPB400 Stargate Gateway[2] (see Figure B.1(a)) is a compact ($8.9x6.35cm$) and low-power onboard computer endowed with a 400MHz PXA55 Intel XScale processor and 64 MB of SDRAM memory. It provides several input/output interfaces, such as Ethernet, RS-232, JTAG, USB, PCMCIA, and Compact Flash (CF). In terms of communication capability, a Bluetooth interface is built in, while 802.11 Wi-Fi can be operated through dedicated PCMCIA or CF cards, or even USB dongles. From a software point of view, the Stargate can run a dedicated Debian-compatible open-source Linux distribution, based upon the 2.4.19 Kernel.

[1] `http://www.xbow.com`
[2] `http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=229`

The MNAV[3] is a family of Inertial Measurement Units (IMUs) that, according to the introduction present on the company website, has been designed for "*the purpose of surface and aerial automated vehicle control and navigation*[4]". The 100CA model[5] (see Figure B.1(b)) is often used for aerial applications, and particularly in conjunction with the above introduced SPB400 board, thanks to the 51-pin connector that allows an easy connection between the two. Its comprehensive onboard servo control solution includes both RC servo control hardware and an RC receiver Pulse Position Modulation (PPM) interface. RC servo hardware provides users with software-based control of up to nine separate servos while the PPM interface enables software interpretation of RC receiver commands thereby offering users both automated software control as well as manual "take-over" capability. The readings provided by the internal sensors are given in output in a digital format and can be accessed via a RS-232 link (in absence of a direct connection between the IMU and the processing device).



(a) (b)

Figure B.1: (a) Crossbow SPB400 Stargate Gateway; (b) Crossbow MNAV 100CA Inertial Measurement Unit. Sources: (a) `http://www.willow.co.uk/html/spb400-_stargate_gateway.html`; (b) `http://www.gpsarea.com/p_detail.asp?ID=412`

The MNAV Autopilot Project[6], started in 2005 and has been regularly updated since then, provides an easy to use open-source software that can be employed as a complete autopilot system with the SPB400 + MNAV hardware, and as a ground

[3] `http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=193`

[4] Other inertial systems developed by Crossbow are those belonging to the NAV420 (`http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=181`) and NAV425EX (`http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=251`) families.

[5] `http://bullseye.xbow.com:81/Products/Product_pdf_files/Inertial_pdf/uNAV_Datasheet.pdf`

[6] `http://micronav.sourceforge.net/`

station also.

Jang e Liccardo describe in [181] how to implement a complete autopilot system on an off-the-shelf RC aircraft. Hing and colleagues [164] instead use the Stargate+MNAV combination to develop an unmanned aerial vehicle piloting system.

B.2 Procerus Kestrel System

Manufactured by Procerus Technologies[7], Kestrel Autopilot System[8] (see Figure B.2) is allegedly the smallest ($5x3.48x1.2cm$) and lightest ($16.7g$) full-featured micro autopilot on the market, incorporating on its body both the IMU and the processing unit.

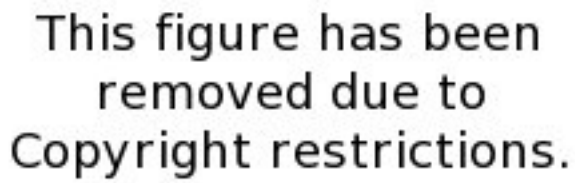


Figure B.2: Procerus Kestrel Autopilot System. Source: `http://www.procerusuav.com/productsKestrelAutopilot.php`

According to the official data sheet[9], the sensorial apparatus consists of a 3-axis angular rate and acceleration measurement device, a magnetometer (2 and 3-Axis), a 20-point sensor temperature compensation system, and absolute and differential pressure sensors providing barometric pressure, wind estimation, aircraft air speed and altitude measurement. The system can use an external GPS unit for inertial navigation and wireless modems communications between the ground station and

[7] `http://www.procerusuav.com/`
[8] `http://www.procerusuav.com/productsKestrelAutopilot.php`
[9] `http://www.procerusuav.com/Downloads/DataSheets/Kestrel_2.4.pdf`

the autopilot. A miniaturised camera can be easily incorporated into the system as well. The computing aspects are managed by a $29MHz$ Rabbit processor with $512Kb$ of RAM memory, significantly slow in comparison with the one used on the SBP400 board, but that can be replaced attaching an external unit to the Kestrel. From an electromechanical perspective, the Kestrel can control 4 servos by default, and up to 12 through additional ports.

The Kestrel can be installed on MAVs of different shapes. Procerus offers the Unicorn (see Figure B.3), a simple airframe made of EPP foam that can be successfully used for research purposes, as a testbed platform. The University of Missouri S&T AESS UAV Team describes in its online blog[10] how they integrated the Kestrel into several MAVs they designed.

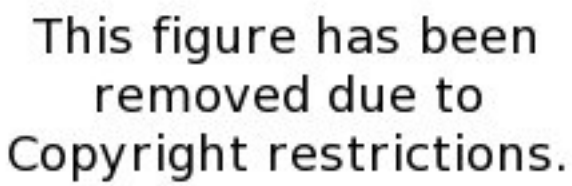


Figure B.3: Procerus Unicorn MAV. Source: `http://www.procerusuav.com/images/large/img_zagi-closed_lrg.jpg`

The autopilot system is completed by two proprietary pieces of software provided by Procerus, the Kestrel Autopilot v2.4 software, to be run on the onboard controller, and Virtual Cockpit v2.6[11] for the ground station.

A new version of the Kestrel Autopilot System (v3.0/VTOL)[12] has recently been released but no third-party reports about its usage have been found yet.

[10] `http://www.aessuav.org/`
[11] `http://www.procerusuav.com/Downloads/DataSheets/Virtual_Cockpit_2.6.pdf`
[12] `http://www.procerusuav.com/Downloads/DataSheets/Kestrel3_VTOL_System_2010.pdf`

B.3 MicroPilot MPxx28

MicroPilot[13] offers a wide range of autopilot systems[14], divided into several families: the entry-level MP1028[15], the mid-range MP2028 (MP2028^{xp}[16] and MP2028^{g}[17]), and the latest top-range MP2128 (MP2128^{g}[18] and MP2128^{heli}[19]).

The widely employed MP2028^{g} (see Figure B.4) weights 28 grams (excluding the GPS antenna) and it is $10x4x1.5cm$ in size. The device has a fully integrated 3-axis gyros/accelerometers, a GPS receiver, pressure altimeter and pressure airspeed sensors; it can control 8/16/24 servos according to the configuration. The onboard computation is guaranteed by the use of a $20MHz$ Motorola processor.

From a technical point of view it is interesting to consider how the designers of the MP2028^{g} have decided to implement two separated energy supply circuits, one of those solely dedicated to the control of the servos.

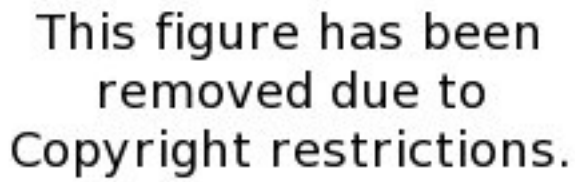


Figure B.4: MicroPilot MP2028^{g}. Source: `http://tom.pycke.be/category/Construction/`

The system does not appear as the most flexible one in terms of customisation possibilities for the end user, who is essentially only allowed to adjust the

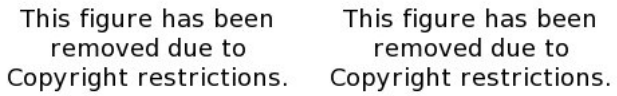
[13] `http://www.micropilot.com/`
[14] `http://www.micropilot.com/products-mp2028-autopilots.htm`
[15] `http://www.micropilot.com/products-mp1028g.htm`
[16] `http://www.micropilot.com/products-mp2028xp.htm`
[17] `http://www.micropilot.com/products-mp2028g.htm`
[18] `http://www.micropilot.com/products-mp2128g.htm`
[19] `http://www.micropilot.com/products-mp2128heli.htm`

autopilot functioning by tuning some parameters within the control feedback loops. HORIZON^{mp} UAV Ground Control Software[20] is the software provided by MicroPilot to be run on the ground station.

An application of the Micropilot MP2028^{g} can be seen in the LinkMAV platform developed by Doherty and colleagues [98].

B.4 Cloud Cap Piccolo

Cloud Cap[21] provides two families of autopilot systems, Piccolo SL[22] and Piccolo II[23], both containing all the required control elements (3-axis gyroscope, 3-axis acceleration sensor, GPS receiver, integrated radio link, etc.) inside a small EMI shielded enclosure. According to the official data sheet[24] the two models are extremely similar to each other in terms of capabilities, with the main difference consisting in the slightly higher flexibility offered by the II model, thanks to the more input interfaces available. This reflects in terms of size, with Piccolo II being significantly bigger than the SL model (13.1x5.56x1.9cm for the Piccolo SL, 14.2x4.6x6.2cm for the Piccolo II; see Figure B.5).



(a) (b)

Figure B.5: (a) Cloud Cap Piccolo SL; (b) Cloud Cap Piccolo II. Source: `http://www.cloudcaptech.com/piccolo_system.shtm`

The two autopilot systems support three on-the-fly flight modes: autonomous, stability augmented steering, and manual control. Piccolo autopilots are available

[20] `http://www.micropilot.com/products-horizonmp.htm`
[21] `http://www.cloudcaptech.com`
[22] `http://www.cloudcaptech.com/piccolo_sl.shtm`
[23] `http://www.cloudcaptech.com/piccolo_II.shtm`
[24] `http://www.cloudcaptech.com/SalesandMarketingDocuments/PiccoloAutopilotSystem.pdf`

in several different software configurations (economy feature set, standard feature set, laser altimeter, RTK DGPS, RTK + moving platform recovery, helicopter operations), depending on the type and on the complexity of the reference application.

Cloud Cap provides two ready-to-use solutions for the ground station, a desktop and a portable (PGS) one[25], both running Piccolo Command Center (PCC)[26] software.

The Piccolo SL (see Figure B.5(a)) is a replacement for the successful Piccolo LT and Piccolo Plus models, that have been used for countless applications. Among those, particularly relevant are the studies carried out by Almeida and colleagues [10] at the University of Porto[27], and by King et al. [190]. Ryan and colleagues [330] have used a Piccolo autopilot in conjunction with a PC104 onboard computer on a modified Sig Rascal 40 ARF aircraft[28]. The Piccolo II has been extensively tested by Jager [178] instead.

B.5 UNAV 35xx and PICOPILOT

UNAV[29] produces some of the most inexpensive complete autopilot systems available on the market. Two families of products are currently available: 35xx[30] and PICOPILOT[31].

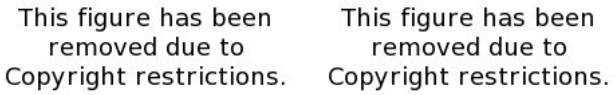
The 35xx family is built around the UNAV 3500FW[32] (10.16x5.08x1.9cm, 35g), a complete autopilot system that comprises a complete onboard AHRS (Attitude and Heading Reference System), combined eight airdata sensors, a waterproof GPS receiver and a radio-modem covering a 6 mile radius. Up to 7 servos can be controlled by this autopilot that can also rely on two serial ports for being extended with external peripherals.

Two miniaturised and slightly less sophisticated versions of the 3500FW (both

[25] http://www.cloudcaptech.com/piccolo_groundstation.shtm
[26] http://www.cloudcaptech.com/piccolo_command_center.shtm
[27] http://whale.fe.up.pt/asasf/index.php/Main_Page
[28] http://www.rcuniverse.com/magazine/article_display.cfm?article_id=895
[29] http://www.u-nav.com
[30] http://www.u-nav.com/3550.html
[31] http://www.u-nav.com/picopilot.html
[32] http://www.u-nav.com/3500fw.html

5.08x2.54x1.27cm in terms of size) are available, namely the 38g 3520[33] and the cheaper 36g 3550[34] (see Figure B.6(a)).

The PICOPILOT family was originally designed for electric motor-gliders, thus the models belonging to this family can control the aircraft on one axis only (rudder or aileron). UNAV makes the PICOPILOT available in four different configurations: PICOPILOT-N[35] (see Figure B.6(b)), PICOPILOT-NA[36], PICOPILOT-NAT[37], and PICOPILOT-RTL[38].



(a) (b)

Figure B.6: (a) UNAV 3550 sUAS autopilot; (b) UNAV PICOPILOT-N. Source: http://www.u-nav.com

PICO-GS[39] is the ground station software provided by the company. It is very basic in terms of functionalities, as it only allows the user to display GPS data received by a PICOPILOT system and transmitted via radio link to the ground station. PICO-GS also features a 'Point-n-Click' waypoint creation utility that eases waypoint programming.

An application of the PicoPilot system to the SIG Kadet[40] RC plane can be found in [231]. The PicoPilot was also considered for the development of the Peregrine Return Vehicle[41], but an alternative solution was eventually preferred.

[33] http://www.u-nav.com/3520.html
[34] http://www.u-nav.com/3550.html
[35] http://www.u-nav.com/picopilot/ppn.html
[36] http://www.u-nav.com/picopilot/ppna.html
[37] http://www.u-nav.com/picopilot/ppnat.html
[38] http://www.u-nav.com/picopilot/pprtl.html
[39] http://www.u-nav.com/picopilot/picogs.html
[40] http://www.sigmfg.com/IndexText/SIGRC74.html
[41] http://spacegrant.colorado.edu/boulder/past/Peregrine05032007/index.htm

B.6 FlexiPilot and EasyUAV

FlexiPilot (see Figure B.7) is a complete autopilot system that was developed at first at a hobbyist level[42] by Krzysztof Bosak[43] to work in conjunction with the EasyUAV MAV platform[44] (See Figure A.15(a)). FlexiPilot was originally designed to work without the need of a ground station and in the easiest way possible, thus integrating features as self-calibration and self-initialisation[45]. The autopilot quickly improved in terms of performances, soon getting integrated into professional solutions for aerial photography as the Pteryx MAV[46] (and not being sold anymore as an individual system).

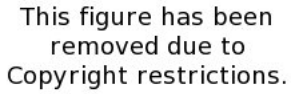


Figure B.7: Two FlexiPilot autopilot systems stacked on top of each other. Source: http://diydrones.com/page/flexipilot-1

The FlexiPilot incorporates a 6-DOF Inertial Measurement Unit and relies on an external GPS receiver. The 5 output channels integrated provide control over rudder, elevator, throttle, and two triggers, thus guaranteeing the applicability of the autopilot on a wide range of aerial platforms.

From a software point of view, the FlexiPilot offers 3D waypoint navigation, automatic landing, and several low level functions that keep the aircraft stable making it able to perform manoeuvres in the cleanest way possible.

[42] http://www.rcgroups.com/forums/showthread.php?t=1137076
[43] http://diydrones.com/forum/topic/listForContributor?user=kbosak
[44] http://www.aerialrobotics.eu/easyuav/easyuav-manual-en.pdf
[45] http://www.aerialrobotics.eu/flexipilot/flexipilot-advantage-en.pdf
[46] http://www.trigger.pl/pteryx/Pteryx-UAV.php

Appendix C

P-ARTS (Plymouth Advanced Robot Training Suite)

"P-ARTS" (Plymouth Advanced Robot Training Suite) is the name given to the computer facility used for carrying out the experiments described in Chapter 6. The set of machines constituting this system have been awarded to our research group by Apple™following our successful application to Apple's ARTS (Apple Research & Technology) programme[1]. The next two sections will briefly describe the main components of P-ARTS both in terms of hardware and software.

C.1 Hardware

P-ARTS is physically constituted by two groups of components.

On one side we have a 24" Apple iMac desktop computer, powered by a 2.8Ghz Intel Core 2 Duo processor and supported by 2GB of RAM and a 300GB hard drive. This computer is not supposed to perform heavy computation, rather to manage and supervise the work done by the "number cruncher" component of the grid. The "brute force" comes in fact from four (now discontinued) Apple Xserve "Xeon[2]" machines. The main specifications for each of these four servers mention: a double 2.8GHz Quad-Core Intel Xeon processors (12MB of L2 Cache memory per processor), 4GB of RAM (800 MHz DDR2 FB-DIMM), and a 80GB hard drive.

[1] `http://www.apple.com/uk/education/arts/`
[2] `http://en.wikipedia.org/wiki/Xserve`

Three of these machines are "headless", while the other one mounts an ATI Radeon X1300 graphics card with $64MB$ of VRAM.

All of the machines are connected to the university network, thus being accessible from any computer within the campus network (either through wired or wireless connection) or from the outside (via VPN connection).

C.2 Software

The grid is controlled by Sun Grid Engine (SGE[3]), a popular open-source software specifically designed for this purpose.

The typical configuration of a grid managed by SGE consists of a "master computer", one or more "submission hosts", and a variable number of "execution hosts". In our case, the iMac works both as master and submission host. This means that it makes sure the entire system is working properly, and allows end-users to interact with the grid submitting/stopping jobs, monitoring their status, setting different levels of priorities for the various jobs, etc.. The four Xserve machines are all set up as execution hosts, which means they receive the jobs submitted by the master and execute them.

A NFS shared file system is used to make all the machines capable of reading/writing data from/on the same files/folders.

[3] `http://gridengine.sunsource.net/`

Appendix D

Mathematical operations

In this final appendix we will provide a reference for some of the mathematical and trigonometrical operations mentioned in the main body of this thesis.

D.1 Distances in three dimensions

The Euclidean distance between two points a and b both laying inside the same three dimensional space can be calculated according to Equation D.1, where (x_a, y_a, z_a) are the coordinates of point a and (x_b, y_b, z_b) those of point b.

$$d = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2} \tag{D.1}$$

D.2 Convert from degrees to radians and vice versa

The conversions from degrees to radians and from radians to degrees can be performed utilising Equations D.2 and D.3 respectively.

$$degrees = radians \times \frac{180}{\pi} \tag{D.2}$$

$$radians = degrees \times \frac{\pi}{180} \tag{D.3}$$

D.3 Mean of circular quantities

In order to calculate the mean of several circular quantities a particular procedure must be followed. Circular quantities cannot in fact be averaged as is typically done for non-circular quantities, by simply adding all of them together and then dividing the result by the amount of quantities considered.

The typical procedure followed consists in the following steps instead. First of all, all the angles must be converted into their corresponding points on the unit circle, e.g., α to $(\cos\alpha, \sin\alpha)$. That is convert polar coordinates to Cartesian coordinates. Then the arithmetic mean of these points can be computed. The resulting point will lie on the unit disk and it can be converted back to polar coordinates. The resulting angle is a reasonable mean of the input angles.

Equation D.4 shows a possible mathematical formalisation of this process based upon complex numbers.

$$\alpha = arg(\frac{1}{N} \times \sum_{j=1}^{N} exp(i \times \alpha_j)) \tag{D.4}$$

D.4 Convert from WGS84 to ECEF navigation coordinates

The two formulas below (Equation D.5 and D.6) show how to convert WGS84 GPS data to the ECEF Cartesian system. *lon* and *lat* respectively correspond to GPS longitude and latitude.

$$x = lon \times \frac{\pi}{180} \times 6378000 \times \cos(lat \times \frac{\pi}{180}) \tag{D.5}$$

$$y = lat \times \frac{\pi}{180} \times 6378000 \tag{D.6}$$