University of Glasgow

Aerospace Sciences Research Division

College of Science and Engineering

School of Engineering

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

Machine Learning Techniques to Estimate the Dynamics of a Slung Load Multirotor UAV System

Aldo Enrique Vargas Moreno

Supervisors

Dr. David Anderson and Dr. Douglas Thomson

October, 2017

Aldo Enrique Vargas Moreno

Machine Learning Techniques to Estimate the Dynamics of a Slung Load Multirotor UAV System

Thesis Version: 1.5-GA

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

October, 2017

Supervisors: Dr. David Anderson and Dr. Douglas Thomson

University of Glasgow

Aerospace Sciences Research Division

School of Engineering

College of Science and Engineering

University Avenue

G12 8QQ Glasgow, United Kingdom

Preface

This thesis presents work carried out by the author in the Aerospace Sciences Research Division at the University of Glasgow in the period from November 2012 to December 2016. The content is original except where otherwise stated.

Glasgow, United Kingdom, October, 2017

Abstract

This thesis addresses the question of designing robust and flexible controllers to enable autonomous operation of a multirotor UAV with an attached slung load for general cargo transport. This is achieved by following an experimental approach; real flight data from a slung load multirotor coupled system is used as experience, allowing for a computer software to estimate the pose of the slung in order to propose a swing-free controller that will dampen the oscillations of the slung load when the multirotor is following a desired flight trajectory. The thesis presents the reader with a methodology describing the development path from vehicle design and modelling over slung load state estimators to controller synthesis.

Attaching a load via a cable to the underside of the aircraft alters the mass distribution of the combined "*airborne entity*" in a highly dynamic fashion. The load will be subject to inertial, gravitational and unsteady aerodynamic forces which are transmitted to the aircraft via the cable, providing another source of external force to the multirotor platform and thus altering the flight dynamic response characteristics of the vehicle. Similarly the load relies on the forces transmitted by the multirotor to alter its state, which is much more difficult to control. The principle research hypothesis of this thesis is that the dynamics of the coupled system can be identified by applying Machine Learning techniques.

One of the major contributions of this thesis is the estimator that uses real flight data to train an unstructured black-box algorithm that can output the position vector of the load using the vehicle pose and pilot pseudo-controls as input. Experimental results show very accurate position estimation of the load using the machine learning estimator when comparing it with a motion tracking system ($\sim$2% offset). Another contribution lies in the avionics solution created for data collection, algorithm execution and control of multirotor UAVs, experimental results show successful autonomous flight with a range of algorithms and applications. Finally, to enable flight capabilities of a multirotor with slung load, a control system is developed that dampens the oscillations of the load; the controller uses a feedback approach to simultaneously prevent exciting swing and to actively dampen swing in the slung load. The methods and algorithms developed in this thesis are validated by flight testing.

Acknowledgements

Firstly, I will like to thank my wife, team-mate, partner, love and best friend Tania for supporting and being with me since the start of this project. Also, my immediate family, who are another one of my pillars - Rosalba, Raul, Spayro, Gabriela, Juan - and my new family OG for supporting me spiritually throughout the development of this project and my life in general.

Secondly, I would like to express my sincere gratitude to my supervisor Dr. Dave Anderson for the continuous support during my PhD research, for his patience, motivation, and immense knowledge. His guidance helped me throughout the time of my research and writing of this thesis. I could not have imagined having a better supervisor and mentor for this great academic period of my life. Besides my supervisor, I would like to thank my thesis committee: Dr. James Whidborne and Dr. Gianmarco Radice, for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives.

My sincere thanks also goes to the entire academic staff in the Aerospace Sciences Research Division at the University of Glasgow, without they precious support it would not be possible to conduct this research. I would also like to make a special mention to Dr. Tonya Lander (University of Oxford) and Dr. Jacob Apkarian (Quanser) for letting me be part of their group in order to create extremely interesting stuff.

I thank my fellow officemates and friends for the stimulating discussions, experiences and for all the fun we have had in the last four years, inside and outside the University, virtual or face-to-face, close or far. It is a complicated task to name all of you, but a special mention must be made to John, Murray and Victor.

Last, but not the least, I would like to express gratitude to Aldux, that inseparable partner for having contributed wonderful ideas to this project besides being such an entrepreneurial person that inspired me to create AltaX.

Contents

List of Figures

List of Tables

Nomenclature

All units of measurement throughout this thesis conform to the International System of Units (SI), with deviations from this rule noted where appropriate.

Symbol	Description
$\mathbf{v}$	Volts
$\mathbf{v_m}$	Terminal phase voltage
$\mathbf{E_n}$	Motor back EMF
$\mathbf{R}$	Resistance
$\mathbf{L}$	Inductance
$\mathbf{i}_n$	Current
$\mathbf{Q}_m$	Motor Torque
η_m	Motor Efficiency
Ω	Motor rotation rate
$\mathbf{K_v}$	Motor speed constant
$\mathbf{K_t}$	Motor torque constant
$\mathbf{J_R}$	Rotor inertia
$\mathbf{P}_{shaft}$	Shaft Power
$\mathbf{P}_{elec}$	Electrical Power
$\mathbf{T}$	Propeller Thrust
$\mathbf{Q}$	Propeller Torque
$\mathbf{C}_T$	Thrust coefficient based on tip speed
$\mathbf{C}_Q$	Torque coefficient based on tip speed
$\mathbf{J}$	Propeller Advance ratio
$\mathbf{D}$	Propeller Diameter
η_p	Propeller Efficiency
ρ	Air density
μ	Air viscosity
$\mathbf{R}_e$	Reynolds number
$\dot{\mathbf{m}}$	Mass flow rate

$\mathbf{V}$	Flight velocity
$\mathbf{V}_e$	Inflow velocity
$\mathbf{V}_{ac}$	Aircraft velocity
$\mathbf{C}_{batt}$	Battery capacity C-rate
$u_{(1-4)}$	Pseudo-controls
$\mathbf{v}$	Linear velocity vector
ω	Angular velocity vector
$\mathbf{a}$	Linear acceleration vector
α	Angular acceleration vector
ϕ, θ, ψ	Roll, pitch, yaw displacements
$\{x, y, z\}$	Components of position
$\{p, q, r\}$	Angular velocities about quadrotor body axes
$\mathbb{1}$	Identity matrix
I	Inertia matrix
W	World frame
B	Body frame
m	Mass
$\mathbf{F}$	Force vector
τ	Torques vector
Ω	Propeller speed

Abbreviations

Acronym	Description
AHRS	Attitude Heading Reference System
AI	Artificial Intelligence
AL	Autonomy Level
ANN	Artificial Neural Network
APT	Advanced Packaging Tool
BLDC	BrushLess Direct Current
BPTT	Back Propagation Through Time
CCW	Counter-ClockWise
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CoG	Center of Gravity
COM	Center Of Mass
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CSV	Comma Separated Values
CV	Computer Vision
CW	ClockWise
DCM	Direction Cosine Matrix
DOF	Degrees Of Freedom
EKF	Extended Kalman Filter
ESC	Electronic Speed Control
ESN	Echo State Networks
FPP	Fixed Pitch Propeller
GA	Genetic Algorithms
GIL	Global Interpreter Lock
GNC	Guidance Navigation and Control
GPS	Global Positioning System
GPU	Graphics Processing Unit

IMU	Inertial Measurement Unit
IPS	Indoor Positioning System
LCF	Linear Complementary Filter
LiPo	Lithium-ion Polymer
LSM	Liquid State Machine
LTS	Long-Term Support
MAST Lab	Micro Air Systems Technology Laboratory
MEMS	Micro-ElectroMechanical Systems
ML	Machine Learning
MLP	Multi-Layer Perceptron
MoCap	Motion Capture
MRUAV	Multirotor Unmanned Aerial Vehicles
MSE	Mean Square Error
MSP	MultiWii Serial Protocol
NED	North-East-Down
PWM	Pulse Width Modulation
RC	Reservoir Computing
RF	Radio Frequency
RLS	Recursive Least Squares
RNN	Recurrent Neural Network
ROS	Robot Operative System
RTOS	Real Time Operating System
RTRL	Real Time Recurrent Learning
RUAS	Rotorcraft Unmanned Aerial Systems
RUAV	Rotorcraft Unmanned Aerial Vehicle
SAS	Stability Augmentation System
SL	Slung Load
SVM	Support vector machines
TDL	Tapped Delay Lines
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UUV	Unmanned Underwater Vehicles
VTOL	Vertical Take-Off and Landing

1 Introduction

This chapter presents the background and motivation for this PhD study and discusses the application of Multirotor Unmanned Aerial Vehicles (MRUAV) slung load flight and the problems associated with this. It further introduces the research question, hypothesis and methodology. Continued by a literature review of the current state of the art of the related topics. Then the contributions of this study are presented. Finally an outline of the thesis and publications is given.

1.1 Background and motivation

Once only seen as a niche military asset (Benjamin, 2013), Unmanned Aerial Vehicles (UAVs) have become an indispensable resource at the forefront of force projection strategies for defence stakeholders around the world. However, UAV technology is also now being embraced by the commercial and academic sectors and as a consequence there has been a significant surge in UAV research projects focussed on commercial exploitation opportunities. There are now many examples throughout the literature of UAVs being designed and used for operations beyond the military, such as aerial photography (Cheng, 2015), infrastructure inspection (Chan et al., 2015), law enforcement (Brumfield, 2014), littoral maritime surveillance (Abhijit, 2016), road traffic monitoring (Kim et al., 2015), disaster and crisis management (Apvrille et al., 2015) and agriculture and forestry (Salamí et al., 2014). This thesis will present the results of a research project investigating and proposing novel solutions to some of the challenges that must be overcome to properly realize the potential of commercial UAV operation.

1.1.1 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles, also commonly called *drones*, historically started as an expendable military asset used for target practice (Benjamin, 2013). From the 1960's onward, the potential for using drones for reconnaissance purposes became apparent and this visionary leap ultimately led to the highly varied operational roles undertaken by modern UAVs. Due to the secretive, classified and potentially clandestine nature of military surveillance operations, their technological development was often kept out of sight of the public.

However, the highly publicised use of drone technology in recent global conflict sparked an intense interest in UAV technology among academics and commercial entrepreneurs alike. As the enabling technologies become more advanced and costs fall, civilian exploitation of drones is developing rapidly. However, when speaking about drones dedicated for civil use, it is important to distinguish between the large, civil vehicles that might one day carry passengers without on-board human supervision (Ehang Inc, 2016), regular drones of similar size to those used in the military and much smaller systems in the small or micro-UAV classification (determined by either the wingspan or rotor diameter, below 1.5m is small and below 20cm is considered micro). By far the most popular configuration in the small/micro class is the quadrotor (Fig. 1.1). Small drones are typically designed to be man-portable and relatively affordable, e.g. converted radio-controlled (RC) planes or simple, small multirotor platforms. When evaluating operational tasks currently performed

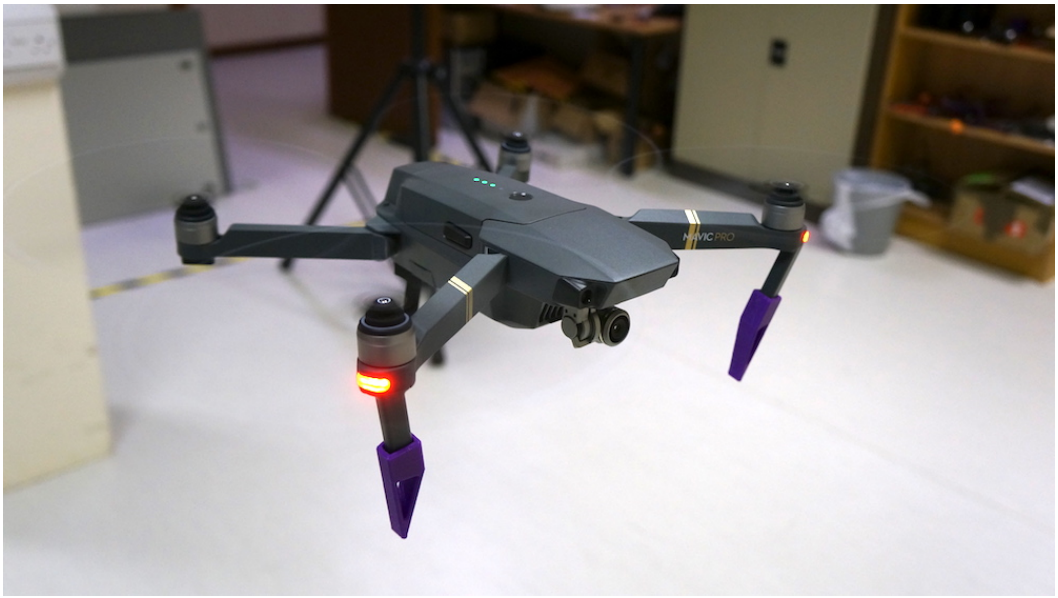


Fig. 1.1.: Small quadrotor - DJI Mavic Pro.

by human operators (usually pilots) for transfer to drone technology, such tasks are typically classified as being either **Dull**, **Dirty** or **Dangerous**. The "dull" classification refers to monotonous, typically long-duration tasks that become uncomfortable for a human operator to perform, a long-range surveillance flight for example. For commercial drones in this classification, some of the most popular emerging applications include the transport of goods (also know as cargo, payload or *load*) (Thiels et al., 2015), conservation and wildlife measurement (Barmpounakis et al., 2017) or agriculture and aerial photography. A "dirty" task is one deemed unpleasant for a human operator to perform such as flight in close-proximity to noxious chemical spills. Finally, the "dangerous" classification is self-explanatory - use of drone technology to keep human operators out of harms way. All three classifications are equally applicable to both civilian and military operations. At this time

there remain many technical limitations in both platform capability and on-board systems preventing widespread adoption of UAVs as commercial assets. Although military-grade UAVs posses platform capabilities that meet or exceed their fixed-wing counterparts, they remain prohibitively expensive for all but the most specialised commercial application. Both purchase and operation costs have to be reduced before drones can enter the mainstream, consequently it is the small-micro-UAV class of drones that is attracting the most interest from companies such as (Amazon, 2013) and (Google, 2014). Smaller aircraft provide the opportunity for operating in urban areas, but at the cost of reduced payload capacity and endurance. Therefore maximizing aircraft performance whether is be lift to drag ratio or rotor thrust becomes a critical technology enabler.

There are also legislative barriers to civilian UAV operation for commercial gain due to a lack of understanding of the capabilities offered by state-of-the-art avionics suites for each class of drone, particularly those in the small/micro-UAV class (Friedenzohn et al., 2013). Smaller/cheaper drones require cost-minimal avionics and on-board sensor suites, while simultaneously having to prove to national certification authorities an unprecedented degree of autonomy and control precision of an integrated system with only antiquated analysis techniques. From an academic perspective, these technical challenges provide a number of interesting research opportunities.

Although there are many operational scenarios currently performed only by manned aircraft, one of the most challenging from a research perspective is the helicopter slung load operation.

1.1.2 Cargo transport application

In cargo transportation operations, if the load being transported is carried inside the vehicle fuselage, a special mechanical design must often be added to the airframe in order for it to transport the load safely and securely. This situation limits the type of vehicle airframe and the shape of the transported cargo. In contrast, if the load is outside of the vehicle fuselage it opens a wide range of aircraft-based transportation applications. Transporting a load external to the vehicle fuselage is usually accomplished by attaching the load to the underside of the vehicle via cables or ropes. At present this type of cargo transportation mechanism is almost exclusively by manned helicopter platforms.

Helicopter slung load operations with the load suspended in various ways from a single attachment point have been common since the 1950's (Cicolani et al., 1995). Since then helicopters have been used for a vast number of different towing assignments ranging from fire-fighting applications over animal transport to container-hauling. With a manned air-

Credit: *Capital Press*

Fig. 1.2.: Helicopter *swinging* a slung load.

craft, slung load operations require a highly skilled pilot due to the flight dynamics of the aircraft being coupled with the load swinging when the aircraft has to manoeuvre. Such an application is called *flying crane*, where helicopters carry loads connected to long cables or slings in order to place heavy equipment when other methods are not available or economically feasible, or when the job must be accomplished in remote or inaccessible areas, such as the tops of tall buildings or the top of a hill or mountain, far from the nearest road. One clear example of the difficulty of flying a helicopter with a slung load attached to it can be seen during the *Oregon Christmas Tree harvest* [1] (Fig.1.2) where the pilot learns to use the swinging of the load in order to deposit the cargo (trees) as fast as possible in the loading truck. Pilot Dan Clark has been quoted[2] as saying that the technique took him approximately 10 years to master. Anecdotal evidence of this type suggests that conventional fixed-controller strategies may not be sufficient, rather that a learning controller would be required to replace the pilot's actions.

[1] `https://www.youtube.com/watch?v=08K_aEajzNA`
[2] `http://www.thenorthwestreport.com/less-travel-could-boost-oregon-christmas-tree-sales/`

Dynamically, attaching a load via a cable to the underside of the aircraft alters the mass distribution of the combined "*airborne entity*" in a highly dynamic fashion. The load will be subject to inertial, gravitational and unsteady aerodynamic forces which are transmitted to the helicopter via the cable, providing another source of external force to the helicopter platform and thus altering the flight dynamic response characteristics of the vehicle. Similarly the load relies on the forces transmitted by the helicopter to alter it's state, i.e. we have moved from a single to two-body system, which is much more difficult to control. Therefore, from an academic perspective, the combination of intricate coupled dynamics and the need for a learning control architecture shows there is significant research opportunity in addressing the challenges of operating unmanned aerial vehicles with slung loads.

1.1.3 Research question & hypothesis

As has been shown there are significant commercial, military and humanitarian benefits to operating autonomous UAVs as cargo-carrying platforms. However matching the capabilities of manned platforms, especially in complex cargo-carrying configurations such as the slung load, still presents significant technical challenges. The central research question posed and addressed by this thesis is then

> *Is it possible to design robust and flexible controllers to enable autonomous operation of a multirotor UAV with an attached slung load of unknown mass and geometric distribution*?

The first step in designing any controller is to construct a mathematical model of the equations of motion of the system. Adding additional mass alters, at the very least, the values of the parameters in the equations of motion that define the model - gains, time constants, mode coupling etc. In control theory, there are two approaches to dealing with this problem: treat it as an uncertainty in the feedback loop and apply a robust controller synthesis technique such as H^{∞} or use an adaptation mechanism to alter the underlying mathematical model and controller. For flight with a suspended load the primary impact of adding the load is to induce lateral pendulous oscillations, which can become unstable. This prominent pendulous oscillatory motion affects the response in the frequency range of the attitude control of the vehicle. Therefore, a fundamental understanding of the dynamics of slung loads as they relate to the vehicle handling is essential in developing effective flight controllers. Finally, given the anecdotal evidence of helicopter pilots and the wide variability in cargo parameters, it is highly unlikely that a robust control strategy would be effective.

Following a thorough literature review and detailed consideration of this problem, the principle research hypothesis of this thesis is that the dynamics (and ultimately control) of

the slung load / MRUAV coupled system can be identified by applying Machine Learning techniques.

Machine Learning addresses the question of how to build computer software that improves automatically through experience. Recent progress in Machine Learning has been driven by the development of new learning algorithms that use experimental data and low-cost computation. One of the most commonly known machine learning subsets is Artificial Neural Network (ANN), inspired by the structure and functional aspects of biological neural networks. The Recurrent Neural Network (RNN) is a class of ANN that represents a very powerful generic system identification tool, integrating both large dynamic memory and highly adaptable computational capabilities. Reservoir Computing (RC) is another approach to design, train, and analyse RNNs. The main advantages of this paradigm are modelling capacity and accuracy, biological plausibility and their extensibility and parsimony. RC has outperformed previous methods of non-linear system identification, prediction and classification (Sec. 5.6). This is one of the paramount capabilities needed in the presented research.

1.1.4 Research methodology

In this thesis the problem of a MRUAV flying with a slung load (Fig. 1.3) is addressed. This is going to be achieved following an experimental approach. Real flight data from the MRUAV/SL system is used as experience, allowing for a computer software to understand the dynamics of the slung in order to propose a swing-free controller that will dampen the oscillations of the slung load when the MRUAV is following a desired flight trajectory. In order to answer the hypothesis of a multirotor flying with a slung load using an experimental method several objectives must be met before. Those objectives include the design of a multirotor test-bed with the capability of carrying a slung load with sufficient extra thrust to be able to manoeuvre while maintaining the maximum time of flight possible and low costs. The laboratory environment must be set-up to facilitate the data gathering process. A proper avionics suite (hardware and software) must be be created and used to the purpose of developing and testing advanced autonomous GNC (Guidance Navigation and Control) algorithms. Then modelling, control and experimental testing of multirotor must be addressed and validated. Several machine learning techniques must be explored, tested and validated. Finally a control system must be created so that it dampens the oscillations of the load during and at the end of transport.

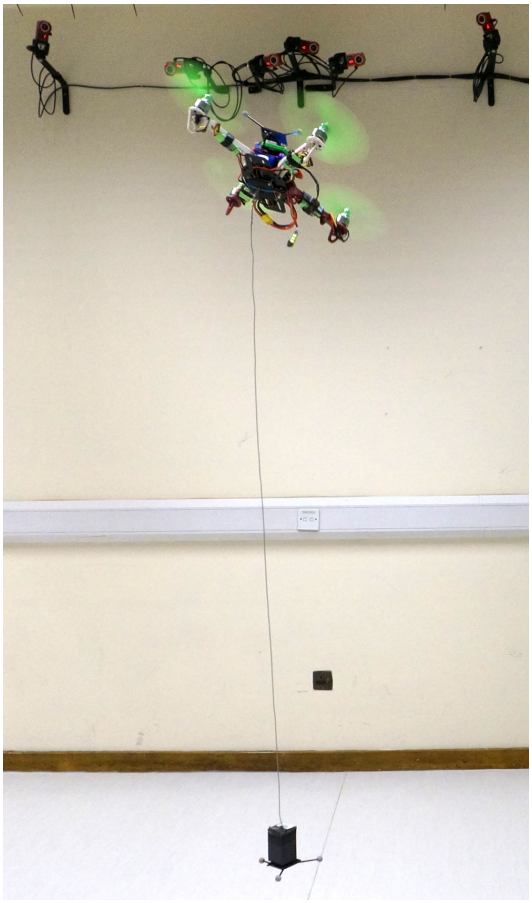


Fig. 1.3.: MRUAV carrying a slung load while manoeuvring.

1.2 Literature review

This section gives an overview of the state of the art of several fields of the research area for this thesis. Such fields include the Unmanned Aerial Systems, quadrotor modelling and control, slung load dynamics and the machine learning field.

1.2.1 MRUAV

Recently, there is a growing interest in developing Unmanned Aerial Systems (UAS) with advanced on-board autonomous capabilities (Kendoul, 2012). Unmanned aerial vehicles are encountered in an increasing number of applications, mostly military but increasingly in the civilian market. Most of the civilian applications require low-altitude flights with hovering and VTOL capabilities, and small rotorcraft UAV are more appropriate for these applications than fixed-wing UAV. Furthermore, most academic and governmental research groups use rotorcraft UAV as experimental platforms to validate their GNC algorithms (Richards et al., 2002) (Shim et al., 2002) (Valenti et al., 2006). A particular advantage an MRUAV has over

other aerial vehicles is its unique ability for vertical stationary flight (hovering) (Ampatis et al., 2014).

Research activities require a robust yet simple design to aid in validation of their GNC algorithms. The use of a single rotor vehicle for such activities would require complicated electro-mechanical components to accurately reposition the rotor and allow changes on the attitude and position of the UAV. Multi rotorcraft offer a solution to simplify the components required to manoeuvre the UAV. Since there is no need for extra electro-mechanical components to reposition the rotor, the only requirement is to pair two or more rotors in order to balance the generated torque. An exception is the trirotor, where one rotor is placed on a tilting mechanism that balances the additional toque.

In recent years, advances in materials, electronic components (Jang et al., 2004) (Ede et al., 2001) (Sai Dinesh et al., 2010), sensors (Burghartz, 2013) and batteries (Tarascon et al., 2001) have fuelled a growth in the development of multirotor aerial vehicles. Therefore, such vehicles are now seriously considered as practical and robust test-beds. The design, construction, and testing of a MRUAV test-bed requires a great amount of time (Michael et al., 2010) and it is one of the core research goals of this thesis.

The most common multirotor platform is the *quadrotor*, but additional multirotor platforms are easily conceived by augmenting the vehicle with additional rotors. This provides greater lifting force without the need to upgrade components, such as the rotor blades and motors. However, the introduction of additional rotors changes the dynamic performance of the system as shown in (Ireland et al., 2015), adding extra mass and typically requiring additional supporting structure.

Optimal selection of the most important components of the multirotor vehicle is critical in order to obtain the necessary performance from the test-bed to succeed in validating the advanced algorithms. In the specific case presented in this thesis, the vehicle will be optimized to carry a slung load with adequate flight times that are necessary to acquire all data for the machine learning experiments. Despite the large number of published works on multirotors, very little is published on design of multirotors given an intended application and desired performance specifications. The most relevant methodologies are presented in (Magnussen et al., 2014) (Magnussen et al., 2015).

As a consequence of the technological advances in sensors, specifically in MEMS (Microelectromechanical systems), the development of specialized computers to handle MRUAV flight control has grown significantly. Such devices are called *flight controllers* and are

capable of computing the necessary rotor speeds to keep the vehicle stable and flying, based on on-board sensors.

There are several publicly available open-source flight controllers. One of the most successful in academia is the Pixhawk (Meier et al., 2011) project. Great success has been shown as well from the MultiWii (MultiWii, 2010), Arducopter (*Arducopter*), OpenPilot (OpenPilot, 2011), Paparazzi (Paparazzi, 2003) (Bronz et al., 2009) and MikroKopter (HiSystems, 2006) projects, among others. It is important to note that each FC has a different level of autonomy based on the sensors and devices being connected to it.

1.2.2 Quadrotor Modelling and Control

The study of the quadrotor's kinematics and dynamics helps to understand its physics and behaviour. Together with the modelling, the determination of the control algorithm structure is fundamental to achieve optimal stabilization.

As mentioned earlier, the quadrotor is a very popular research platform. Therefore, the number of projects tackling the modelling and control has considerably and suddenly increased, with the most relevant works presented in (Bouabdallah, 2007) (Bresciani, 2008) and (Mellinger, 2012). In classical mechanics, the Newton–Euler equations describe the combined translational and rotational dynamics of a rigid body (Mahony et al., 2012a) (Hahn, 2002) (Bishop, 2007) (Featherstone et al., 2000).

After dealing with the kinematic and dynamic modelling, the on-board estimation of orientation (attitude) and heading needs to be addressed. The more accurate results are obtained by combining data from multiple types of sensors to take advantage of their relative strengths, such technique is called *data fusion*, also known as sensor fusion. There is a large academic and commercial activity due to the variety of new on-board sensors that the MRUAV can carry such as new generation MEMS, LIDAR, SONAR, 3D cameras, GLONASS, i.a. Data fusion refers to a variety of techniques, technologies, systems, and applications that use data derived from multiple information sources (Elmenreich, 2002). Fusion applications range from real-time sensor fusion for the navigation of mobile robots to the off-line fusion of human or technical strategic intelligence data (Rothman et al., 1991).

Pixhawk project have designed an attitude estimation algorithm based on the Extended Kalman Filter (EKF). An explanation of the algorithm can be found in (Meier et al., 2011) and (Simon, 2006). In the other reviewed flight controller (MultiWii), a non-linear complementary filter is implemented with the rotation matrix representation, based on (Mahony et al., 2005), with the only difference being that MultiWii leaves the corrections of errors

in the rate integration due to the gyroscope imperfections to the effectiveness of the complementary filter. A full description of a complementary filter can be seen in (Oliveira et al., 2000).

Regarding stabilization or attitude control, the most used linear regulators in the MRUAV field (COTS and hobby grade) are PID (Schiavoni et al., 2015). By design, the quadrotor is simply controlled by independently changing the speed of the four rotors (Beard, 2008) (Adigbli et al., 2007). The dynamics of the quadrotor are considered unstable (Miller, 2011), with the characteristics of dynamics such as being intensively non-linear, multi-variable, strongly coupled, and under-actuated. Therefore a feedback control topology is needed (Stevens et al., 2003) due to the fact that the model reveals that the poles are located on the right of the real-imaginary plane and its damping ratio is negative. One of the most commonly used techniques (Nelson, 1998) in order to provide a solution is a Stability Augmentation System, which makes the vehicle stable via the rate measurement in the feedback loop. (Bouabdallah, 2007) derives the quadrotor model in the Euler–Lagrange formalism and applies it in a comparison of PID and LQ control methods. The inner attitude loop control is performed inside the flight controller, using accelerometers, gyroscopes and it runs approximately at 286Hz on the MultiWii board (MultiWii, 2010) and 400Hz on the Pixhawk flight controller (Meier et al., 2011).

In this thesis, a position control strategy that uses the pseudo-controls throttle, roll, pitch, yaw (derived from the attitude controllers mentioned above) for station-keeping or maintaining the position at a desired location is presented. There are similar approaches in (Mellinger, 2012) and (Khalil, 2002). This approach will be later used by the trajectory controller as well. (Michael et al., 2010) (Mellinger et al., 2014) describe several linear PID controllers for tracking trajectories. In (Mellinger, 2012) a quadrotor model with non-linear rigid body dynamics and a first-order rotor response is linearised with the rotor dynamics neglected in order to obtain the control law for orientation and, therefore, the position of the vehicle.

One of the main differences between this thesis and other similar approaches is the use of distributed on-board computing for the control structure (Flight Stack Sec. 3.3). In (Heng et al., 2011) (Honegger et al., 2012) only one computer is used in order to validate their advanced GNC algorithms, while in (Mellinger et al., 2014) (Mahony et al., 2012b) the vehicle was controlled using a ground station desktop computer.

If we consider the concept of distributed computing (Coulouris et al., 2012), where components interact with each other in order to achieve a common goal, a more robust and functional system can be obtained. By adding a companion computer to the avionics system,

we can increase the available functionality and, consequently, the computational resources available for running guidance and navigation algorithms are considerably increased. This system is named *Flight Stack* (Vargas et al., 2016).

1.2.3 Slung Load Dynamics

Over the last 50 years, applications of rotorcraft carrying external suspended loads have been of significant interest in the aerospace research community due to the inherent stability problems such systems suffer from. Rotorcraft suspended load operations have experienced further development and extensive use since the Vietnam war. In the following years (1965-1975) the research turned to the stabilization of difficult loads using heavy lift helicopters (Cicolani et al., 1995). The solutions found included suspensions with multiple attachment points and various control devices. An early successful operation using single helicopter slung load was based on using suspensions consisting of cables and spreader bars (Korsak et al., 1972).

A common obstacle to further operational development is the complexity of the system motion and its guidance and control along manoeuvring flight paths (Cicolani et al., 1986), which somewhat slowed the progress of slung load operations until recently. Progress beyond hover operations suffered due to the lack of practical and realistic equations of motion for use in simulation studies and, therefore, the development of experimental studies was affected as well.

In the work of (Cicolani et al., 1995) several systematic approaches to derive the equations of motion of the slung load system were identified. Their working equations for applications were formulated almost entirely in terms of the objects and operations of 3-dimensional vector mechanics, their simulation work was demonstrated and now it is used on a number of similar studies.

Another successful example of simulation of single helicopter slung load operation is presented in (Faille et al., 1995). They studied the stabilization and regulation problem of a helicopter/slung-load system by using a non-linear model of the system that was linearised (only a set of equations) in order to apply linear control theory. In their model, the load needs 9DOF (degrees of freedom) that corresponds to 18 states and, when connected to helicopter, the model contains 33DOF resulting in 66 states. An assumption is made that it is possible to measure the position and orientation of the slung load, therefore this methodology only works in simulation scenarios. Finding information about the current state of the slung load becomes paramount if the control of the load must be addressed. This work

is important for historical reasons, since it proposes a practical methodology to tackle the problem.

For an elaborate survey of the different studies in this area throughout the last decades the reader may refer to the work by (Fusato et al., 2001) and (Luigi, Cicolani, 1992), containing a number of historical references dealing with different types of stability analysis. Two major works are found in (Prasad Sampath, 1980) and (Bisgaard et al., 2006). In the first one, the author used a Lagrange formulation to tackle the modelling problem, being this one of the first studies that created a complete set of equations of motion in 12 degrees of freedom, which included all body-to-body suspension schemes. In the second study, the model was derived using the Udwaidia-Kalaba equation and a redundant coordinate formulation in which the wires were inserted as acceleration constraints.

The majority of studies have been focused on determining stable flight regimes with respect to slung load parameters to avoid instabilities (Prasad Sampath, 1980) (Poli, 1973) (Prabhakar, 1977). Alternative efforts have considered modifying the shape of the load (Hoh et al., 2006) as well as adding extra components (gyroscopes, fins, drogues) to the load to make it stable related to the rotorcraft (Micale et al., 1973) (Feaster et al., 1977). Adding components of the load is a possible solution, but reduces the applicability and practicability of the overall system, therefore it has to be pondered according to the application.

The problem of state estimation for slung load systems is mentioned sparsely in literature. In (Dukes, 1973) the difficulty in reliably estimate slung load states is mentioned and to overcome this problem an open loop control approach is suggested. (Gupta et al., 1976) considers the design of state estimation for the slung load using an attitude measurement, the angles of a measurement cable from the helicopter to the ground and the angles of the suspension cable as sensor input to a linear Kalman filter.

One particularly relevant study was focused on designing stability augmenting techniques for slung load systems and stability analysis to determine favourable wire lengths, vehicle/load mass ratios, and other parameters (Bisgaard et al., 2006). This study resulted in one of the first experimental systems for small scale rotorcraft. Autonomous small scale rotorcraft have changed the perspective on this field of engineering, making it more accessible for academic research groups and allowing to tackle the problems related with slung load systems. Before this seminal work by Bisgaard, testing of stability augmenting solutions on real aircraft was limited to research and development groups with access to heavy lift, expensive helicopters (military and defence companies).

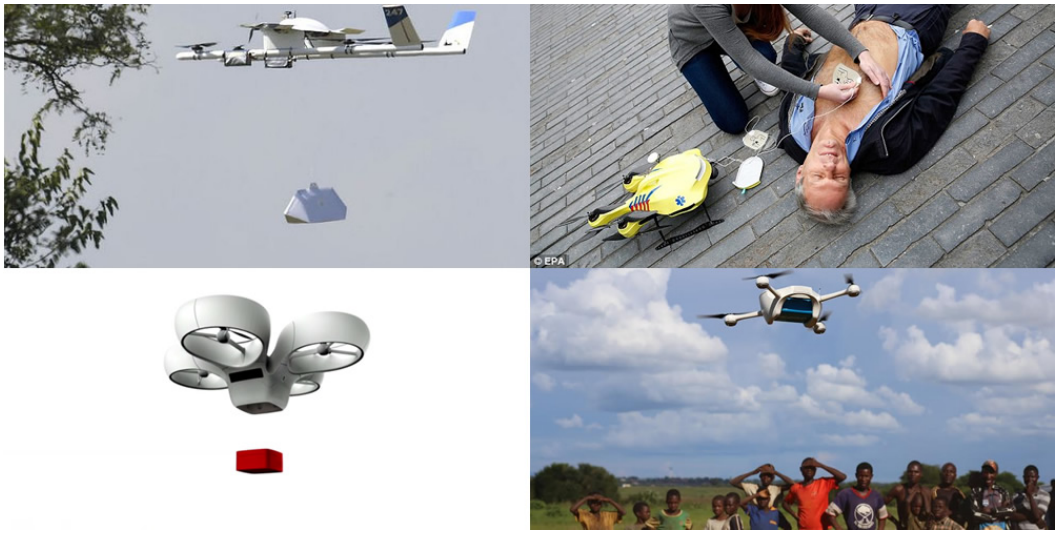
In more recent work (Bisgaard et al., 2010) presents a design and verification of an estimation and control system for a helicopter slung load system. In this work, Bisgaard uses a computer vision approach to create an estimator capable of estimating the states of the load. Vision-based sensor data estimates the relative position of the load and estimates the wire length. After testing the estimator, a feed-forward control system based on input shaping is developed so that it enables the helicopter to perform manoeuvres with a slung load without inducing residual oscillations. An estimator for the slung load position inspired by the latter work is created for this thesis.

Tackling the slung load problem using a multi rotorcraft is found in more recent work. (Palunko et al., 2012) addressed the problem of trajectory tracking while carrying a slung load, using dynamic programming in order to generate a swing-free trajectory for the quadrotor slung load system. The estimation of the slung load states is done by using an indoors motion tracking laboratory.

In (Palunko et al., 2013) a model-free approach to solve the slung load swing trajectory tracking using a reinforcement learning algorithm is proposed. The slung load states are obtained using an indoors motion tracking system. Their method converges quickly to learn the policy function that minimizes the tracking error of the load with respect to the reference trajectory, their results are proved experimentally.

Another approach of simulation work on dynamic modelling of the quadrotor/slung load system can be found on (Sadr et al., 2014), the model was obtained and verified by comparing two Newton–Euler and Lagrange methods. Their control methodology involved a feed-forward algorithm for reducing or cancelling the swinging load oscillation by implementing input shaping theory which convolves the reference command with a sequence of impulses, no experimental work was carry out. A interesting modelling approach for the slung load was presented in (Feng et al., 2015), where they modelled as a three-dimensional point mass pendulum where the dynamics of the slung load are constructed analytically by calculating the suspension angles of the load. An adaptive control scheme is then proposed, it addresses specifically the existence of the external force and torque caused by the slung load. Both modelling and control techniques are verified only with simulations.

A hybrid dynamical system is proposed in (Tang et al., 2015) where the quadrotor is considered as a rigid-body and the load as a point-mass, but their hybrid model comes from two subsystems of models, their numerical and experimental results indicate that the method is practical for generating trajectories that include aggressive obstacle avoidance manoeuvres and hybrid state transitions. The slung load orientation and position is obtained by using an indoors motion tracking laboratory. Attempts using drones in the cargo transport can be



Credit: *Raptopoulos, Delft and Momont*

Fig. 1.4.: Examples of cargo MRUAV on the healthcare industry.

from from a range of sectors, including the healthcare industry, food, and postal deliveries. In the healthcare industry drones can transport medicines and vaccines, and retrieve medical samples into and out of remote or otherwise inaccessible regions (Andreas Raptopoulos, 2013) (Fig.1.4 bottom left and bottom right). *Ambulance drones* are developed to rapidly deliver defibrillators in the crucial few minutes after cardiac arrests (Alec Momont, 2014) (Fig.1.4 top right).

Foodwise, there are currently several companies creating delivery services for their food goods. The most relevant are *Burrito-by-drone* (Fig. 1.5 top left), *Domino's Pizza* (Fig.1.5 top right), *7-eleven* (Fig.1.5 bottom left) and *Old Hamburg Schnitzelhaus AIR* (Fig. 1.5 bottom right). In the postal delivery sector, postal companies have been forced to seek new ways to expand their traditional letter delivery business models. Different postal companies from Australia, Switzerland, Germany, Singapore and Ukraine have undertaken various UAV trials as they test the feasibility and profitability of unmanned delivery UAV services. Again, the most relevant are Amazon, 2013 (Fig.1.6 top middle), DHL, 2013 (Fig.1.6 bottom left) and Google, 2014 (Fig.1.6 bottom right). After reviewing the literature, a very important need for an experimental slung load estimator is found. Such an estimator can then be used on different types of control strategies.

1.2.4 Machine Learning

Recent advances in Machine Learning can be used to solve a tremendous variety of problems and Deep Learning (LeCun et al., 2015) is pushing the boundaries even further. Data and analytic capabilities have made a leap forward in recent years. The volume of available

Credit: *Burrito-by-Drone, Domino's pizza, 7-eleven, and OHS*

Fig. 1.5.: Examples of cargo MRUAV on the food industry.

data has grown exponentially, more sophisticated algorithms have been developed, and computational power and storage have steadily improved.

There are several applications for Machine Learning, the most significant of which is predictive data for different fields (engineering, science, finance, entertainment) (Mitchell, 1997). Conventional software programs are hard-coded by humans with specific instructions on the tasks they need to execute. By contrast, it is possible to create algorithms that *learn* from data without being explicitly programmed (Shavlik et al., 1990). The concept underpinning machine learning is to give the algorithm a massive number of *experiences* (training data) and a generalized strategy for learning, then let it identify patterns, associations, and insights from the data. In short, these systems are trained rather than programmed.

Some machine learning techniques, such as regressions, support vector machines (Cortes et al., 1995), and k-means clustering (MacQueen, 1967), have been in use for decades. Others, while developed previously, have become viable only now that vast quantities of data and unprecedented processing power are available. Recurrent neural networks (Boden, 2001), a frontier area of research within machine learning, uses neural networks with many layers to push the boundaries of machine capabilities.

ML techniques can be useful for solving dynamic optimization and control theory problems, which are exactly the type of issues that come up in modeling complex systems in fields such as engineering and economics (Wagstaff, 2012). Therefore combining ML with other tech-

Credit: *Amazon, DHL and Google*

Fig. 1.6.: Examples of cargo MRUAV on the postal industry.

niques, could have an enormous range of uses. Acquired knowledge (inductive algorithms) (Kocabas et al., 1991) can be used to detect patterns, trends and structure in many domains. This source discusses learning at different levels which are knowledge, symbol and device. While (Dietterich, 1986) concludes there are two types of learning; knowledge-level which is the same as knowledge acquisition and symbol-level which is speed-up learning where knowledge is used more efficiently.

In the symbol-level, also called symbolic learning, the basic fields involve learning by being told, deduction and induction (Kocabas et al., 1991). The latter is characterised as *learning as search* (Mitchell, 1982). One of the induction techniques used in this thesis is the so called supervised induction, which relies upon a set of pre-classified examples which are ideally indicative of the concept that it is to be learned, it is also called supervised learning. It involves an algorithm that learns to discriminate between the given concepts (Carbonell et al., 1990).

As explained above, inductive machine learning is the process of learning a set of rules from instances (examples in a training set). A large number of techniques have been developed to

tackle inductive learning such as logical/symbolic techniques, perceptron-based techniques and statistics (Bayesian Networks, Instance-based techniques) (Kotsiantis, 2007).

In the logic based algorithms, two methods stand out, decision trees and rule-based classifiers. (Urnkranz, 1999) provides an overview of work in decision trees with practical applications while (Urnkranz, 1999) delivers an excellent overview of existing work in rule-based methods. For the perceptron-based techniques the most relevant methods include single and multi layered perceptron which are based on the notion of perceptron (ROSENBLATT, 1961), such techniques are also called artificial neural networks.

Artificial neural networks are reviewed in (Rumelhart et al., 1986b) (Blum et al., 1992) (Sethi, 1990). The latter describes nets with two hidden layers as being able to form decision boundaries of any complexity. During learning, neural networks distribute a representation across many units, or may dedicate neurons to individual subtasks. There are several algorithms with which a network can be trained (Neocleous et al., 2002). However the most well-known and widely used learning algorithm to estimate the values of the weights of a network is the Back Propagation (BP) algorithm (Rumelhart et al., 1986a). Feed-forward neural networks are usually trained by the original back propagation algorithm or by some variant. Their greatest problem is that they are too slow for most applications.

A recurrent neural network (RNN) is an artificial neural network whose neurons send feedback signals to each other. A large number of reviews already exist of some types of RNN, the most relevant being (Williams et al., 1989) (Jaeger, 2005a) (Jaeger, 2001) (Chow et al., 1998). Many applications using RNN have addressed problems involving dynamical systems with time sequences of events.

Recurrent neural networks are being used to track water quality and minimize the additives needed for filtering water, the studies of time sequences of musical notes, applications focusing on systems for language processing, real-time systems, trajectory problems, and robotic behaviour (Jain et al., 2000). This wide range of applications make it a favourable candidate for the task of this thesis which is to simulate, predict, classify and control a non-linear dynamical system (MRUAV/slung-load system). Sometimes it is hard to obtain an analytical description of the system (system model) and therefore a solution is to use black-box modeling techniques. RNN have shown outstanding performance in such tasks (Holzmann, 2009) (Gonzalez-Olvera et al., 2010) (Romero Ugalde et al., 2013).

Training a RNN is inherently difficult (Pascanu et al., 2012) and there are two widely known issues with the training process, the vanishing and the exploding gradient problems which are detailed in (Bengio et al., 1993). However RNNs, represent a very powerful generic

tool, integrating both large dynamical memory and highly adaptable computational capabilities. They are the Machine Learning (ML) models used in this thesis and are the most closely resembling of biological brains, the substrate of natural intelligence. In order to overcome the downsides of traditional RNN training such as Back Propagation Through Time (Campolucci et al., 1996) and Real Time Recurrent Learning (Jaeger, 2005b), a novel paradigm of computation with dynamical systems, namely Reservoir Computing (RC) has been proposed (Verstraeten et al., 2007) which can be utilized to achieve efficient training of RNNs.

Reservoir computing has emerged as an alternative to gradient descent methods for training recurrent neural networks. The main aspect of an RC network is that the recurrent connections of the reservoir are fixed, while the readout output weights only are trained. This characteristic simplifies much of the training of recurrent networks, as any standard classification or regression method can be used to train the output layer.

In (Williams et al., 1989) (Werbos, 1990) (Puskorius et al., 1994), the used algorithms adapt all connections (input, recurrent, output) by some version of gradient descent which makes them perform slowly, as well as the learning process being prone to become disrupted by bifurcations (Doya, 1992) which means convergence cannot be guaranteed, therefore RNNs were rarely fielded in practical engineering applications. In RC algorithms, training is fast, does not suffer from bifurcations, and is easy to implement. On a number of benchmark tasks, RC algorithms have outperformed several other methods of non-linear dynamical modelling as shown on (Jaeger et al., 2004) (Jaeger, 2007).

Echo State Networks (ESN) are a *flavour* of RC (Lukoševičius et al., 2009). The main idea comes from a continuous neural hardware micro-circuitry. ESNs have the advantage of overcoming the difficulties of traditional dynamic RNN in large-scale training. They can also approximate non-linear systems precisely producing excellent results in their predictions. The ESN is practical and conceptually simple, but requires some experience and insight to achieve good performance.

The ESN idea is shared with Liquid State Machines (Maass et al., 2002) (Yamazaki et al., 2007), both were developed independently, but simultaneously. Both techniques, as well as back-propagation decorrelation learning rule (Steil, 2004), are considered flavours of the reservoir computing framework.

For the dynamic modelling of systems, (Antonelo, 2011) created reservoir computing networks (particularly ESN) trained to predict the position of the robot from the sensory signals to then create forward models of ground robots, it was found that it is possible to use the

network in the opposite direction for predicting local environmental sensory perceptions from the robot position as an input, thus learning an inverse model. All of the proposed models were used on robot navigation systems to be able to safely and purposefully navigate in complex dynamic environments.

(Ploger et al., 2004) applied RNN and ESN to generate a dynamical model for a differential drive robot using supervised learning and secondly to the training of a respective motor controller, proving that ESN can be implemented in the actual hardware of the motor controller. (Ni et al., 2011) used ESN for the application of aircraft predictive control by forecasting the attitude control signal to further enhance the aircraft's flying qualities, proving that RNNs have an excellent non-linear approximation, memory and predictive ability.

A very interesting approach using neural networks to tackle slung load problems is found in (De La Torre et al., 2013a). They proposed a neuro-predictive trajectory generation architecture for slung load systems using a system uncertainty identifying neural network. It is shown that the effect of system uncertainty on a model predictive control approach can be mitigated by the use of neural networks.

While discussing the system identification, (Gonzalez-Olvera et al., 2010) presented a new recurrent neuro-fuzzy network for modeling and identification of a class of non-linear systems using only output measurements, with an algorithm based on the adaptive observer theory. Their trained network effectively learned the dynamics of the system on two examples based on physical systems with experimental data (a visual servoing system and a traffic cell).

In (Jaeger, 2002b), the creator of the ESN flavour shows basic ideas and examples on how to use the Recursive Least Squares (RLS) algorithm in combination with ESN to identify a 10th order NARMA system. (Bian et al., 2011) used ESNs to improve the accuracy of a 6DOF model for Unmanned Underwater Vehicles (UUV), which are a highly complex non-linear dynamic systems, by using a *meta-learning* strategy for off-line training and genetic algorithms to optimize the main parameters, with their results showing that the training strategy is simple, efficient and easy to operate.

A neural network system identification technique is used in (Shamsudin et al., 2010) to model the dynamics of a small scale helicopter where the test data is from a non-linear dynamics simulator. In (Liu, 2001) several techniques and training approaches were developed using RNN for applications to non-linear dynamical system identification, concluding that the training process of RNN can be simplified due to the simple gradient calculation

in feed-forward neural networks and that the RNN structure considered is appropriate for performing non-linear dynamical system identification.

1.3 Thesis Contributions

One of the major contributions of this thesis is the development of estimators capable of estimating the position of the slung load relative to the vehicle. The main techniques investigated are Computer Vision and Machine Learning. The first method uses a stream of images from a downwards looking gimballed-camera to calculate a position vector of the load relative to the MRUAV. The second estimator uses real flight data to train a machine learning architecture that can predict the position vector of the load in the MRUAV fixed frame using the vehicle pose and pilot pseudo-controls as input. Experimental results show very accurate position estimation of the load using the machine learning estimator when comparing it with a motion tracking system (~2% offset).

The next major contribution relates to enableling flight capabilities of a MRUAV with slung load for general cargo transport, a control system is developed that dampens the oscillations of the load when the MRUAV is following a desired flight trajectory. Such control scheme uses a feedback approach to simultaneously prevent exciting swing and to actively dampen swing in the slung load. The methods and algorithms developed are validated by flight testing.

Another contribution is a methodology for the characterization and maximization of the thrust as well as the prediction of the time of flight of a hovering multirotor while maintaining constraints of specific mission requirements. The results shows that this methodology is more accurate than on-line calculators (~2% vs ~5% offset) when comparing with real flights. Using 3D printed components such as the Rotite (Burns, 2014) elements resulted in a time of flight increasing by approximately 22%. More importantly, as an added benefit, the arm of the quadrotor was able to *rotate* without coming loose from the frame, therefore improving crash-survivability of the quadrotor frame.

Regarding control of multirotors, the avionics suite (Flight Stack and DronePilot) is another contribution resulting from the necessity of solving the research hypothesis. Although there are several similar flight stacks on the market, they can be 13 times as expensive as the one produced in this thesis, this allows the construction of several models without defeating the fundamental constraint of cost. The software contribution (DronePilot) ties the avionics suite and allowed the vehicle to perform accurately and precisely. The hardware and software proposed are validated by numerous flight testing.

The black-box system identification quadrotor models generated in this thesis have good generalization capabilities with good accuracy. More importantly, it was demonstrated that the learned dynamics can be used effectively on-board of the system, inside the flight stack. This is a valuable contribution that has an application in GPS-denied environments.

1.4 Thesis Structure

Chapter 2 This chapter addresses the description and optimal selection of propulsion components for a multirotor, for a given payload capacity, number of rotors and flight duration. Using a simplified approach mathematical models are developed for motors, propellers, electronic speed controllers (ESC) and batteries, therefore allowing the characterization and maximization of the thrust. Time of flight of a hovering multirotor prediction models are developed using experimental data.

Chapter 3 In this chapter the set-up of the Micro Air Systems Technology Laboratory (MAST Lab) is presented, followed by the description of the avionics suite which compromises the *Flight-Stack* and the *DronePilot* software framework. Sections of this chapter are published in Vargas et al., 2016.

Chapter 4 The derivation of the quadrotor vehicle mathematical model is provided in this chapter. This result is very important because it describes how the multirotor moves according to its inputs. The model equations from this section will be inverted so that in the control section 4.3 are used to identify which inputs are needed to reach a certain position. Experimental results from this section are used in Vargas et al., 2016.

Chapter 5 In this chapter, a discussion about machine learning is presented. The algorithms and methodologies used in this research effort are introduced. The goal of this section is to show the key algorithms and theory that is used in this research effort.

Chapter 6 With the machine learning techniques introduced alongside the multirotor mechanics and dynamics, this chapter is focused on *black-box* system identification to find system models capable of converging with the non-linear dynamics of MRUAV. Experimental flight data is presented. This chapter is based on Vargas et al., 2015b, coming from techniques used in Vargas et al., 2014.

Chapter 7 The slung load dynamics when coupled with the MRUAV (quadcopter) dynamics are analysed and presented in this chapter. Using computer vision and machine learning techniques, prediction of the position of the slung load is presented. A controller to dampen the oscillation of the load is presented and tested using the frameworks proposed in prior

chapters. Some of the work presented in this chapter was presented in Vargas et al., 2014 and Vargas et al., 2015a.

Chapter 8 This section summarizes the contributions of this thesis, discusses the results and proposes solutions to improve this work.

1.5 Publications

The works created and described in this thesis contributed to the following publications:

- Aldo Vargas, Murray Ireland, and David Anderson. *Swing free manoeuvre controller for RUAS slung-load system using ESN*. In: Proceedings of the 1st World Congress on Unmanned Systems Engineering. 2014

- Aldo Vargas, Murray Ireland, and David Anderson. *System Identification of multi-rotor UAVs using echo state networks*. In: AUVSIs Unmanned Systems. 2015.

- Murray Ireland, Aldo Vargas, and David Anderson. *A Comparison of Closed-Loop Performance of Multirotor Configurations Using Non-Linear Dynamic Inversion Control*. In: Aerospace 2.2 (2015), pp. 325352. URL: http://www.mdpi.com/2226-4310/2/2/325/. 2016

- Aldo Vargas, Murray Ireland, and David Anderson. *Swing-Free Manoeuvre Controller for Rotorcraft Unmanned Aerial Vehicle Slung-Load System Using Echo State Networks*. In: International Journal of Unmanned Systems Engineering 3.1 (2015), pp. 2637. URL: http://www.ijuseng.com/#/ ijuseng-3-1-26-37- 2015/4587568279. 2016.

- Aldo Vargas, Murray Ireland, Kyle Brown and David Anderson. *The MAST Lab flight stack for GNC of micro UAVs*. MDPI Robotics (ISSN 2218-6581). Pending publication.

- Aldo Vargas and David Anderson. *Computer vision technique to estimate the slung load dynamics when coupled to a Multirotor Unmanned Aerial Vehicle*. Revista Internacional de Investigación e Innovación Tecnológica (ISSN 2007-9753). Latindex Folio: 23614. 2017.

2 Multirotor Design

In this chapter, the description of the various components for multirotor vehicles is presented. A methodology for the characterization and maximization of the thrust is presented as well as the prediction of the time of flight of a hovering multirotor while maintaining constraints of specific mission requirements. Experimental tests of the rotor components are presented and compared with on-line calculators. In order to answer the research question (Sec. 1.1.3), a multirotor vehicle with the capability of carrying a slung load with sufficient extra thrust to be able to manoeuvre while maintaining the maximum time of flight possible and low costs is needed. Usually, multirotors do not require a human pilot to be on-board which allows it to be used in dangerous situations or in hazardous environments, such as disaster areas. Not having a pilot means that the MRUAV must contain advanced on-board autonomous capabilities and operate with varying degrees of autonomy. The mechanical simplicity of MRUAV makes them ideal test-beds for GNC (Guidance Navigation and Control) research due to their ease of construction, agility, expandability and reusability. A multirotor is a rotorcraft with more than two rotors, their advantage is the simpler rotor mechanics required for flight control. Rather than employing mechanically-complex main and tail rotors, a multirotor employs several identical rotors to provide both lift and control. Designing a miniature autonomous multirotor is basically dealing with numerous design parameters that are closely linked. Taking a decision about all these parameters requires a clear methodology. Selecting the correct hardware for a multirotor can be challenging in order to get the best flight performance of the system. One of the first parameters to choose is the number of actuators the vehicle will have. In this document we will refer to ***rotor*** (actuator) as the combination of propeller, electric motor and speed controller. The most common number of rotors used in research labs around the world is four, or most commonly named as quadrotor or quadcopter. Quadrotors have four fixed-pitch propellers in a ***plus***

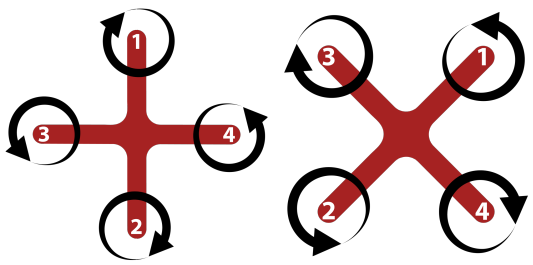


Fig. 2.1.: Quadrotor possible flying configurations

or ***cross*** configuration as showed in figure 2.1, both are slightly different, one of the differences is the placing of the flight controller and the control mixing (e.g. To fly forward on the plus configuration rotor 4 must produce more thrust than rotor 3 while in cross configuration rotors 2 and 4 most produce more thrust than rotors 3 and 1), the other differences are discussed in the modelling Section 4.1. Rotating the two pairs of propellers in opposite directions removes the need for a tail rotor. Increasing or decreasing the speed of the four propellers simultaneously permits climbing and descending. Vertical rotation (*yawing*) is achieved by creating an angular speed difference between the two pairs of rotors.

2.1 Frame

In this section the airframe for multirotors is discussed, which is a mechanical structure that holds the parts that compose a multirotor. As stated before the most common multirotor platform is the quadrotor, but additional multirotor platforms are easily conceived by augmenting the quadrotor with additional rotors. This provides greater lifting force without the need to upgrade components, such as the rotor blades and motors. However, the introduction of additional rotors changes the dynamic performance of the system Ireland et al., 2015, adding extra mass and typically requiring additional supporting structure. The sec-

Credit: *MAST Lab*

Fig. 2.2.: TEGOv2 - 3D printed quadrotor performing a hover flight.

ond iteration for the design of the test-bed multirotor (Fig. 2.2) for the MAST Lab (Sec. 3.2) was considered using a 4 rotor configuration, because its easier and more practical when

repairing and maintaining. Many research groups have begun constructing multirotors with four rotors as robotics research tools/platforms.

2.1.1 Structures

The frame holds together the entire aircraft and it must survive constant crashes and/or hard-landings (which could potentially damage the entire vehicle) and it serves as vibration dampening/isolation from the rotors to the flight controller. The mechanical structure

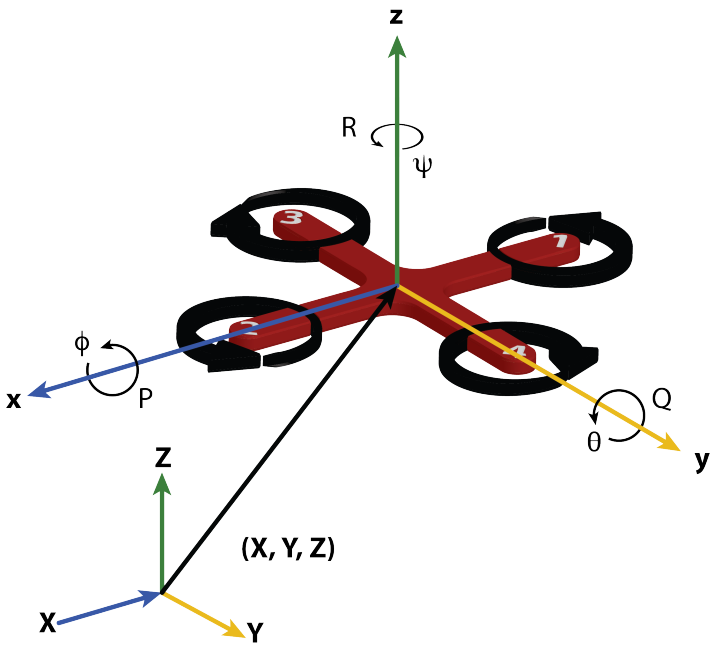


Fig. 2.3.: Quadrotor free body diagram

design of multirotors is relatively simple. Typical multirotors utilize the spar method, with each spar anchored to a central hub/plate like spokes in a wheel. A example of a multirotor arm is showed in figure 2.4. This arm is part of glass fibre frame that it also has being used in the MAST Lab. The arm is made from a durable nylon polyamide material, while the centre plates are made from glass fibre. Glass fibre is a general purpose plastic reinforcement material and relatively inexpensive. It is durable, has good heat resistance and good tensile strength. Nylon polyamide is a type of plastic, which is a human-made synthetic polymer and its used on the multirotor industry due to its semi-crystalline property which is generally a very tough material with good thermal and chemical resistance. Some assumptions are to be made in order to simplify the design process. The multirotor inertia is increased with an increase of frame size, therefore the frame diameter is kept as small as possible only to keep the propeller tip to tip distance equal to a small positive number.

Fig. 2.4.: Nylon polyamide multirotor arm and assembled glass fiber frame

2.1.2 Configurations

As stated at the introduction of this chapter, multirotors can utilize two or more rotors, it depends on the maximum payload the vehicle is going to carry or by other specification also heavily considered in this type of aircraft, which is transportability. This specification refers to the capability of easiness of transportation when the vehicle its not flying. For a quadrotor set-up, one set of hardware can give the optimal design, while for a six rotor set-up the same hardware may not necessarily give the same response as showed on table 2.1. For better stability, fault tolerance or stronger lift force the number of motors must be increased. The number of rotors (motor and propeller tuple) is constrained to an equal

# Rotors	Motor	Battery	Propeller	Weight	Hover Time
4	2200kv	2.2ah, 20C	5x3 in	303 grams	17 minutes
6	2200kv	2.2ah, 20C	5x3 in	572 grams	11 minutes

Tab. 2.1.: Loiter flight times of similar configuration multirotors

number of counter-rotating rotors to eliminate the torque in the frame yaw (ψ) axis (figure 2.3). If we consider the configuration of a multirotor using just three rotors, then the vehicle can be called trirotor (Fig. 2.5), and it employs three rotors and a servo (small servomotor/actuator used in small-scale robotics) to rotate one of them to compensate for adverse torque in yaw axis.

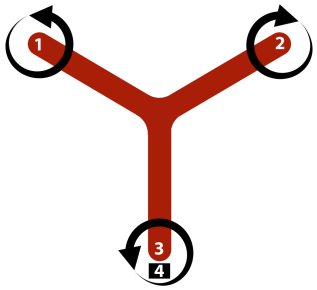


Fig. 2.5.: Trirotor - three rotor configuration vehicle.

2.1.3 Redundancy

For a multirotor to fly and not spin around an even number of rotors is needed and these need to spin in opposite directions to counteract the torque spin. On a trirotor, this torque twisting effect is countered by the rear rotor being angled to vector the thrust in the opposite direction to the twisting effect. On a quadrotor there is no need to compensate because each rotor has an equal opposite to counteract the spin. If one rotor fails then you are effectively left with a trirotor configuration but without the thrust vectoring capability of an angled third rotor. So, the vehicle then it will spin out and crash. In a hexarotor configuration (Fig. 2.6 left) there are 3 opposing pairs of rotors. If you lose one rotor you'll be left with 5 spinning rotor, however, the flight controller will detect a drop of stability and a resulting yaw effect and will reduce the thrust generated on one of the rotors that would be spinning in the opposite direction, effectively leaving you with a quadrotor configuration, the only reason this might fail is if the weight of the vehicle and payload cannot be lifted with only four rotors, otherwise it will be able to continue performing the mission. If we increase the

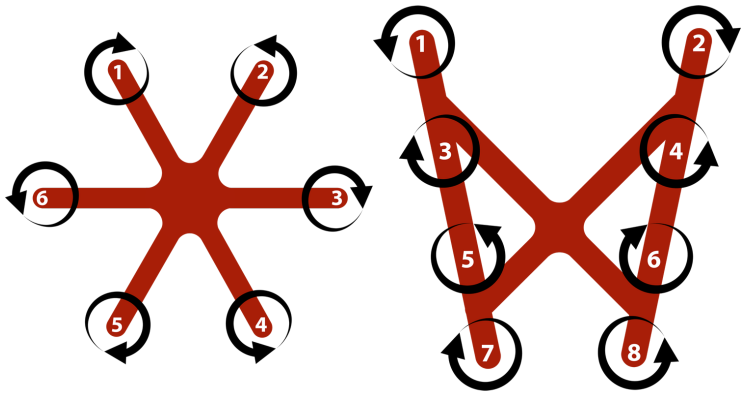


Fig. 2.6.: Two redundant multirotor configurations - Left: Hexarotor. Right: Octorotor V-shape

number of rotors, the redundancy will be increased but the weight and complexity factor of the multirotor will became a decision factor. Figure 2.6 right shows a multirotor, using eight rotors, in a V-shape align configuration, for full redundancy.

2.1.4 Materials

There is two key parameters when designing/choosing a frame for a multirotor, one is weight and the other one is sturdiness. The most common materials used in the construction of multirotor frames are carbon fibre, glass fibre, plastics (nylon, ABS, PLA), aluminium, wood. Recently there is more accessible 3D printers capable of printing parts or even the entire frame, in this research effort, there is a special interest in the usage of this type of

technology to aid in the building stage of multirotors. If carbon fibre frame is used, then the frame weight is calculated as the length of the frame multiplied by $0.045kg/m$ which is the weight of a $7x9x1000mm$ carbon fibre tube.

3D printing

3D printing is the process of being able to print any object layer by layer (Anastasiou et al., 2013) or making a three-dimensional solid object from a digital model. Its also called additive manufacturing. The current generation of 3D printers typically requires input from a CAD program in the form of an STL file, which defines a shape by a list of triangle vertices. The printer used in this multirotor research effort is a *Makerbot Replicator 2*, this machine uses a extrusion deposition method, also called Fused Deposition Modelling (FDM). The part is produced by extruding small beads of material which harden immediately to form layers. A thermoplastic filament is supplied to a extrusion nozzle head, the nozzle heats the material and turns the flow on and off, stepper motors are used to move the extrusion head in both horizontal and vertical directions. Control of the machine is typically done by CAM software (Computer Aided Manufacturing). Entire frames and parts for small scale multirotor vehicles can be printed using this practical device. More information about this printer can be found in Appendix A.1. The material used on the frame printings is PLA (polylactic acid) which is a thermoplastic aliphatic polyester derived from renewable resources. In this research, several parts and frames have being design and built having in mind 3D printing technologies. Fig. 2.2 shows the latest version of the first 3D printed quadrotor designed in the MAST Lab, named *TEGO*. Which evolved from a standard quadrotor frame showed in figure 2.7. As stated before, there is two main design parameters for a frame in a multirotor,

Fig. 2.7.: TEGO v1 quadrotor

sturdiness and lightweight. The arms in the model TEGOv1 are constructed using a truss structure (Fig. 2.8), which compromises triangular units constructed with straight members whose ends are connected at joints referred to nodes. External forces and reactions to those forces are considered to act only at the nodes and result in forces in the members

which are either tensile or compressive forces. This type of structure makes it ideal for multirotor arms for robustness and crash resistance. The main issue with the first version of

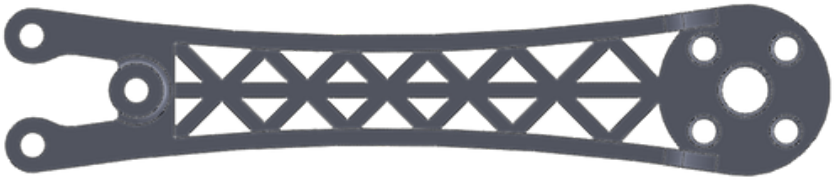


Fig. 2.8.: TEGOv1 truss structure arm

the TEGO model frame was the weight of it, in order to interconnect the four arms with the top and bottom centre plate nuts and bolts had to be used. For a airframe of this size/scale adding this type of extra weight has a consequence in reduced time flights. The next design iteration was using a special mechanical fastener called Rotite (Burns, 2014), this avoid the usage of extra joining parts and therefore the frame weight was reduced.

Rotite

Rotite is basically a mechanical fastener. More deeply is a device that includes first and second inter-engageable parts, each having a longitudinal axis and a connecting face extending substantially transversely to the longitudinal axis. The first part has an engagement formation extending substantially axially and the second part has a receiving formation extending substantially axially, and in which the engagement formation is receivable. The engagement and receiving formations each includes a substantially helicoidal surface extending at least partially around the longitudinal axis of the respective part of the connector, so that rotation of the parts relative to one another about the longitudinal axes, when the parts are substantially co-axially aligned, so that the connecting faces of the parts face one another in a substantially axial direction, causes engagement of the engagement formation with the corresponding receiving formation. More information about this mechanical fastener can be found in Appendix A.2. The arm and centre plate of TEGOv1 was adopted in order to fit this mechanical fastener and the result was a airframe $\frac{2}{3}$ lighter than TEGOv1, mainly because no bolts/nuts where used to put the frame together, making a lighter air-frame translated in a substantial increase of flight times (Vargas, 2013a). The truss-rotite arm is showed in figure 2.9 and it became the first application of Rotiteś in the aerospace industry. This frame was named TEGOv2 and the maiden video can be seeing at [1]. Using this mechanical fastener did not only reduced weight of the airframe, it also help in the crash-survivability characteristic of the vehicle. When testing the vehicle in the MAST Lab, and

[1] `https://www.youtube.com/watch?v=G9jUP6Z5ENA`

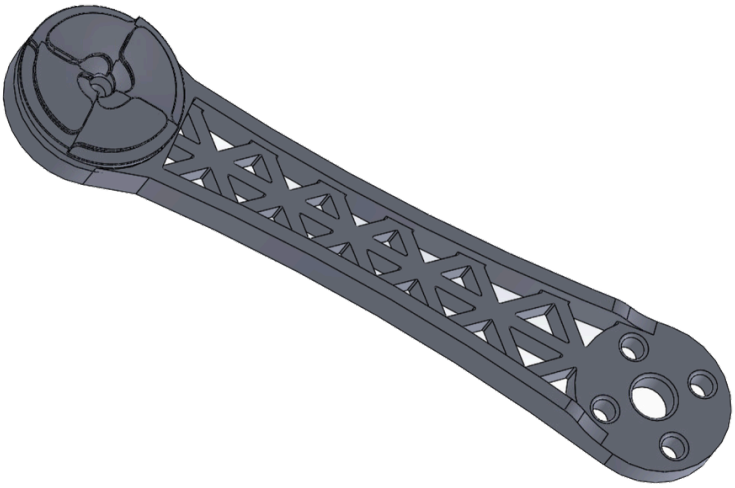


Fig. 2.9.: TEGOv2 arm with Rotite element

Fig. 2.10.: TEGOv2 with Rotite a)normal b)crash-survivability characteristic

having several crashes with TEGOv1 and TEGOv2 (Fig. 2.10 a) in some cases the arm just bended (almost in the Rotite opened position) instead of breaking, without disassembling as showed in Fig. 2.10 b. This is a desired behaviour on multirotor frames and research platforms, where crashing when testing and tuning advanced novel algorithms is something regular.

2.2 Motor

High torque and high speed motors that develop rotational speeds in excess of $5000rpm$ are necessary for multirotors in order to achieve stability and movement in all degrees of freedom. The current trend in multirotor motor technology is to use permanent magnet (PM) brushless motors (BLDC). The highest efficiency and highest power density is achieved with permanent magnets brushless motors Gieras, 2014.

Fig. 2.11.: BLDC Motor

2.2.1 Brushless motors

Brushless motors fall into the two principal classes, sinusoidally excited and square wave (trapezoidally excited) motors. Sinusoidally excited motors are fed with three-phase sinusoidal waveforms and operate on the principle of a rotating magnetic field (Gieras, 2002). They are simply called sine-wave motors or PM (Permanent Magnet) synchronous motors. Square wave motors are also fed with three-phase waveforms shifted by 120° one from another, but these wave-shapes are rectangular or trapezoidal (Fig. 2.12). It is important to notice that all phase windings conduct current at a time. The later is most commonly use in multirotor design. The most important advantages for using BLDC motors on mul-

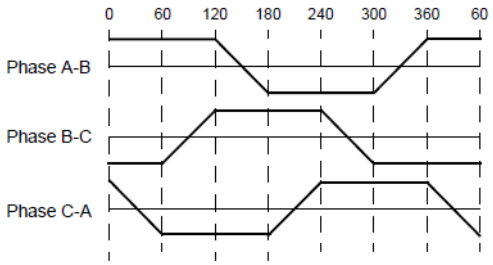


Fig. 2.12.: Three-phase trapezoidally excited waveform for PM BLDC motor

tirotors are high power to weight ratio, high efficiency, high torque, good dynamic control for variable speed applications, absence of brushes and commutator. This absence means there is no problem of mechanical wear of the moving parts (Jang et al., 2004), also better heat dissipation property and ability to operate at high speeds (Ede et al., 2001) make them superior to the conventional direct current motors.

2.2.2 Mathematical model

BLDC motor are considered as a three phase synchronous machine as stated before. Since its rotor is mounted with permanent magnets, some dynamic characteristics are different. Flux linkage from the rotor is dependent upon the magnet. Therefore, saturation of magnetic flux linkage is typical for this kind of motors. One structure of the BLDC motor is

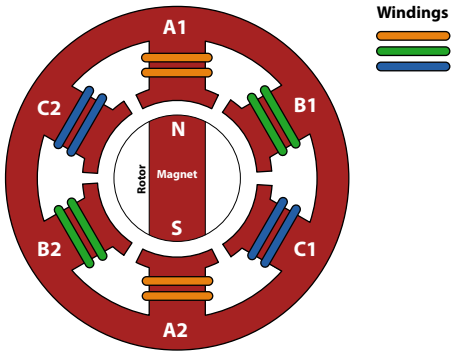


Fig. 2.13.: BLDC motor basic diagram

fed by a three phase voltage (which should not exceed the maximum voltage limit of the motor), the model of the armature winding for the BLDC motor is expressed as follows

$$\begin{aligned} \mathbf{v_a} &= \mathbf{R i_a} + \mathbf{L_a}\frac{d\mathbf{i_a}}{dt} + \mathbf{e_a} \\ \mathbf{v_b} &= \mathbf{R i_b} + \mathbf{L_b}\frac{d\mathbf{i_b}}{dt} + \mathbf{e_b} \\ \mathbf{v_c} &= \mathbf{R i_c} + \mathbf{L_c}\frac{d\mathbf{i_c}}{dt} + \mathbf{e_c} \end{aligned} \tag{2.1}$$

where the armature inductance $\mathbf{L_a} = \mathbf{L_b} = \mathbf{L_c} = \mathbf{L}$, the armature resistance (in Ohms) $\mathbf{R_a} = \mathbf{R_b} = \mathbf{R_c} = \mathbf{R}$, terminal phase voltage (in Volts) $\mathbf{v_n}$, motor input current (in amperes) $\mathbf{i_n}$ and motor back EMF (in volts) $\mathbf{e_n}$. The matrix form is

$$\begin{bmatrix} \mathbf{v_a} \\ \mathbf{v_b} \\ \mathbf{v_c} \end{bmatrix} = \begin{bmatrix} \mathbf{R} + \frac{d\mathbf{L}}{dt} & 0 & 0 \\ 0 & \mathbf{R} + \frac{d\mathbf{L}}{dt} & 0 \\ 0 & 0 & \mathbf{R} + \frac{d\mathbf{L}}{dt} \end{bmatrix} \begin{bmatrix} \mathbf{i_a} \\ \mathbf{i_b} \\ \mathbf{i_c} \end{bmatrix} + \begin{bmatrix} \mathbf{e_a} \\ \mathbf{e_b} \\ \mathbf{e_c} \end{bmatrix} \tag{2.2}$$

2.2.3 Motor parameters

The back EMF constant $\mathbf{K_v}$, also called motor velocity constant, measured in *rpm per volt*, which is the ratio of the motor's unloaded (no load attached to its axle) RPM to the peak voltage on the phases connected to the armature. On literature the terms $\mathbf{K_e}$ and $\mathbf{K_b}$ are

also used as terms of back EMF. While $\mathbf{K_v}$ is used as the current trend in manufacturers of multirotor parts specifications. Back EMF is expressed as

$$\mathbf{E_b}(t) = \mathbf{K_v}\phi\Omega \tag{2.3}$$

where $\mathbf{E_b}$ is back EMF, $\mathbf{K_v}$ the constant, ϕ is the field flux and Ω is the angular speed. The back EMF is calculated on each phase (shifted 120°) as

$$\begin{aligned}
\mathbf{e_a}(t) &= \mathbf{K_v}\phi(\theta)\Omega(t) \\
\mathbf{e_b}(t) &= \mathbf{K_v}\phi(\theta - \frac{2\pi}{3})\Omega(t) \\
\mathbf{e_c}(t) &= \mathbf{K_v}\phi(\theta + \frac{2\pi}{3})\Omega(t)
\end{aligned} \tag{2.4}$$

$\mathbf{K_v}$ can be found experimentally by testing the motor and knowing the terminal resistance $\mathbf{R}$, and is not strictly limited to factors like armatures, poles, or geometric characteristics of the motor. It is possible that a wider flat construction can lead to a better $\mathbf{K_v}$ given the same winds and same quality materials. The torque $\mathbf{Q_E}$ is calculated as follows:

$$\mathbf{Q_E} = \frac{\mathbf{e_a i_a} + \mathbf{e_b i_b} + \mathbf{e_c i_c}}{\Omega} \tag{2.5}$$

where the torque constant $\mathbf{K_t}$ is the torque produced per ampere.The calculations of torque per phase are

$$\begin{aligned}
\mathbf{Q_a}(t) &= \mathbf{K_t}\phi(\theta)\mathbf{i_a}(t) \\
\mathbf{Q_b}(t) &= \mathbf{K_t}\phi(\theta - \frac{2\pi}{3})\mathbf{i_b}(t) \\
\mathbf{Q_c}(t) &= \mathbf{K_t}\phi(\theta + \frac{2\pi}{3})\mathbf{i_c}(t)
\end{aligned} \tag{2.6}$$

After using Newton's second law of motion, the torque balance equation is described as:

$$\mathbf{Q_E}(t) - \mathbf{Q_L}(t) = \mathbf{J_R}\frac{d\Omega(t)}{dt} + \mathbf{B}\Omega(t) \tag{2.7}$$

where $\mathbf{Q_L}(i)$ is the load torque (in $\mathbf{Nm}$), $\mathbf{J_R}$ being the rotor inertia (in kgm^2) and $\mathbf{B}$ the damping constant. The motor torque is related to the applied current that sets the strength of the magnetic fields generated by the motor windings, a simplified version of 2.7 is

$$\mathbf{Q}_m = \mathbf{K_t}(\mathbf{i} - \mathbf{i_0}) \tag{2.8}$$

The internal back EMF $\mathbf{v_m}$ is proportional to the rotation rate Ω via the constant $\mathbf{K_v}$, using usual electrical circuit equations and conservation of energy, gives the following parameters in function of the motor current $\mathbf{i}$ and the motor terminal voltage $\mathbf{v}$:

$$\mathbf{Q}_m(\mathbf{i}) = \frac{\mathbf{i} - \mathbf{i_0}}{\mathbf{K_v}} \tag{2.9}$$

$$\Omega(\mathbf{i}, \mathbf{v}) = (\mathbf{v} - \mathbf{iR})\mathbf{K_v} \tag{2.10}$$

$$\mathbf{P}_{shaft}(\mathbf{i}, \mathbf{v}) = \mathbf{Q_m}\Omega = (\mathbf{i} - \mathbf{i_0})(\mathbf{v} - \mathbf{iR}) \tag{2.11}$$

$$\mathbf{P}_{elec}(\mathbf{i}, \mathbf{v}) = \mathbf{vi} \tag{2.12}$$

$$\eta_m(\mathbf{i}, \mathbf{v}) = \frac{\mathbf{P}_{shaft}}{\mathbf{P}_{elec}} = (1 - \frac{\mathbf{i_0}}{\mathbf{i}})(1 - \frac{\mathbf{iR}}{\mathbf{v}}) \tag{2.13}$$

These equations depend on the tuple of motor parameters [$\mathbf{R}$, $\mathbf{i_0}$, $\mathbf{K_v}$], which are usually provided by the manufacturers or are very easily obtainable. Using equations 2.8 and 2.9 we can find the relation between them (2.14). $\mathbf{K_t}$ is inversely related to $\mathbf{K_v}$. This means that if $\mathbf{K_v}$ is bigger, $\mathbf{K_t}$ gets smaller. Stronger magnets with higher flux density decrease the voltage needed to produce a torque

$$\mathbf{K_t} = \frac{1}{\mathbf{K_v}} \tag{2.14}$$

There is two common types of BLDC motors, in-runner and out-runner. The out-runner motor usually spins slower than their in-runner counterparts, which have a more traditional layout, in the world of direct current motors. The out-runner can generate more torque than a similar size in-runner motor because geometrically it can accommodate a higher number of permanent magnets than an in-runner of similar size as showed on figure 2.14. The

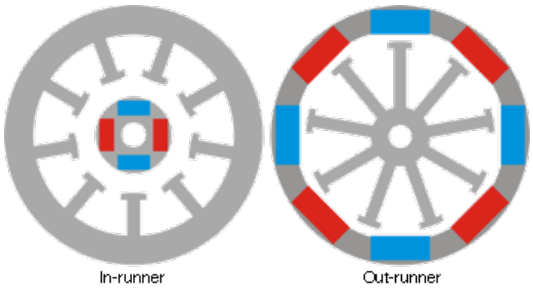


Fig. 2.14.: BLDC types. In-runner and Out-runner

number of poles is important when choosing a motor, the most common in-runner BLDC motor have 2 or 4 magnetic poles. Currently the trend is to have more than 10 poles in a out-runner BLDC configuration. Figure 2.16 shows a multirotor specific motor with 12 neodymium magnets and 9 poles, this motor is used on small scale multirotors (frame size less than $250mm$). When having more poles, the geometric characteristics of the motor change in order to accommodate them, ending with a much *wider* motor, this is commonly

called *pancake* motor 2.15. *Pancake* motors will produce more torque and consequently will have a lower $\mathbf{K_v}$. Having a lower $\mathbf{K_v}$ translates in more torque and therefore we can create a rotor with a much bigger propeller and produce the same thrust but at lower RPM, which is useful to reduce vibrations and having a much more stable flight. One of the reasons

Fig. 2.15.: Example of a *pancake* motor.

why more poles in the BLDC motor means more torque is because its able to handle greater current loads. This is because the manufacturer has to reduce the winding when increasing the number of poles and that means adding more copper to the winding, when the cross section of a wire is increased the resistance decreases. This decreased resistance would allow a greater current load to pass through the motor. The greater current loads will result in greater power. The torque is gained as a result of lower $\mathbf{K_v}$. Depending on the exact

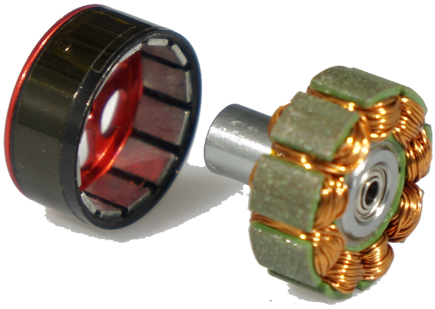


Fig. 2.16.: Brushless motor with neodymium magnets and poles exposed.

application of the multirotor this may be an advantage or disadvantage. More torque may mean greater acceleration but a lower $\mathbf{K_v}$ value will reduce the maximum rotational speed achievable.

2.3 Propeller

The propeller is basically a device which transforms rotational motion into linear thrust, it can be viewed as a series of airfoils that each generate lift and drag when rotating at high speeds, and together they constitute the total aerodynamic thrust and torque on the

propeller. An example of a nylon (plastic) fixed pitch propeller is showed in figure 2.17 a carbon fibre close-up is showed in figure 2.18. A pressure difference is produced between

Fig. 2.17.: Common nylon propeller used in multirotors.

the forward and rear surfaces of the airfoil-shaped blade, and air is accelerated behind the blade. The most common propeller used in the multirotor industry are fixed pitch propeller (FPP), in this type of device, the angle of attack of the blade remains fixed and in order to increase/decrease the thrust generated the rotational speed must be changed. This type of propellers come in two form factors, clockwise and counter-clockwise (also called right and left). The angle of attack or *pitch* describes how much the blade of the propeller is twisted relative to the path it travels as it turns, is usually measured in inches. Variable-pitch propellers can also be used on a multirotor (Wong et al., 2007), this type of propeller has blades that can be rotated around their long axis to change the blade pitch, can also create reverse thrust for braking or going backwards without the need to change the direction of shaft revolution. The material from which the propeller is made might affect the efficiency of the propeller at different RPMs. This may occur for softer propellers due to flexure of the blades changing effective angle of attack at radial sections away from designed angles (Harrington, 2011). Increasing the propeller pitch and number of blades generally generates more thrust, but at a cost of efficiency and increased electrical and mechanical power requirements on the motor. Increasing the propeller radius is generally more efficient, assuming the rest of the drive system is capable of handling the load. This is because the larger propeller, with all else being equal, may spin slower to generate the same lift. This allows the induced velocity to drop, thereby increasing propulsive efficiency. A important

Fig. 2.18.: Carbon fibre fixed pitch propeller.

assumption is made that the calculated dynamic performance is a measurement of the rotational attitude performance of the multirotor, not the translational velocity or position. It is based on static propeller tests and system inertia, hence variable airflow through the propeller during flight is not taken into consideration. The payload and batteries will not affect the dynamic performance since they are modelled as a point mass in the centre of the rotation. Figure 2.19 shows a 3-bladed propeller. The majority of propellers used in

Fig. 2.19.: 3 bladed fixed pitch propeller.

the radio control industry have two blades but propellers with three or even four blades are available. Adding more blades decreases the overall efficiency of the propeller because each blade has to cut through more turbulent air from the preceding blade. Experimental results about a vehicle using standard two bladed rotors and three bladed ones is presented on the systems analysis section in table 2.4. There is a new trend on propellers for multirotor use and is to use foldable propellers (Fig. 2.20), this type of propellers help a lot in the transportability parameter. Two of the foldable propellers are required to form a single rotor, and when this ones are folded, the size of the vehicle decreases and it became easier to transport. This type of elements are usually for bigger size of propellers, starting with a length of $15in$.

Fig. 2.20.: Carbon fibre foldable propeller.

2.3.1 Propeller parameters

A propeller is characterized by the thrust and power coefficients, which depend primarily on the advance ratio λ, the blade Reynolds number $\mathbf{R}_e$, and on the prop geometry.

$$\mathbf{C}_T = \mathbf{C}_T(\lambda, \mathbf{R}_e, geometry) \tag{2.15}$$

$$\mathbf{C}_Q = \mathbf{C}_Q(\lambda, \mathbf{R}_e, geometry) \tag{2.16}$$

$$\mathbf{J} = \frac{\mathbf{V}}{\Omega \mathbf{D}} \tag{2.17}$$

$$\mathbf{R}_e = \frac{\rho \Omega \mathbf{R} c_{ave}}{\mu} \tag{2.18}$$

The dimensional thrust and torque can be calculated for any other $\mathbf{V}$ and Ω by dimensionalizing the coefficients.

$$\mathbf{T}(\Omega, \mathbf{V}) = \frac{1}{2}\rho(\Omega\mathbf{R})^2\pi\mathbf{R}^2\mathbf{C}_T = \frac{1}{2}\rho\mathbf{V}^2\pi\mathbf{R}^2\frac{\mathbf{C}_T(\Omega, \mathbf{R}_e)}{\mathbf{J}^2} \tag{2.19}$$

$$\mathbf{Q}(\Omega, \mathbf{V}) = \frac{1}{2}\rho(\Omega\mathbf{R})^2\pi\mathbf{R}^3\mathbf{C}_Q = \frac{1}{2}\rho\mathbf{V}^2\pi\mathbf{R}^3\frac{\mathbf{C}_Q(\Omega, \mathbf{R}_e)}{\mathbf{J}^2} \tag{2.20}$$

2.3.2 Static thrust

Calculations of the static thrust generated by a propeller are needed in order to ensure that the elements have been selected when designing a multirotor. Static thrust is defined as the amount of thrust produced by a propeller which is located stationary to the earth. This calculation is particularly important because multirotors are more likely to perform at low speeds relative to the earth. This low-speed performance ensures that the calculations of static thrust can be applied to a wide range of flight conditions. The propeller thrust is based on momentum theory, the theoretical thrust for a *stationary* aircraft is

$$\mathbf{T} = \dot{\mathbf{m}}\mathbf{V}_e \tag{2.21}$$

For a moving aircraft, however, only the velocity of the air which is due to the air having been accelerated by the propeller is what contributes to the thrust.

$$\mathbf{T} = \dot{\mathbf{m}}\Delta\mathbf{V} = \dot{\mathbf{m}}(\mathbf{V}_e - \mathbf{V}_{ac}) \tag{2.22}$$

Based in equation 2.22, as the aircraft velocity, $\mathbf{V}_{ac}$, increases, thrust decreases. This is due to the fact that the propeller exit velocity (or induced velocity) is approximately con-

stant, and therefore the result of $(\mathbf{V}_e - \mathbf{V}_{ac})$ approaches zero as the aircraft top speed is reached.

$$\dot{\mathbf{m}} = \rho \mathbf{A}_{prop} \mathbf{V}_e \tag{2.23}$$

The mass flow rate (2.23) is the density of the air times the cross-sectional area through which the air is flowing, times the velocity of the air. Therefore the dynamic thrust is defined as

$$\mathbf{T} = \frac{\pi}{4} \mathbf{D}_{prop}^2 \rho \mathbf{V}_e (\mathbf{V}_e - \mathbf{V}_{ac}) \tag{2.24}$$

$\mathbf{V}_e$ is assumed to be equal to the pitch speed of the propeller, defined as the distance the propeller moves forward through the fluid during one revolution. $\mathbf{V}_e$ only depends on the propeller rotational speed and the pitch. Another assumption must be made to simplify the calculations of the thrust generated by a propeller, and that refers to the air density, which is assumed to be at standard day (Atmosphere, 1964) which is as a way of defining certain properties of the atmosphere in a manner which allows those who use our atmosphere to effectively calculate and communicate its properties at any given time. The standard day assume $\rho = 1.225 kg/m^3$. Making $\mathbf{V}_{ac}$ zero we can then calculate the static thrust generated by a propeller, equation 2.25 shows the static thrust considering the common units marked by the manufacturers, those are thrust in *newtons*, propeller geometry in *inches*.

$$Prop_{Thrust} = 1.225 \frac{\pi (0.0254 Prop_{diameter})^2}{4} (Prop_{RPM} 0.0254 Prop_{Pitch} \frac{1}{60})^2 \tag{2.25}$$

This equation helps on the rotor design analysis in order to be able to choose a propeller, and get the initial estimate for thrust for hover in order to obtain an estimated time of flight.

2.4 Electronic Speed Controller

The ESC's is basically a computer system that will generate a three-phase square wave with the purpose of varying a brushless motor's speed. Each phase is shifted 120 degrees, as showed in figure 2.12. This special circuits have being analysed heavily in the literature as showed in Sai Dinesh et al., 2010 and Alexanderson et al., 1938. ESCs are very commonly used in the radio controlled hobby industry therefore some standards apply, this device is usually a stand alone unit which plugs directly to the flight controller to receive a control signal, which is usually a Pulse Width Modulated signal (PWM) and the output is 3 phases which are connected to the brushless motor. Figure 2.21 shows each independent phase

cable for the motor input on the left, in the centre the electronics components (ATMEGA or Arduino controller) and on the PWM signal input with the power source cables. Commercially, ESC are classified with its maximum current that can be pulled by the motor, the ESC showed in figure 2.21 is rated for maximum $20amperes$. In a more general sense, a

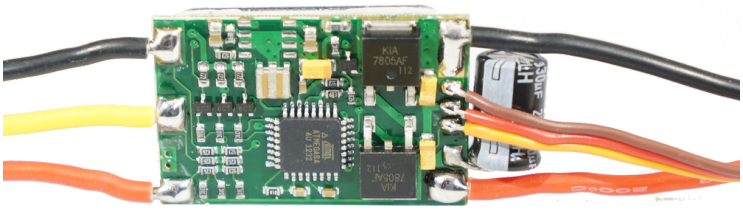


Fig. 2.21.: Generic ESC unit, outputs on the right and inputs on the left.

electronic speed control is a PWM controller for a electric motor. The ESC accepts a nominal $50Hz$ PWM servo signal as an input whose pulse width varies from $1ms$ to $2ms$. This type of control is very similar to how a *radio control servo* works, with the difference that in the servo the position is controlled while in the ESC speed is the factor being controlled. A ESC expects a pulse about every $20ms$, when $1ms$ at $50Hz$ is supplied to the ESC, the unit will respond to turn the motor off, while at $1.5ms$ PWM input signal the motor will be at approximately half of its total speed. When $2ms$ is applied, the motor will run at its maximum speed. A example of the a PWM signal can be seen at figure 2.22.

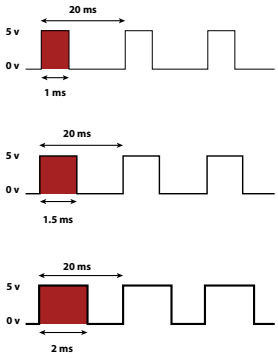
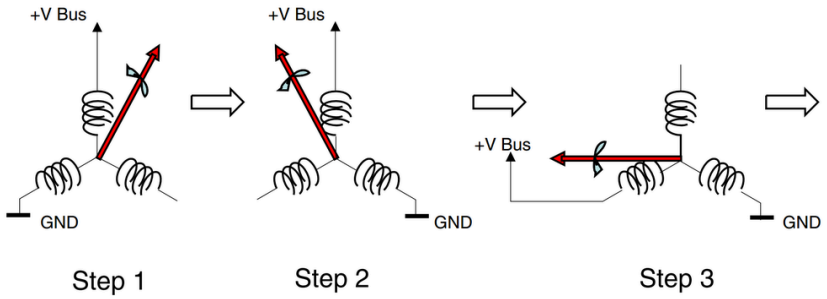


Fig. 2.22.: Pulse width modulation input signal range for a ESC.

2.4.1 Motor control

In BLDC motor control, the electrical cycle is subdivided into six commutation steps ST, 2009. For each step, the bus voltage is applied to one of the three phase windings of the motor while the ground is applied to a second winding. The third winding remains open. The successive steps are executed in the same way except that the motor phase wind-

ing changes to generate a rotating stator field [2], the sequence can be seen in figure 2.23. Modern ESC, or the ones specially designed for multirotor use contain a micro-controller interpreting the input signal and appropriately controlling the motor using a built-in software, also called firmware. In some speed controllers its actually possible to change or upgrade this software (firmware) for an alternate open-source one, or an improved one. This is done with purpose of increasing the performance of the rotor or to add extra features like active braking, reducing rotor synchronization problems and the possibility to synchronise the flight controller control loop with the motor output signal. This features improve throttle responsiveness and make the multirotor perform better and more robust. The default



Credit: *STMicroelectronics*

Fig. 2.23.: Motor control sequence.

PWM signal update rate is $50Hz$, this is to ensure maximum compatibility with normal RC equipment which normal analog servos sustain well (digital servos can accept higher rates), but there is a new trend of electronic speed controllers that can work using $400Hz$ as refresh rate. The recommended output rate for multirotors ESC is indeed $400Hz$, this is in order to minimize latency. Important to notice that is not because the output would require $400Hz$ (as the rotors on a multirotor spin only on the range of $80Hz$ to $120Hz$ and cannot change speed multiple times during a single revolution), but to actually overcome the input filtering most electronic speed controllers have and to minimize worst-case latency if the attitude control loop is not synchronized to the PWM generation.

2.5 Battery

The most common type of battery pack used in multirotors are lithium-ion polymer (LiPo) battery, because its the battery pack that delivers very high energy density (Salameh et al., 2009) and very high discharge rate conserving a low weight. This type of batteries (LiPo) are capable of specific energy of up to around $250Wh/kg$ Tarascon et al., 2001, about an order of magnitude lower than gunpowder, and two orders of magnitude lower than kerosene. Unlike cylindrical and prismatic cells, with metal casing, LiPo cells have a flexible

[2] `http://www.st.com/web/en/resource/technical/document/user_manual/CD00236524.pdf`

(polymer laminate) case, this translate in a cell 20% lighter than the equivalent cell of same capacity. This is an advantage to the multirotor application, where the overall weight is a very important factor to consider. However the lack of a hard case makes them more dangerous to handle, crush or penetration of the cell can result in a catastrophic failure. LiPo cells are affected also by overcharge, over-discharge and over-temperature issues. Each LiPo cell have a nominal voltage of $3.7\,\mathrm{V}$ and a full charge voltage of $4.2\,\mathrm{V}$. LiPo batteries

Fig. 2.24.: lithium-ion polymer battery with 3 cells and 2220 milli-Amp-hour capacity.

itself are stacked type and they come in different configurations. The configuration refers to the number of series and parallel cells that forms a pack. Figure 2.24 shows a $3s1p$ battery pack, with 3 cells in series (s), therefore this pack will provide a nominal voltage of $3.7 + 3.7 + 3.7 = 11.1\,\mathrm{V}$. The unit for battery capacity $\mathbf{C}_{batt}$ is milliampere hour (*mAh*), which is a rating of how many amperes a battery can output for one hour before its depleted. *C-rating* (discharge rate) is a multiplier which, when applied to the battery capacity, gives the theoretical maximum current the battery should be able to provide. The battery pack showed in figure 2.24 can output $2.2amp$ for one hour (C-rate) and it has a C-rating of $40 - 45C$ which means it can deliver $88\,\mathrm{A}$ to $110\,\mathrm{A}$.

2.6 Rotor

In this document the word rotor is used to describe the propeller, motor and ESC tuple. Its the main actuator for multirotors, is the part of the system that produces thrust that translates in the capacity to fly. With the information on the previous sections, we can proceed and analyse the dynamics behind the rotor. Propeller composition, radius, pitch, and number of blades must also be chosen to work properly with the chosen motor. The way the word is used on this document is not to be confused with the rotor of a helicopter, these terms are widely distant from each other. Multirotors have the advantage of extreme mechanical simplicity, it involves a number of direct drive motors, the same number of

propeller and that is it; while the rotor in a helicopter is much more complex because the angle of attack of the propeller needs to be changed. When selecting motors for a multirotor, companies that sell off-the-shelf components will give thrust tables that tell us how much thrust a specific motor can generate. This usually depends on a few factors, the propeller used and the voltage applied to them. Next to these two parameters you will usually see the current and power consumed and the resulting thrust generated, usually expressed in grams. For a multirotor be able to hover, we must ensure that the rotor can generate a thrust equal to or greater than the mass of the model. The electrical energy consumed by an aircraft is transferred to the kinetic energy of the moving air. Since multirotor are suspended in the air, there must clearly exist an opposing force $\mathbf{F}$ that is directed in a opposite direction $\mathbf{F} = \mathbf{mg}$, where $\mathbf{m}$ is the mass of the vehicle and $\mathbf{g}$ is the acceleration of gravity ($\mathbf{9.8}\frac{\mathbf{m}}{\mathbf{s^2}}$).

2.6.1 Mathematical analysis

For the purpose of matching the motor to a load, such as a propeller, we first manipulate relation 2.10 from the motor parameter analysis into a function for the current:

$$\mathbf{i}(\Omega, \mathbf{v}) = (\mathbf{v} - \frac{\Omega}{\mathbf{K_v}})\frac{1}{\mathbf{R}} \tag{2.26}$$

and then substitute into all the other right-hand sides to give the following functions of motor speed and voltage:

$$\mathbf{Q}_m(\Omega, \mathbf{v}) = [(\mathbf{v} - \frac{\Omega}{\mathbf{K_v}})\frac{1}{\mathbf{R}} - \mathbf{i_0}]\frac{1}{\mathbf{K_v}} \tag{2.27}$$

$$\mathbf{P}_{shaft}(\Omega, \mathbf{v}) = [(\mathbf{v} - \frac{\Omega}{\mathbf{K_v}})\frac{1}{\mathbf{R}} - \mathbf{i_0}]\frac{\Omega}{\mathbf{K_v}} \tag{2.28}$$

$$\eta_m(\Omega, \mathbf{v}) = [1 - \frac{\mathbf{i_0 R}}{\mathbf{v} - \Omega/\mathbf{K_v}}]\frac{\Omega}{\mathbf{v K_v}} \tag{2.29}$$

then, we need to use equation 2.24, that was obtained on the propeller part, which involves the diameter of the propeller, velocity of the air, among other parameters. An assumption has to be made, that the velocity of the air is half of the velocity of air accelerated by the propeller:

$$\mathbf{V}_e = \frac{1}{2}\Delta\mathbf{V} \tag{2.30}$$

the power absorbed by the rotor can be expressed like in equation 2.31, assuming rotational losses are negligible.

$$\mathbf{P}_{elec} = \frac{\mathbf{T}\Delta\mathbf{V}}{2} \tag{2.31}$$

2.6.2 Experimental analysis

In the case the manufacturer of the motor does not give the thrust that the component can produce with a certain propeller, or that the components selected does not match the one of the manufacturer, a experimental analysis must be performed in order to know what is the rotor performance and be able to estimate time of flight of the vehicle or to be sure that the correct components are being chosen. This experimental analysis involves performing real tests of the rotor at different conditions of a possible flight, in order to see the thrust, current and PWM that the rotor will produce, use and require. A special tool was

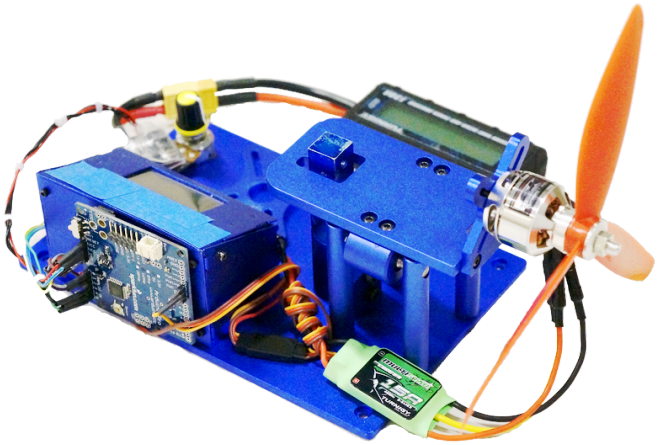


Fig. 2.25.: Rotor analysis tool.

designed and built for this academic effort, and its presented with more detail at Appendix A.3. This tool, showed in figure 2.25 has the capability to obtain the thrust produced by a propeller attached to a motor, electrical current and voltage consumed by the motor and the possibility to control the speed of the motor, this is done by generating a PWM signal that then is sent to the ESC that will eventually drive the motor (as showed on sections before). The idea behind this experiment is to simulate, on bench-test conditions, the same environment and parameters that a multirotor has when it is flying. The limitations of this tool relies on the quality of the sensors. The second limitation is in the design of the structure that holds the rotor, as it can block the free flow of air coming from the propeller. In order to demonstrate how to use this type of tool in the analysis of a rotor, data was obtained and analysed from this rotor analysis tool using two different propellers and one type of BLDC motor, which belong to one of the *test-bed* (Fig. 2.26) vehicles of the MAST Lab. The elements being analysed are showed in figure 2.27. The motor is a brushless DC motor, $1130kv$, model *Turnigy SK3-2826*, and two sets of propellers, with same diameter ($7inches$) and pitch ($3.8inches$), but with two and three bladed respectively. The tests where performed trying to follow standard day conditions, and the procedure was *scripted*. Having a micro-controller on-board the rotor analysis tool, it allowed us to

Fig. 2.26.: MAST Lab test-bed vehicles, using sets of different propellers.

Fig. 2.27.: *Top*: BLDC Motor 1130kv, *Left*: 7x3.8in 3-bladed propeller, *Right*: 7x3.8in 2-blade propeller.

script the tests to ensure that they where almost identical one from another, thus having very clean and precise data in order to perform comparisons using a three-bladed and a two-bladed propellers. Several sets of data were obtained and a polynomial of second order was used to provide the best fit (in a least-squares sense), to later be used on time of flight calculations. The fluctuations from the best fit line shown on each plot can be deemed negligible due to the conditions of the experiment. Firstly, the efficiency of the motor decreases due to heating, the battery behaves the same way. Mitigation of this loss of power and therefore torque was attempted by completing the experiment as fast as possible. Furthermore, the power source ($2.2Ah$, $40-50C$ discharge rate, LiPo battery,) causes the voltage supplied to the motor to drop when the temperature and load to it increases. Using the best fit polynomial equation of *thrust vs current*, we can obtain an equation of the current being drawn from the motor and ESC as a function of the thrust from the propeller,

where constants are obtained experimentally. This equation (2.32) is going to be later used on the *System Analysis* section.

$$\mathbf{i}_{rotor} = a_2\mathbf{T}^2 + a_1\mathbf{T} + a_0 \tag{2.32}$$

2.7 Flight Controller

An important component of the multirotor is the flight controller, which is a device that based on on-board sensors computes the necessary rotor speeds to keep the vehicle stable and flying. The flight controller can be defined as a system used to control the trajectory of a vehicle without constant control by a human operator being required. Flight controllers do not replace a human operator, but assist them in controlling the vehicle, allowing them to focus on broader aspects of operation. As stated at the beginning of this chapter, we are using commercial off-the-shelf (COTS) parts, and in the FC (flight controller) sense, this tendency remains. There are several publicly available open-source flight controller, one of the most successful in academia is the Pixhawk Meier et al., 2011 project. Great success are showed as well in the projects MultiWii MultiWii, 2010, Arducopter *Arducopter*, OpenPilot OpenPilot, 2011, Paparazzi Paparazzi, 2003 Bronz et al., 2009 and MikroKopter HiSystems, 2006 among others. Its important to notice that each FC has a different level of autonomy based on the sensors and devices being connected to it. In this research two FC will be used, Pixhawk (Fig. 2.28 - a) and Multiwii (Fig. 2.28 - b). They both make use of open source software. Chapter 3 describes the *Flight Stack*, which is a tuple of computer systems that work cooperatively in order to achieve a particular mission. The flight controller will be called *inner loop computer*.

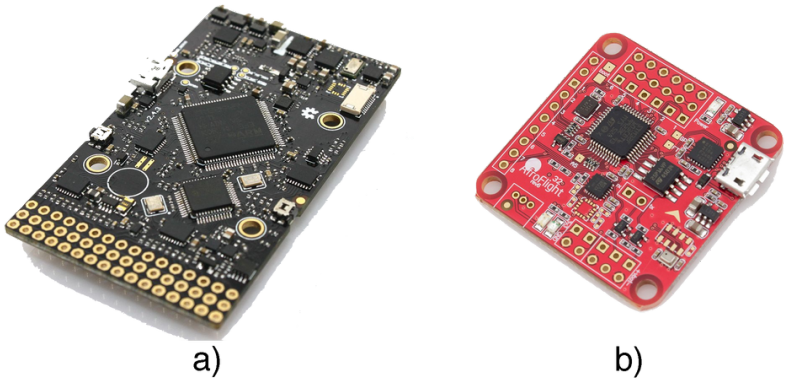


Fig. 2.28.: Flight controller board. a) Pixhawk b) MultiWii (Naze32)

2.7.1 Sensors

The basic sensory package on most FC includes accelerometers, gyroscopes, magnetometers and barometers. Specifications on the components of the flight controllers used in this project is showed in Tab. 2.2. Recent advances in the construction of micro-electromechanical systems (MEMS) (Burghartz, 2013) have made it possible to manufacture small and light motion sensors to be used on flight controllers. Accelerometers measure linear acceleration of the FC in the inertial reference frame, while gyroscopes measure the angular velocity of each axis of the reference frame. Magnetometers are used to correct attitude information and estimate drift of gyroscopes. Flight controllers can estimate the altitude relative to a take-off point based on pressure measurements from the barometer.

Description	**MultiWii (Naze32)**	**Pixhawk**
Processor	STM32F103CB 32-bit	STM32F427 Cortex M4
Frequency (MHz)	72	168
Accelerometer	MPU6500	MPU6000, LSM303D
Gyroscope	MPU6500	MPU6000, L3GD20
Magnetometer	HMC5883L	LSM303D
Barometer	MS5611	MS5611

Tab. 2.2.: Basic components of two (Pixhawk and MultiWii) flight controllers

2.7.2 Attitude estimation

Recently, there is a large academic and commercial activity on the topic of sensor fusion due to the variety of new on-board sensors that the MRUAV can carry. Data fusion refers to a variety of techniques, technologies, systems, and applications that use data derived from multiple information sources (Elmenreich, 2002). Fusion applications range from real-time sensor fusion for the navigation of mobile robots to the off-line fusion of human or technical strategic intelligence data (Rothman et al., 1991). Sensor fusion is the combination of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually. Systems that employ sensor fusion methods expect a number of benefits over single sensor systems such as sensor deprivation, limited spatial coverage, limited spatial coverage, imprecision and uncertainty. The two latter are common problems in the MRUAV sector. The main advantages of using sensor fusion techniques from a set of heterogeneous or homogeneous sensors are:

- Robustness and reliability: Multiple sensor suites have an inherent redundancy which enables the system to provide information even in case of partial failure

- Extended spatial and temporal coverage: One sensor can look where others cannot and vice versa
- Increased confidence: A measurement of one sensor is confirmed by measurements of other sensors
- Reduced ambiguity and uncertainty: Joint information reduces the set of ambiguous interpretations of the measured value
- Robustness against interference: By increasing the dimensionality of the measurement space the system becomes less vulnerable against interference
- Improved resolution: When multiple independent measurements of the same property are fused, the resolution of the resulting value is better than a single sensors measurement

When estimating orientation and heading, the best results are obtained by combining data from multiple types of sensors to take advantage of their relative strengths. Gyroscopes can be integrated to produce angle estimates that are reliable in short periods of time, but they will tend to drift in the long run. Accelerometers, on the other hand, are sensitive to vibration, but can be used in the long run to provide angle estimates that do not degrade (or drift) as time progresses.

Combining gyroscopes and accelerometers can produce a better attitude estimation, that is angle estimates that are resistant to vibration and immune to long-term angular drift. If a magnetometer is added, then it can help in the correction of the offsets created by the gyroscope through time and the accelerometer vibrations. The magnetometer will provide a heading estimation.

Extended Kalman Filter

Pixhawk project have designed an attitude estimation algorithm based on the extended Kalman filter (EKF). A more extended explanation of the algorithm can be found in Meier et al., 2011 and Simon, 2006. Let $\mathbf{p}$ and $\mathbf{v}$ be three-dimensional position and velocity in earth-fixed frame, $\mathbf{q}$ the quaternion, and $\mathbf{b}$ the gyroscope bias. Let $\mathbf{R_{eb}}(q)$ and $\Omega(\mathbf{q})$ be rotation matrix that converts body-fixed frame to earth-fixed frame and quaternion rates matrix, respectively, as a function of the unit quaternion. Let $\mathbf{a}$ denotes linear acceleration

in body-fixed frame and ω the angular velocity in body-fixed frame. Then, the state equation in discrete time can be written as

$$\mathbf{x_k} = \begin{bmatrix} \mathbf{p_k} \\ \mathbf{v_k} \\ \mathbf{q_k} \\ \mathbf{b_k} \end{bmatrix} = \begin{bmatrix} \mathbf{v_{k-1}} \\ \mathbf{R_{eb}}(\mathbf{q_{k-1}}) \cdot \mathbf{a_{k-1}} \\ \frac{1}{2}\Omega(\mathbf{q_{k-1}}) \cdot \omega_{\mathbf{k-1}} \\ \mathbf{W_{b,k-1}} \end{bmatrix} \tag{2.33}$$

where the gyroscope bias b is modelled with noise $\mathbf{W_b}$. The system input $\mathbf{u}$ consists of measurements of angular velocity $\omega_\mathbf{m}$ and linear acceleration $\mathbf{a_m}$:

$$\mathbf{u_k} = \begin{bmatrix} \omega_{\mathbf{m,k}} \\ \mathbf{a_{m,k}} \end{bmatrix} = \begin{bmatrix} \omega_{\mathbf{m,k}} - \mathbf{W}_{\omega,\mathbf{k}} + \mathbf{b_k} \\ \mathbf{a_k} - \mathbf{W_{a,k}} - \mathbf{R_{eb}^T}(\mathbf{q_k}) \begin{bmatrix} 0 & 0\mathbf{g} \end{bmatrix}^T \end{bmatrix} \tag{2.34}$$

where $\mathbf{W_a}$ and $\mathbf{W}_\omega$ represent noise and $\mathbf{g}$ is gravity. When substituting 2.34 into 2.33 the non-linear model is created:

$$\mathbf{x_k} = \mathbf{f}(\mathbf{x_{k-1}}, \mathbf{u_{k-1}}) + \mathbf{W_{k-1}} = \begin{bmatrix} \mathbf{v_{k-1}} \\ \mathbf{R_{eb}}(\mathbf{q_{k-1}})(\mathbf{a_{m,k-1}} + \mathbf{W_{a,k-1}}) + \begin{bmatrix} 0 & 0\mathbf{g} \end{bmatrix}^T \\ \frac{1}{2}\Omega(\mathbf{q_{k-1}})(\omega_{\mathbf{m,k-1}} + \mathbf{W}_{\omega,\mathbf{k-1}} - \mathbf{b_{k-1}}) \\ \mathbf{W_{b,k-1}} \end{bmatrix} \tag{2.35}$$

where $\mathbf{W_k} = \begin{bmatrix} \mathbf{W}_{\omega,\mathbf{k}}, & \mathbf{W_{a,k}} & \mathbf{W_{b,k}} \end{bmatrix}^T$ represents the process noise. The states are estimated by the standard EKF algorithm and measurements from accelerometers, gyroscopes, magnetometers, GPS, and barometer are fused to estimate the states. Being $\mathbf{m_b}$ the magnetic field of the Earth, $\mathbf{m_b}$ the one of the body frame, $\mathbf{P_z}$ the barometric pressure sensor reading, $\mathbf{h_b}$ the height relative to take-off and as before $\mathbf{v_k}$ the measurement noise. Then the non-linear measurement model is:

$$\mathbf{z_k} = \mathbf{h}(\mathbf{x_k}) + \mathbf{v_k} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{m_b} \\ \mathbf{h_b} \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \mathbf{R_{eb}^T}(\mathbf{q})\mathbf{m_e} \\ -\mathbf{P_z} \end{bmatrix} \tag{2.36}$$

Linear Complementary Filter

In the other analysed flight controller (MultiWii), a non-linear complementary filter is implemented with the rotation matrix representation, based in Mahony et al., 2005, with the only difference that (at the moment of this document being written) the MultiWii code does not ensure that the attitude estimate is SO(3) (3D rotation group) compatible (re-

orthogonalizing a DCM [Direction Cosine Matrix] or normalizing a quaternion), which leaves the corrections of errors in the rate integration due to the gyroscope imperfections to the effectiveness of the complementary filter. Therefore a linear complementary filter should be analysed first. A full description of a complementary filter can be seen at Oliveira et al., 2000. A LCF is designed to fuse multiple independent noisy measurements of the same signal that have complementary spectral characteristics. The complementary filter to estimate the angle θ is obtained:

$$\dot{\hat{\theta}} = y_u + k_p(y_x - \hat{\theta}) \tag{2.37}$$

being y_u the rate measurement and y_x the angle measured by accelerometer. $\hat{\theta}$ denotes the estimate of θ and k_p is a gain that determines crossover frequency. In this implementation the gyroscope bias varies over time. To compensate for this behaviour, a integrator is added:

$$\dot{\hat{\theta}} = y_u - \hat{b} + k_p(y_x - \hat{\theta}) \tag{2.38}$$

where, $\hat{b} = -k_i(y_x - \hat{\theta})$.

Non-linear Complementary Filter

If we use the LCF (2.38), then extended it to the non-linear SO(3) group Mahony et al., 2005 and finally add a bias estimate, the non-linear complementary can be seen at 2.39. The SO(3), often called 3D rotation group, is the group of all rotations about the origin of three-dimensional Euclidean space $\mathbb{R}^3$ under the operation of composition.

$$\dot{\hat{\mathbf{R}}} = \hat{\mathbf{R}}(\Omega_y - \hat{b} + \lambda)_\times \tag{2.39}$$

where,

$$\begin{aligned} \dot{\hat{b}} &= -k_i\lambda \\ \hat{b}(0) &= \hat{b}_0 \\ \lambda &= \text{vex}(\pi_a(\tilde{\mathbf{R}})) \\ \tilde{\mathbf{R}} &= \hat{\mathbf{R}}^T\mathbf{R}_y \end{aligned}$$

being $\pi_a(\tilde{\mathbf{R}}) = 1/2(\tilde{\mathbf{R}} - \tilde{\mathbf{R}}^T)$ and $\tilde{\mathbf{R}}$, $\tilde{\mathbf{R}} \in$ SO(3) attitude estimate and estimate error, respectively. The operator $\text{vex} : \text{SO(3)} \to \mathbb{R}^3$ denotes the inverse operation of a skew-symmetric matrix. $\mathbf{R}_y$ is the rotation matrix reconstructed using roll and pitch measured

from the accelerometer. Being Ω_y the measurement from the three-axis gyroscope of the flight controller. $\mathbf{R}$ has to satisfy the constrain $\mathbf{R}^T\mathbf{R} = \mathbf{I}$, and therefore the computation load becomes an issue in implementing this on an embedded system, this is why MultiWii has implemented this algorithm with the rotation matrix representation. Its also called DCM (Direction Cosine Matrix) and it consists of cosines of angles of all possible combinations of body and global vectors (Starlino, 2011).

2.8 Endurance Prediction

In order to explain better the methods derived and used in this document, we will perform calculations on a vehicle built-in-house at the MAST Lab. This test-bed vehicle was designed and built for in-doors applications, but if a GPS component is added, it can perform automatic flights out-doors. The components of this vehicle can be seen on table 2.3.

Part	**Description**
Frame	Glass fiber, $330mm$ rotor to rotor
ESC	Multistar $15amps$
Motor	Turnigy SK3 2826, $1130kv$
Propeller	Several (tested: $7x3.8in$ two/three blade)
Avionics	Altax Flight Stack
Weight (less battery)	$642grams$
Battery	LiPo, $3S$, $2.2ah$, $40-50C$ discharge rate

Tab. 2.3.: Test-bed vehicle components/information.

2.8.1 Time of flight

The estimated time of flight is a very important parameter when designing a multirotor, yet its not an easy parameter to calculate or estimate (without experimental tests), and often you need to assume factors including meteorological (wind speed, temperature, air density), energy losses, among others. In this analysis we will focus on calculating the time of flight when the vehicle is hovering in laboratory conditions. Using the equations derived in previous Sections 2.3 and 2.6, it is possible to accurately estimate the time of flight while a vehicle is hovering, or holding a specific position. This parameter will allow us to change components of the vehicle in order to make it perform better for the mission at hand. Using

equation 2.30 and 2.31 combined with Newton Second Law of Motion, the *rotor* power required for hover for a multirotor with four rotors is

$$\mathbf{P}_{rotor} = \sqrt{\frac{2(\frac{m_{veh}}{4}g)^3}{\pi \mathbf{D_{prop}}^2 \rho}} \tag{2.40}$$

By summing all of the consumed power, an approach to calculate time of flight while hovering is:

$$\mathbf{T}_{flight} = \eta_{factor} \frac{60}{1000} \frac{\mathbf{C}_{batt}\mathbf{v}_{nom}}{\mathbf{P}_{rotors} + \mathbf{P}_{avionics}} \tag{2.41}$$

The power consumed by the avionics is considered as a constant, while the power required from the rotors can be calculated from experimental data as showed in equation 2.32, which is a polynomial fit of the gathered experimental data using the rotor analysis tool.

2.8.2 On-line calculators

There are available on-line several tools that help to to calculate, estimate, evaluate and design electric motor driven systems for RC (remote controlled) models, being the most popular and accurate, one that is called *eCalc*[3]. It will be used in this research as a common method to compare results of theoretical and experimental calculations regarding the design of multirotors. As reported by the author of the tool, *eCalc* has a $\pm 20\%$ of accuracy when estimating the time of flight of a vehicle with the selected components. This type of tool contains a database with experimental data from propellers, electric motors, batteries among other components. This data is used to calculate, based on equations similar to the ones presented on this document, the hover and mixed flight time of a vehicle and several other parameters including electric power, mechanical power, maximum payload possible, efficiency and security parameters coming from the manufacturers data-sheets in order to keep the elements in the operational range. Its important to notice that the simulation models and information are a key factor for *eCalc* business, therefore private and inaccessible. This service (*eCalc*) charges a yearly subscription in order to get access to all of the components that they have analysed and tested.

2.8.3 Flight tests

The real flight tests where performed using a automatic pilot framework, called *DronePilot*, this framework will be presented in Chapter 3.5. The main advantage of performing this flight tests using this framework is to avoid pilot fatigue. The objective of this tests is to

[3] `http://www.ecalc.ch/`

hover at a specific location in space inside a laboratory (MAST Lab 3.2) while maintaining a certain altitude above the floor. A human pilot is capable of performing this test, but pilot fatigue can affect its performance capability affecting the repeatability of the tests. Therefore, using an automatic pilot it is possible to repeat experiments with the same flight performance, ensuring that the results are indeed comparable when changing hardware like propellers and batteries.

2.8.4 Method comparison

In the table 2.4, we can see the flight times comparison of the test-bed vehicle (Fig. 2.29), varying the propellers and the methods to calculate the time. The *eCalc* flight time are the estimation of the on-line calculator. The experimental flight time (Theory/Experimental), is calculated using equation 2.40 and the experimental data from the rotor analysis tool. While the real ones come from an experiment where the vehicle is being flown by *DronePilot* using motion capture data as reference and holding a desired position and altitude. In

Flight Time	**Two Blade**	**Three Blade**
eCalc	9 min 48 sec	10 min 57 sec
Theory/Experimental	10 min 11 sec	11 min 19 sec
Real Flight	10 min 21 sec	11 min 29 sec

Tab. 2.4.: Flight times computation comparison.

this experiments, the time of flight predictions from *eCalc* are less accurate than our theory/experimental method, but their estimates are useful when there is no experimental data available. Using this vehicle configuration the results showed that having the same

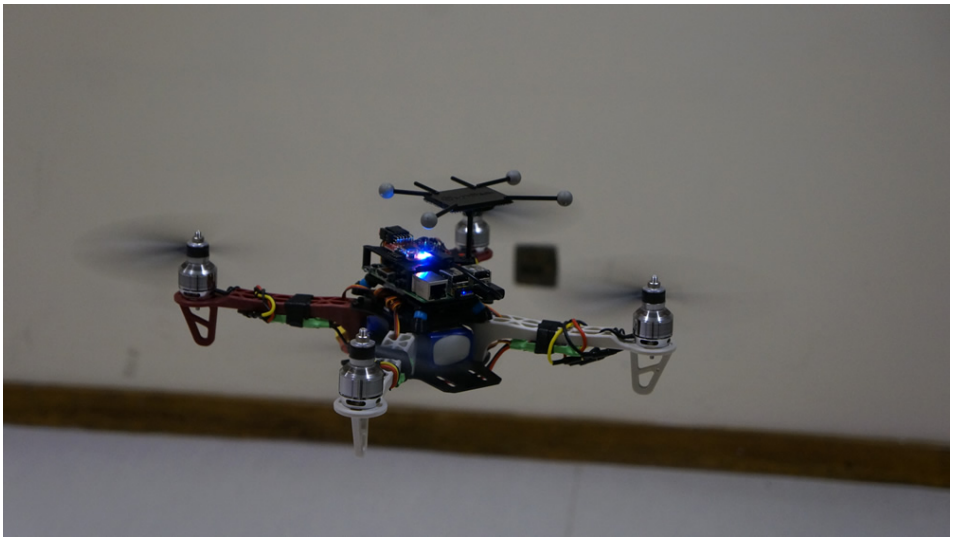


Fig. 2.29.: Quadrotor performing a hover test using three bladed propellers.

vehicle weight (± 10 grams difference), but with the added thrust generated by the extra

blade in the rotor, thus reducing the effort to the battery while holding a specific altitude of one meter (to avoid ground effect problems) the flight times are increased by 9.8%. The time of flight endurance predictions using the method showed in section 2.8 were off by 1.63% while the predicted using the on-line method were 5.61% off, thus proving that our methodology outperforms the endurance prediction.

2.9 Summary

This chapter introduced and described the various components needed to build a multirotor vehicle. A method for estimating the time of flight of a multirotor was presented and compared with other methodologies such as an on-line estimator calculator. It was shown using experimental data that the method presented in this chapter is closer to the real flight times of the test-bed vehicle than the time obtained using the on-line estimator.

3

Laboratory Set-up

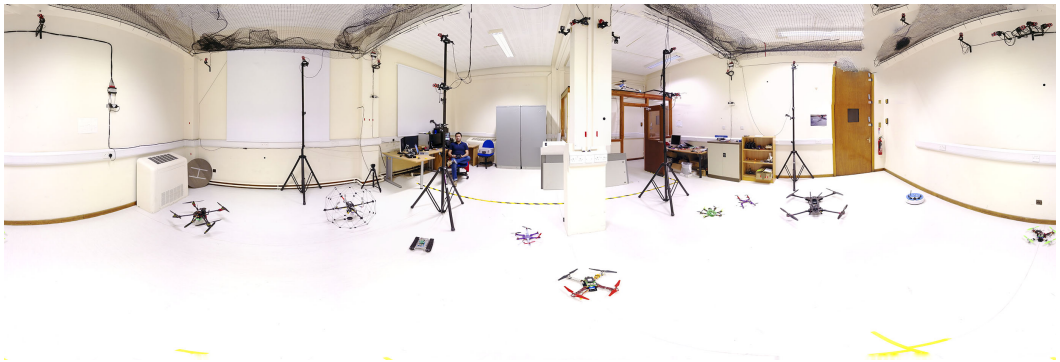


Fig. 3.1.: MAST Lab flight area and vehicles.

In this chapter one of the contributions of this thesis is presented, such contribution relates to the avionics solution (*Flight Stack*) used throughout the development of this work. The flight stack was created for data collection, algorithm execution and control of MRUAV. A key consideration in the design of the flight stack is the distribution of computation between on-board and external processing and communication between vehicles with external systems. With experimental results presented on Chapters 4, 6, 7 the flight stack shows successful autonomous flight with a range of algorithms and applications.

Also in this chapter, the set-up of the Micro Air Systems Technology Laboratory (MAST Lab) and the test-bed vehicle is presented, explained and analysed. The MAST Lab provides the ideal platform for research and investigation of small-scale autonomous vehicles and their associated technologies. This laboratory compromises state-of-the-art facilities and resources which will be described and discussed. One of the most important resources in the laboratory is the motion capture system. This system can provide the position and orientation of reflective markers that can be placed on a multi-rotor, therefore behaving like a very precise indoors positioning system for MRUAV.

As discussed in Chapter 2, multirotors are complex systems with many design constraints, it is not practical to include all functionality within a single general avionics board (flight controller), as has been the approach on the past as shown on Fig. 8.2. Instead, an improved approach taken is to adopt a modular design of the avionics suite, we will call this one the flight stack. A software framework is needed for the companion computer, it is presented and analysed in this chapter. This framework is called *DronePilot* (Vargas, 2014).

3.1 Experimental design

The ability to design good scientific experiments is a key skill for any researcher. Without a grasp of experimental design, even the best technological developments may never succeed in being published or turning into real products. In the past years, there has been much discussion within the robotic research community about a lack of standard experimental procedures (Antonelli, 2015). This is due to the lack of uniformly good experimental work and reporting within the robotics community as a whole. Experimental practices are usually learned from existing research papers, which can be of varying quality. This means that robotic researchers sometimes don't even learn the basics of experimental design. Experimental design will be different depending on the type of robotics being researched. An experiment with multirotors will be very different from an experiment with surgical robots. However, there are still common aspects between the two.

3.2 Micro Air Systems Technology Laboratory

The research focus of the MAST Lab (Fig. 3.1) is on developing new control methods and algorithms for UAVs. We are interested in coordinating actions with multiple UAVs and methods for dealing with the interactions between them. In order to perform autonomous control of UAVs the system requires some sort of feedback, the internal AHRS (Attitude Heading Reference System) sensors on the flight controller could produce errors and/or in some cases it can produce drifting. This is the reason for the use complex algorithms like the ones discussed on the attitude estimation Section 2.7.2. Thus it is necessary to gather position data from an external source, like GPS (Global Positioning System), however, the GPS signals will not penetrate the building walls to provide the position of the vehicle in our indoors laboratory. A special indoor positioning system is therefore required.

3.2.1 Indoor positioning system

IPS (Indoor Positioning System) is a system that helps in the location of objects or people inside a building using radio waves, magnetic fields, acoustic signals, or other sensory information collected by mobile devices. There are several commercial systems on the market, but there is no standard for an IPS system. In the MAST Lab case, the IPS uses reflective markers and infra-red cameras. This technique is called motion capture (MoCap).

Motion Capture

Motion capture is the process of recording the movement of objects or people. It is used in military, entertainment, sports, medical applications, and for validation of computer vision and robotics. This type of optical system utilizes data captured from image sensors (cameras) to triangulate the 3D position of an object. Two or more cameras are required to provide overlapping projections. Data acquisition is traditionally implemented using special markers attached to the object. The MoCap system is used to get the position and attitude of the vehicle being analysed and controlled inside the MAST Lab. It acts as a extremely precise indoors GPS. The motion capture system implementation of the MAST Lab compromises 18 OptiTrack^{TM} V100 cameras that point to various areas of the flight area. The cameras emit infra-red (IR) light, using an array of IR-LEDs, then the reflective markers on-board the aircraft reflect this IR light. This enables the cameras to identify and detect individual markers on the aircraft and then compute the markers body position and orientation. The distribution of the MoCap cameras can be seen in figure 3.2. This system

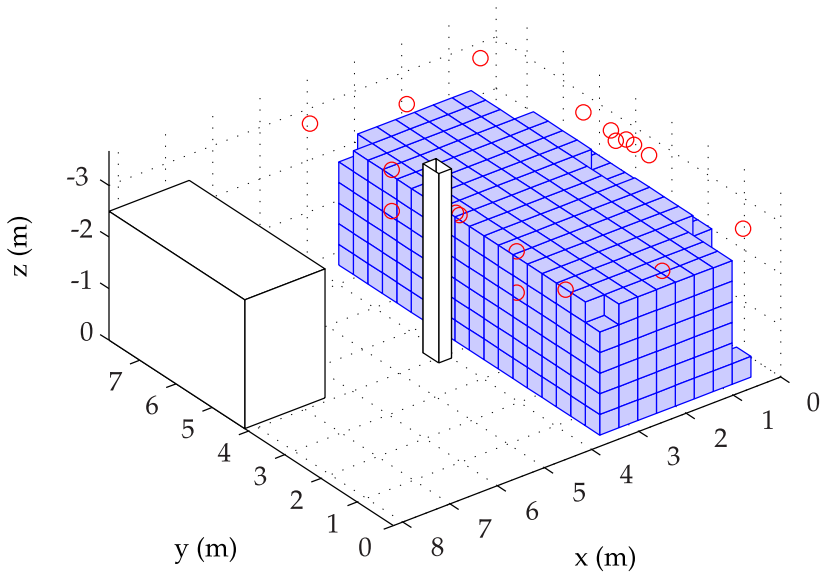


Fig. 3.2.: Distribution of IPS image sensors in the MAST Lab.

estimates the states of the vehicles at $100Hz$. It can also identify position estimates in the range of millimetres which is beyond the requirements of the test flights. This MoCap system is connected to a ground station computer which computes the orientation and position of the markers, and transmits this information to the flight stack.

3.2.2 Previous data flow

Before the flight stack was designed and tested, the first GNC algorithms were performed on a different structure shown in Fig. 3.3. This structure was limited by several factors including:

- Limited bandwidth and baud rate of the RF (Radio Frequency) radio
- MATLAB application crashes (software application stops functioning properly)
- Arduino-based (Atmel AVR 8-bit CPU) flight controller with a slower attitude stabilization control loop time than 32-bit counterparts ($80Hz$ vs $243Hz$)
- Guidance and navigation implemented in an external computer, not on-board the vehicle

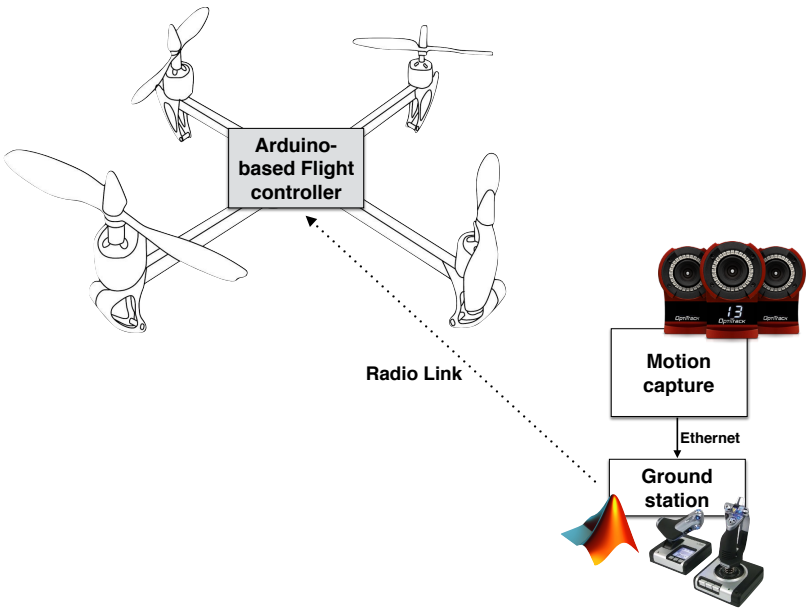


Fig. 3.3.: Previous structure to control MRUAV in the MAST Lab.

These factors affected significantly the performance of the vehicle and controllers. The limitations also caused several accidents when either the MATLAB software or the RF suffered a disruption. One of the biggest problem was the limited rates in the communication link, this factor affected the outer control loop, making it extremely complicated to tune in order to make the vehicle accurately track a trajectory.

3.3 Flight Stack

Due to multi-rotors being inherently unstable, they require active stabilisation. This stabilisation is deployed on a flight controller board, which contains sensors that provide

feedback. The stabilisation of a multi-rotor is time sensitive and thus the flight controller must run a real time operating system (RTOS). The Pixhawk project (Meier et al., 2011) includes a RTOS system called *NuttX* that has an emphasis on standards compliance and small footprint, scalable from 8-bit to 32-bit micro-controller environments. If we consider the addition of a companion computer to the avionics system, we can increase the available functionality, e.g. interfacing with more hardware, providing a development environment, communication protocols, etc. Consequentially, the computational resources available for running guidance and navigation algorithms receives a considerable increase. This system is named the flight stack. A key consideration in the design of the flight stack is the dis-

Fig. 3.4.: Flight Stack using a Raspberry Pi with: a) Naze32 b) Pixhawk

tribution of computation between on-board and external processing and communication between vehicles and with external systems. The flight stack can be defined as a system used to control the stability/trajectory of a UAV without constant control by a human operator being required. The flight stack is a tuple of computer systems that work cooperatively in order to achieve a particular mission. This proposed flight stack is composed of a flight controller, companion computer and communications systems. In the GNC (Guidance, Navigation and Control) architecture, the flight controller (*inner loop computer*) is in charge of the Control, that is the stability and fly-ability of the vehicle while the companion computer (*outer loop computer*) is in charge of the Guidance and Navigation. It guides the vehicle to a specific location with a pre-programmed navigation course. Figure 3.4(a) shows a flight stack ready to be mounted on a vehicle, this particular model is composed of a Raspberry Pi as companion computer and a Naze32 as flight controller. A similar flight stack using a Pixhawk as flight controller is displayed on figure 3.4(b).

3.3.1 Companion computers

The outer loop computer is in charge of guiding the vehicle through a specified trajectory or flight plan. It also handles the communication with the flight controller and with the

user on the ground station. There are currently several promising solutions on the market for on-board processing to act as companion computers, we decided to focus on credit card-sized single board computers that use Linux-kernel-based operative systems, due to their size, weight and power requirement. Two companion computers series were tested during the effort of this research, one is the *Raspberry Pi* (Mitchell, 2012) shown in figure 3.5 (a) and the *Odroid* series (Hardkernel Co., 2014) as displayed in figure 3.5 (b) and (c). There are three versions of Raspberry Pi that can be used for this flight stack, Raspberry Pi B+, Raspberry Pi 2 and Raspberry Pi 3. The Odroid versions tested on the flight stack were the U3 and the XU4. Table 3.1 shows some of the main characteristics of the single-board

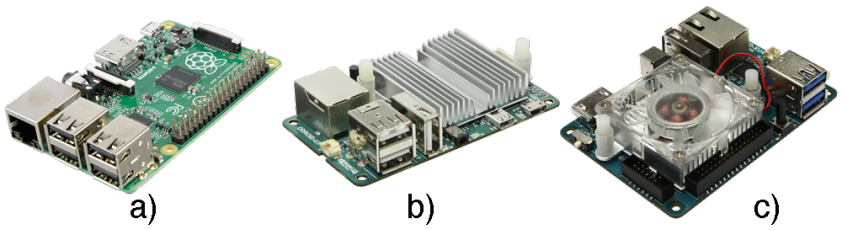


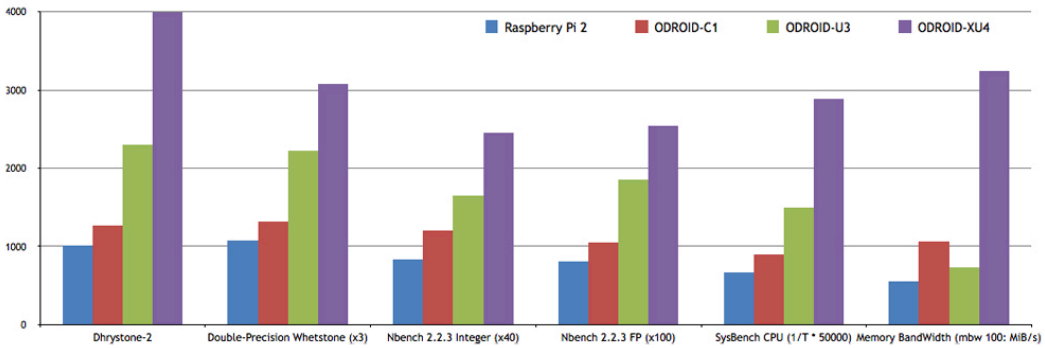
Fig. 3.5.: Companion Computers. a) Raspberry Pi b) Odroid U3 c) Odroid XU4

computers used in the flight stack. The most powerful of them is the *Odroid XU4*. Figure 3.6

Description	**RaspberryPi 2**	**RaspberryPi 3**	**Odroid U3**	**Odroid XU4**
Processor	Quad-core ARM Cortex-A7	64-bit Quad-core ARMv8	Exynos 4412 Prime	Exynos 5422 Octa ARM Cortex-A15
Frequency	900 MHz	1.2 GHz	1.7 GHz	2.0 GHz
RAM	1 GB	1 GB	2 GB DDR2	2 GB DDR3
Storage	MicroSD	MicroSD	MicroSD, eMMC	MicroSD, eMMC
USB ports	4	4	3	3
Weight	45 g	45 g	48 g	60 g
Power consumption	3.0 W	4.0 W	3.8 W	4.6 W

Tab. 3.1.: Companion computers main characteristics.

shows several benchmarks to measure the computing power on the Raspberry Pi 2, Odroid C1 (not used on the flight stack), Odroid U3 and Odroid XU4. The computing power of the XU4 was measured to be 3-4 times faster than the Raspberry Pi 2. The U3 and XU4 can boot from a MicroSD card or an eMMC module. An eMMC (embedded multimedia card) module outperforms a standard SD card in the capacity of read and write files. This is a great advantage for the flight stack to boot the operating system and log data to files faster (black box functionality).



Benchmarks (Index Score)	Raspberry Pi 2	ODROID-C1	ODROID-U3	ODROID-XU4
Dhrystone-2	1006.6	1262.8	2300.7	3994.1
Double-Precision Whetstone (x3)	1076.1	1318.8	2217.9	3074.7
Nbench 2.2.3 Integer (x40)	840.0	1208.0	1648.0	2451.1
Nbench 2.2.3 FP (x100)	809.0	1050.0	1860.0	2536.9
SysBench CPU (1/T * 50000)	669.3	902.2	1497.0	2891.8
Memory BandWidth (mbw 100: MiB/s)	557.1	1069.6	736.6	3243.8

Credit: *Hardkernel*

Fig. 3.6.: Benchmark comparison.

Operating system

It was decided to focus on companion computers that use Linux-kernel-based operating systems, in this way we can use this flight stack (methodologies and software) fitting not only one specific companion computer but several, ensuring scalability and upgrade-ability of the flight stack. Debian and Ubuntu are the most influential Linux distributions (Tabassum et al., 2014). The Raspberry Pi can use a Linux-kernel operating system (Raspbian) based on the Debian ARM hard-float (armhf) architecture port. Being based on Debian, Raspbian comes with the APT (Advanced Packaging Tool) as its package manager, which is used to install software from the vast Raspbian repositories. The Raspberry Pi 2 comprises an ARM7 CPU architecture, therefore now compatible with Ubuntu. One of the main advantages of using this distribution of Linux is that the company release a special version called *Long-term support* (LTS). Long-term support includes updates for new hardware, security patches and updates to the *Ubuntu stack* (cloud computing infrastructure). This computer software product life-cycle management policy ensures the reliability of the flight stack for a specified period of time. Long-term support not only extends the period of software maintenance, it also alters the type and frequency of software updates (patches) to reduce the risk, expense, and disruption of software deployment, while promoting the dependability of the software. Another advantage of using Ubuntu LTS as operating system on the flight stack is to have full support of ROS (Robot Operative System) on each LTS release. Although the DronePilot framework does not include ROS support (at the moment of this document being written), its important to keep the compatibility with such tools.

3.4 DronePilot

A framework is a reusable set of libraries, scripts, tool sets and classes for a software system. DronePilot is a software framework with the aim of controlling Unmanned Aerial Vehicles, also called *drones*. This framework was developed due to the necessity of integrating several software components in order to be able to control multi-rotor vehicles inside the MAST Lab. It is an open source project, based on a GNU license, released to be used widely on academia, it is well documented and being updated constantly. The main functionalities of DronePilot are:

- Interfacing with flight controllers (MultiWii and Pixhawk).
- Bridging between the ground station computer and the inner loop computer, this allows the control of vehicles from the ground station.
- Black box that records flight data on every iteration of the outer loop.
- Autonomous control of multi-rotors for various applications.

The majority of the software used in the DronePilot framework is written in the widely used general-purpose, high-level programming language Python. Python is a dynamic object-oriented programming language, it offers strong support for integrating with other technologies, higher programmer productivity throughout the development life cycle, and is particularly well suited for large or complex projects with changing requirements. Python was designed from the ground up to be embeddable, which is of great advantage if the outer loop computer changes. Python is very well designed, fast, robust, portable, and scalable because it has been developed as an open source project by thousands of contributors (Rossum et al., 2011). With an uncluttered, easy-to-learn syntax and well-developed advanced language features, Python often exceeds the capabilities of comparable commercially available solutions.

3.4.1 Core

The core of the DronePilot framework is a multi-threaded structure to ensure the transition and rates between on-board and external computation. This property is called concurrency and it involves arbitrary and dynamic patterns of communication and interaction between parts of the algorithm. The base goals of the core in the framework include correctness, performance and robustness. Because the framework uses shared resources, it require the inclusion of some kind of arbiter in the implementation. DronePilot makes use of the Global Interpreter Lock (GIL), provided by the Python programming language.

The most common script uses at least two threads. Running several threads is similar to running several different programs concurrently, with the benefit that the process shares the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes. The other advantage is some threads behave like light-weight processes and they do not require much memory overhead, therefore being more computationally cheap that using separated process. One disadvantage of using GIL is that the usage of arbiters introduces the possibility of indeterminacy in concurrent computation which has major implications for practice including partial correctness and performance.

Figure 3.7 shows how objects operate with one another and in what order, this is also called a sequence diagram. This diagram shows the most simplistic application using the DronePilot framework. This application includes only two threads. The *Comms* thread is in charge of the communications from the ground station to the vehicle, receiving and making the data available for the rest of the threads and functions. This data includes the pilot commands (from a joystick), for when the vehicle is being flown manually and the other set of data are the vehicle states (position and orientation), that comes from the MoCap system.

The second thread (*Pilot*) is the one that is in charge of processing the information from the *Comms* thread, interfacing with the flight controller and recording timestamped data for further analysis.

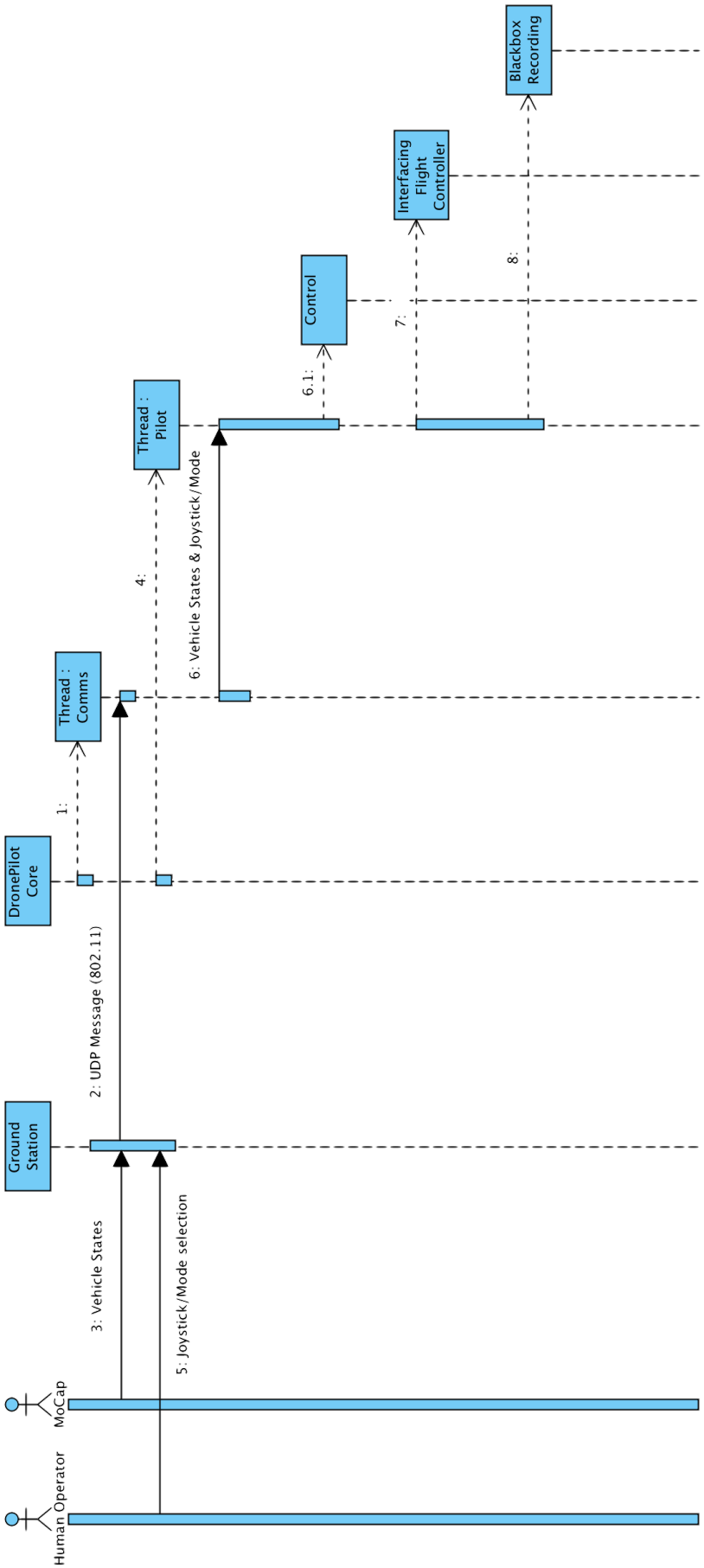


Fig. 3.7.: Sequence diagram of a simple DronePilot application.

3.4.2 Data Flow

Using Python gives the advantage of using complex multi-protocol network applications such as Twisted, a development framework well suited to running large numbers of concurrent network, database, and inter-process communication links within the same process. This module is used for the communication between the ground station and the vehicle. Twisted is an event-driven networking engine written in Python and licensed under the open source MIT license (Lefkowitz, 2002). This module was chosen due to the high rate

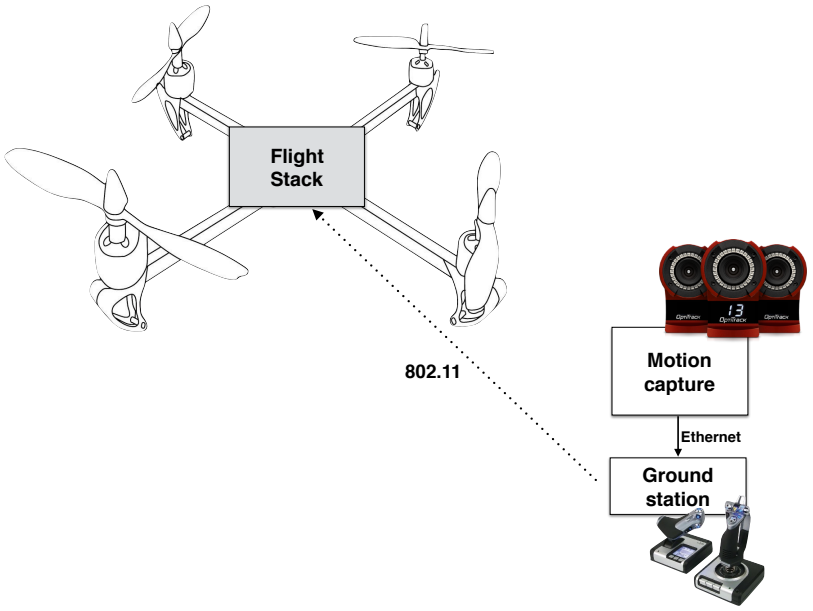


Fig. 3.8.: Data flow.

of communication that can achieved while using the User Datagram Protocol (UDP), which is one of the core members of the Internet protocol suite („User Datagram Protocol“). UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system (Kurose et al., 2013). The data flow is showed on figure 3.8. The motion capture system is connected, via 802.3 (IEEE, 2013) to a ground station computer running Matlab/Simulink to get the state estimation of the vehicle and the inputs from a joystick device. This data is sent to the companion computer via a 802.11 (Crow et al., 1997) network. Inside the companion computer, the UDP datagrams are received inside the *Comms* thread and then processed by the *Pilot* thread, then the new computed

command is sent via USB to the flight controller which then translates into changes of the propeller speeds of the rotors. The rate at which everything happens is $100Hz$.

3.4.3 Interfacing

In order for the *Pilot* thread to be able to interact with the proposed flight controllers (2.7), the DronePilot framework needs to *talk* the language of the flight controllers. This is the MultiWii Serial Protocol (MSP) and MAVlink (Micro Air Vehicle Link) respectively.

MSP

A Python module named *pyMultiWii* (Vargas, 2013b) was created using the same open source methodology (GNU license) in order to communicate the companion computer with MultiWii enabled flight controllers using the MSP (MultiWii Serial Protocol). More details about this module can be found on Appendix A.4. MSP is a protocol designed by the MultiWii community, with the idea to be light, generic, bit wire efficient, secure. The MSP data frames are structured as showed on figure A.6. DronePilot uses *pyMultiWii* to ask

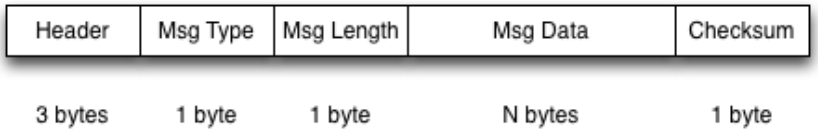
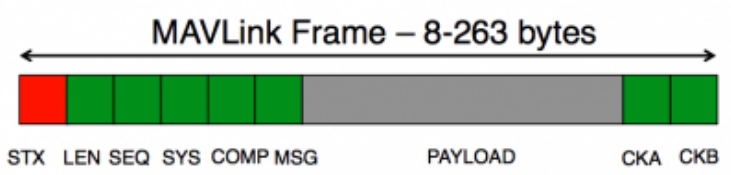


Fig. 3.9.: MSP data frame.

the flight controller for the orientation of the board and for sending angle commands that the flight controller must track. More information about this controlling methodology is showed on Chapter 4.

MAVlink

MAVLink is a protocol for communicating with small unmanned vehicle (Meier, 2009). It is designed as a header-only message marshaling library. MAVlink packs C-structs over serial channels with high efficiency and is extensively tested on the PX4, PIXHAWK, APM and Parrot AR.Drone platforms and serves there as communication backbone for the MCU/IMU communication as well as for Linux interprocess. DronePilot does not *talk* the MAVlink protocol directly, instead it uses extra python modules that have this functionality for it, thus making the full integration of MAVlink more simple and reliable. These modules are MAVProxy (Tridgell, 2013).



Credit: *QGroundControl*

Fig. 3.10.: MAVlink data frame.

3.4.4 Black box

Multirotors in any condition of flight can be viewed in terms of its input (e.g. pilot commands and/or desired trajectories) and output parameters (e.g. orientation and position), without any knowledge of its internal workings, colloquially known as a black box model. One of the flight recorder capabilities of DronePilot is to allow *replay* of a flight performed previously. This is in order to analyse the flight performance, check for errors on the control algorithms, or even perform techniques such as system identification (Chapter 6). This

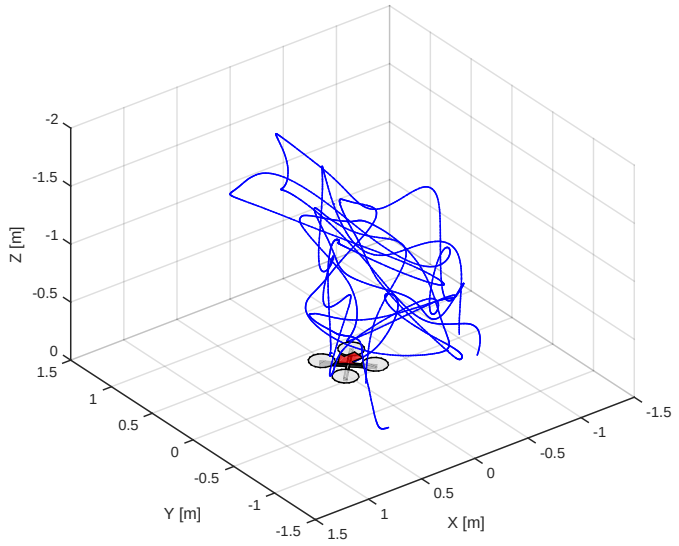


Fig. 3.11.: 3D plotting of a quadrotor trajectory flight.

functionality saves data as a timestamped Comma Separated Values (CSV) file, the flight recording is done at the same rate as the outer loop performs, usually is $100Hz$, or a row recorded each 0.01 seconds. Figure 3.11 shows a 3D plot of a flight recorded in order to perform system identification on the test-bed quadrotor, the data plotted in such image is position of the vehicle, but in order to perform the system identification techniques, the inputs to the systems are also needed.

3.4.5 Applications

After describing the flight stack, the main objective is to reduce the amount of effort when designing new and novel algorithms for MRUAV, the entire flight stack can be considered as a single system, instead of combination of computers like it is, a diagram of how can be considered is showed on figure 3.12. The type of applications that DronePilot was designed

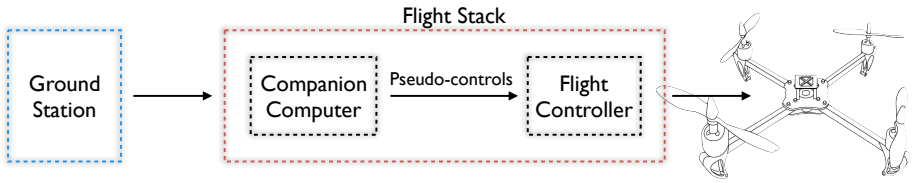


Fig. 3.12.: FlightStack single system diagram.

for include:

- Manual indoor flight
- Autonomous indoor flight
- System Identification of Multirotors
- Control of novel multirotor configurations
- Automation of performance tests on rotors
- Test of state-of-the-art control algorithms
- Control of multirotors with a slung load
- Swarm control of multirotors

Most the applications showed on the list above will be analysed in later sections.

3.5 Test-bed quadrotor

Test-beds are especially important for multi-robot research with UAVs as well as the flight stack, a quadrotor test-bed (Fig. 3.13) was develop in the MAST Lab using 3D printing technologies and Consumer Off-The-Shelf (COTS) components. The flight stack proposed in this section was used to control the quadrotor and log the data needed for several research papers, including (Vargas et al., 2014).

Although there are several platforms on the market, one of the advantages of building our own platform is economical, a MRUAV in a quadrotor configuration similar to our own is the Asctec Pelican (*Ascending Technologies*), which is 13 times as expensive as than the

Fig. 3.13.: Test-bed quadrotor hovering.

Component	Description
Motor	Turnigy SK3 2826, $1130kv$
Propeller	APC $7x3.8in$
ESC	Multistar $15amps$
Avionics	AltaX Flight Stack

Tab. 3.2.: Test-bed quadrotor component list.

one produced in the MAST Lab. This allows the construction of several models without financial limits. This test-bed uses the same rotor components analysed on section 2.8, the components are shown on Table 3.2.

Figure 3.14 shows a slightly different version of the test-bed, using an Odroid U3 as companion computer, and on top of the vehicle an *asymmetrical* array of four reflective markers (for the MoCap) is displayed.

3.6 Summary

This chapter presented the configuration of the MAST Lab, also a comparison of the previous methodology with the new methodology is presented. Such new methodology provides better performance and capabilities to carry out research and investigation of small-scale autonomous vehicles and their associated technologies.

This chapter also introduced the concept of the flight stack as well as the components included in it and the DronePilot framework, which is one of the contributions of this thesis. The next chapter will deal with the modelling and control of multirotor vehicles.

Fig. 3.14.: Test-bed quadrotor with an Odroid U3 as companion computer.

4 Quadrotor Modelling and Control

Multirotors are underactuated non-linear dynamically unstable systems that present challenges when attempting to model and control them. The model of the MRUAV system, as in other engineering problems (Gonzalez-Olvera et al., 2010), is a crucial part of the analysis and design of controllers. The study of the MRUAV kinematics and dynamics helps to understand its physics and behaviour. Together with the modelling, the determination of the control algorithm structure is fundamental to achieve optimal stabilization in order to be able to solve the research question of this thesis. The most common multirotor used in research labs around the world is the quadrotor. In this section, the derivation of the quadrotor model is presented. With the model introduced, a position and trajectory controller is presented followed by experimental tests and results.

4.1 Basic concepts

As stated in Chapter 2, a quadrotor is modelled with four rotors in a cross configuration. Each propeller is connected to the motor directly, all of the propellers axes of rotation are fixed and parallel. The rotors contain fixed-pitch blades and the air flows downwards in order to lift the vehicle upwards. These considerations point out that the structure is quite rigid and the only things that can change are the propeller speeds. In Chapter 2, the motor and propellers were analysed. In this section, neither the motors nor the propellers dynamics are fundamental because the movements are directly related just to the propellers velocities. Another neglected component is the flight stack, which is not essential to understand how the quadrotor flies. The basic model to evaluate the quadrotor movements it is composed of just a thin cross structure with four propellers on its ends. The front and the rear propellers rotate counter-clockwise, while the left and the right ones turn clockwise. Figure 4.1 shows the structure model in hovering condition, where all the propellers have the same speed. The fixed-body is shown with an arrow to the centre of the frame and the black-thick arrows at the end of the cross axes represent the angular speed of the propellers. That same black-thick arrow shows the propeller direction of rotation. In figure 4.1 all propellers rotate at same speed (hovering condition) in order to counterbalance the acceleration due to gravity, therefore the vehicle performs stationary flight and no forces or torques move it from its position, this is called hover flight. In reality the vehicle will drift

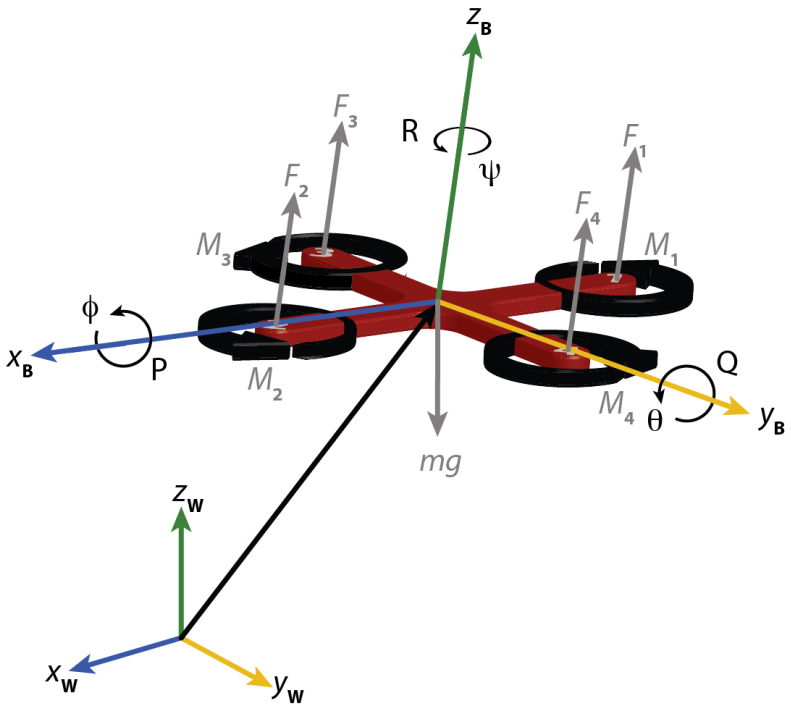


Fig. 4.1.: Simplified quadrotor in hovering.

and the inputs to it should be controlled in order to make it perform stationary flight. This will be described later on this section. The quadrotor has six DOF, even if it's equipped with four actuators, hence it is not possible to reach a desired set-point for all the degrees of freedom, but at maximum four, which are quite easy to choose the best ones and decoupled them to make the controller easier to design. These four quadrotor pseudo-controls are thus related to the four basic movements which allow the vehicle to reach a desired altitude and orientation, called *throttle*, *roll*, *pitch* and *yaw*. We will call these ones the pilot inputs.

4.1.1 Pseudo-controls

There are two possible *configurations* to control a quadrotor, one is called plus ($\oplus$) while the other is cross ($\otimes$) both will be explained. These differ in several factors, even with identical hardware and software configuration. For a quadrotor with arm of length L from the centre of mass of the vehicle, in the $\oplus$ configuration the thrust forces are applied at a distance L, meanwhile in the $\otimes$ configuration the thrust forces are applied at a distance of $L\cos(\pi/4)$, approximately at $0.71L$ since the arms are at a $45°$ angle from the axis of rotation. The moment of inertia behaves similarly, so the difference is really that the torque can be applied with all four rotors, and therefore have $\sqrt{2}$ more available torque to rotate. This means you can get about 41% more rotational acceleration from an $\otimes$ than a $\oplus$ configuration. The next difference is the vehicle visibility, $\oplus$ will *usually* have one marked front arm, while the

$\otimes$ will have two marks signalling front, this gives an advantage of more visibility to the $\otimes$ configuration. The last difference is *camera arc clearance* when a $\otimes$ can have a camera pointed forward without obstruction from the frame more easily than a $\oplus$. Some designs remove this concern, either by mounting the camera on a stand-off or with moving landing gear. The most obvious difference is how to control the rotor speeds in order to move the vehicle in the desired set-points.

Throttle (u_1)

This command is created by increasing (or decreasing) all of the propeller speeds by the same amount, this leads to a vertical force in the body-fixed frame which raises and lowers the quadrotor. Figure 4.3 shows the throttle command on a quadrotor sketch at hover conditions, with all propellers spinning at ω_{Hover}.

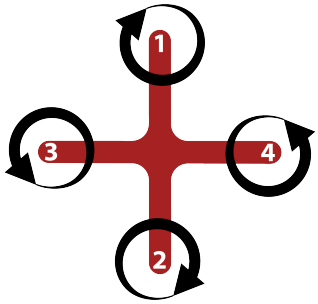


Fig. 4.2.: $\oplus$ Plus configuration Throttle command diagram.

Roll (u_2)

For $\oplus$ this command is created by increasing (or decreasing) the left propeller speed and by decreasing (or increasing) the right one. For $\otimes$ this command is created by increasing (or decreasing) the two left propellers speed and by decreasing (or increasing) the two right ones. Such command leads to a torque with respect to the x axis which makes the quadrotor rotate around that axis and makes the vehicle translate left or right. Figure 4.3 shows the roll command on a quadrotor sketch showing faster ω_3 (thicker arrow) and slower ω_4, while ω_1 and ω_2 remain at ω_{Hover}, this combination will make the quadrotor roll to the right $\rightarrow$.

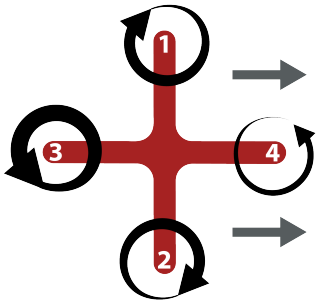


Fig. 4.3.: $\oplus$ Plus configuration Roll command diagram.

Pitch (u_3)

For $\oplus$ this command is created by increasing (or decreasing) the front propeller speed and by decreasing (or increasing) the back one. For $\otimes$ this command is created by increasing (or decreasing) the two front propellers speed and by decreasing (or increasing) the two back ones. This command leads to a torque with respect to the y axis which makes the quadrotor

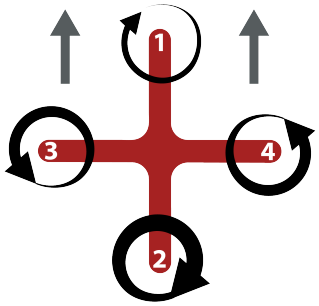


Fig. 4.4.: $\oplus$ Plus configuration Pitch command diagram.

rotate around that axis and makes the vehicle move forward and backward. Figure 4.4 shows the pitch command on a quadrotor sketch showing faster ω_2 and slower ω_1, while ω_3 and ω_4 remain at ω_{Hover}, this combination will make the quadrotor pitch forward $\uparrow$.

Yaw (u_4)

For $\oplus$ and $\otimes$ this command is created by increasing (or decreasing) the counter-clockwise propellers speed and by decreasing (or increasing) the clockwise ones. The yaw movement is generated thanks to the fact that the left-right propellers rotate CCW while the front-rear ones rotate CW, therefore when the overall torque is unbalanced, the vehicle turns on itself around z. Figure 4.5 shows the yaw command on a quadrotor sketch showing faster (ω_3, ω_4) and slower $\omega_1, \omega_2)$, this combination will make the quadrotor rotate clockwise. This command can produce changes on the vehicles current height.

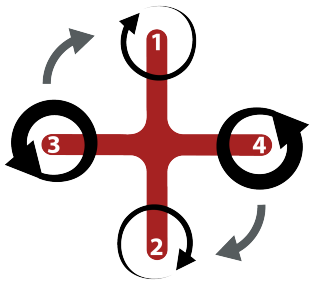


Fig. 4.5.: $\oplus$ Plus configuration Yaw command diagram.

4.2 Modelling

The coordinate systems and free body diagram for the quadrotor are shown in Fig.4.1. The world frame W is defined by axes x_W, y_W and z_W with z_W pointing upward. The body frame, B, is attached to the center of mass of the quadrotor with x_B coinciding with the preferred forward direction and z_B perpendicular to the plane of the rotors pointing vertically up during perfect hover. The $Z-X-Y$ Euler angles are used to model the rotation of the quadrotor in the world frame. To get from W to B, we first rotate about z_W by the yaw angle ψ, then rotate about the intermediate x-axis by the roll angle ϕ, and finally rotate about the y_B axis by the pitch angle θ. The rotation matrix for transforming coordinates from B to W is given by

$$R = \begin{bmatrix} c_\psi c_\theta - s_\phi s_\psi s_\theta & -c_\phi s_\psi & c_\psi s_\theta + c_\theta s_\phi s_\psi \\ c_\theta s_\psi + c_\psi s_\theta s_\phi & c_\phi c_\psi & s_\psi s_\theta - c_\psi c_\theta s_\phi \\ -c_\phi s_\theta & s_\phi & c_\phi c_\theta \end{bmatrix} \tag{4.1}$$

where $c_k = \cos(k)$, $s_k = \sin(k)$. The position vector of the center of mass in the world frame is denoted by r. The forces on the system are gravity, in the $-z_W$ direction and the forces from each of the rotors F_i, in the z_B direction. The equations governing the acceleration of the center of mass are

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \Sigma F_i \end{bmatrix}. \tag{4.2}$$

The components of angular velocity of the robot in the body frame are p, q, and r. These values are related to the derivatives of the roll, pitch, and yaw angles according to

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c_\theta & 0 & -c_\phi s_\theta \\ 0 & 1 & s_\theta \\ s_\theta & 0 & c_\phi c_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{4.3}$$

In addition to forces, each rotor produces a moment perpendicular to the plane of rotation of the blade M_i. Rotors 1 and 3 rotate in the z_B direction while 2 and 4 rotate in the z_B direction. Since the moment produced on the quadrotor is opposite to the direction of rotation of the blades, M1 and M3 act in the z_B direction while M2 and M4 act in the z_B direction. We let L be the distance from the axis of rotation of the rotors to the center of the quadrotor. The moment of inertia matrix referenced to the center of mass along the x_B y_B z_B axes, I, is found by weighing individual components of the quadrotor. The angular acceleration determined by the Euler equations is

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{4.4}$$

4.3 Control

In this section the control strategies and algorithms are presented. Also experimental results of flight tests are shown in the last subsection. This section is related to the previous one (Sec.4.2), because it analyses the quadrotor model and tries to *invert* it to reach a certain position and attitude. A variety of approaches to quadrotor control are found in the literature (Sec.1.2.2), either linear and non-linear methods. The simplicity of the quadrotor system and its stability and agility under closed-loop control have led it to be a popular platform for non-linear techniques such as dynamic inversion which is used in this research alongside a typical feedback loop. The PID algorithm uses a control loop feedback mechanism to minimize the error between the desired output signal and the real output signal. In the industrial area the most used linear regulators are PID (Schiavoni et al., 2015). The reasons of this success are primarily:

- simple structure
- good performance for several processes
- tunable even without a specific model of the controlled system

Particularly in robotics, and very often in aerial robotics, PID techniques represents the basics of control. Even though a lot of different algorithms provide better performance than PID, this last structure is often chosen for the reasons expressed above. A standard PID structure is composed of three contributes:

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{4.5}$$

Where u is the variable being controlled, e the error between the desired value and the output y. K_P, K_I, K_D being the proportional, integral and derivative coefficient. The contributions can be defined as:

- P: This contribution is proportional to the error on the inside while outside the output will be minimum or maximum
- I: This contribution varies according to the integral of the error, it increases the overshoot and the settling time and eliminates the steady state error
- D: This contribution varies according to the derivative of the error, it helps to decrease the overshoot and the settling time

As explained on Chapter 3, the entire control strategy is performed on-board the vehicle in the tuple of computers called Flight Stack (Vargas et al., 2014). The inner attitude loop control is performed by the COTS flight controller, using accelerometers, gyroscopes and it runs approximately at $286Hz$ on the MultiWii board (MultiWii, 2010) and $400Hz$ on the Pixhawk flight controller (Meier et al., 2011). While the outer position loop control is calculated by the *companion computer*, using the vehicle position reference coming from the MoCap system (Sec.3.2.1), this loop runs at $100Hz$. A block diagram of the nested feedback loops can be seen at figure 4.6. The dynamics of the quadrotor were described in previous

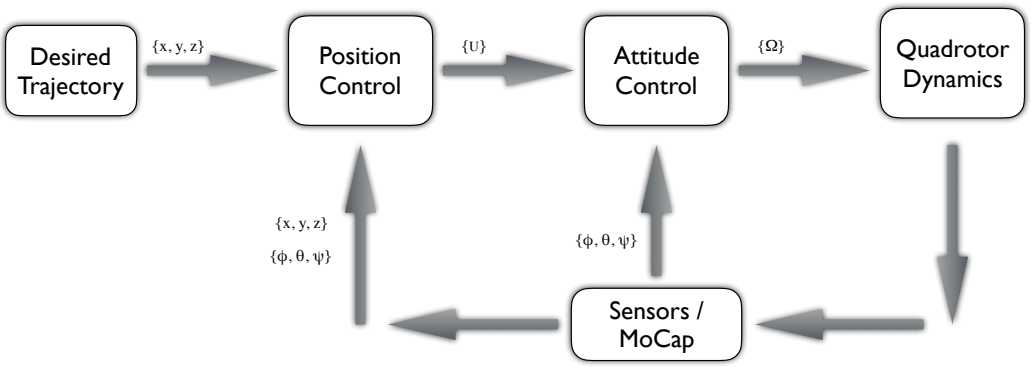


Fig. 4.6.: Strategy control block diagram.

sections, the next equation is composed of linear equations in the World inertial frame and angular equations in the body frame:

$$\begin{cases} \ddot{x} = (s_\psi s_\phi + c_\psi s_\theta c_\phi)\frac{u_1}{m} \\ \ddot{y} = (-c_\psi s_\phi + s_\psi s_\theta c_\phi)\frac{u_1}{m} \\ \ddot{z} = -g + (c_\theta c_\phi)\frac{u_1}{m} \\ \dot{p} = \dfrac{I_{yy} - I_{zz}}{I_{xx}} qr - \dfrac{J_{TP}}{I_{xx}} q\Omega + \dfrac{u_2}{I_{xx}} \\ \dot{q} = \dfrac{I_{zz} - I_{xx}}{I_{yy}} pr - \dfrac{J_{TP}}{I_{yy}} p\Omega + \dfrac{u_3}{I_{yy}} \\ \dot{r} = \dfrac{I_{xx} - I_{yy}}{I_{zz}} pq + \dfrac{u_4}{I_{zz}} \end{cases} \tag{4.6}$$

Such system shows how the quadrotor accelerates according to the basic movement commands given. This command movements (*pseudo-controls* Eq.4.7) are going to be *fed* into the inner loop controller, which is the COTS flight controller.

$$\begin{cases} u_1 \longrightarrow Throttle \\ u_2 \longrightarrow Roll \\ u_3 \longrightarrow Pitch \\ u_4 \longrightarrow Yaw \end{cases} \tag{4.7}$$

The objective of the quadrotor stabilization is to find which values of motor speeds (PWM) will maintain the vehicle in a certain orientation (ϕ, θ, ψ) required in the desired trajectory task. This process is known as dynamic inversion. The dynamics of the quadrotor must be simplified in order to provide a model that can be easily implemented in the control algorithms. Dynamic inversion is a popular control strategy for the quadrotor, as demonstrated by its use in Voos, 2009, Das et al., 2008 and Mistler et al., 2001. Glad et al., 2000 describe the theory of dynamic inversion, also known as as input-output linearisation, and apply it to a general SISO system. Dynamic inversion is applicable to systems which may be described in control-affine form, such as the quadrotor models. The quadrotor is an under-actuated system, with four inputs and six degrees of freedom. This impacts the ability to feedback linearise the system and requires the use of a nested-loop structure in the controller. To invert the quadrotor system, it is necessary to define two separate outputs. Das et al., 2008 describe a tracking output $y_t = h_t(x) = [x, y, z, \psi]^T$ and a flat output $y_f = h_f x) = [z, \phi, \theta, \psi]^T$, the flat outputs may be related to the pseudo-controls (Eq. 4.7) by a series of SISO systems, then it is possible to invert each system such that linear relationships are obtained between the flat output and new pseudo-controls.

4.3.1 Attitude controller

If we consider attitude movements that are close to the nominal hover state where the roll and pitch angles are small and from 4.9 we assume that the products of inertia are small (ideally, they are zero because the axes are close to the principal axes) and $I_{xx} \approx I_{yy}$ because of the symmetry then:

$$I_{xx}\dot{p} = u_2 - qr(I_{zz} - I_{yy}) \tag{4.8a}$$

$$I_{yy}\dot{q} = u_3 - pr(I_{xx} - I_{zz}) \tag{4.8b}$$

$$I_{zz}\dot{r} = u_4 \tag{4.8c}$$

We can also assume the component of the angular velocity in the z_B direction r, is small so the rightmost terms in (4.8a) and (4.8b) which are products involving r are small compared to the other terms. We note that near the nominal hover state $\dot{\phi} \approx p$, $\dot{\theta} \approx q$ and $\dot{\psi} \approx r$. The vector of desired rotor speeds can be found from the desired net force (u_1, des) and moments (u_2, des, u_3, des and u_4, des) by inverting

$$u_{des} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega^2_{1,des} \\ \omega^2_{2,des} \\ \omega^2_{3,des} \\ \omega^2_{4,des} \end{bmatrix}. \tag{4.9}$$

As previously stated, the open-loop dynamics of the quadrotor are highly unstable (Miller, 2011). A feedback control topology is needed due to the fact (Stevens et al., 2003) that the model reveals the poles are located on the right of the real-imaginary plane and its damping ratio is negative. One of the most commonly used technique (Nelson, 1998) in order to provide a solution is a Stability Augmentation System (SAS), which makes the vehicle (aircraft) stable via the rate measurement in the feedback loop. The SAS will actuate the vehicle pseudo-controls to dampen out the vehicle buffeting regardless of the attitude, this technique is heavily used to stabilize the heading of the quadrotor (u_4), or commonly known as the yaw rate controller. As stated in Section 2.7, two COST flight controllers are used in this research (MultiWii and Pixhawk), their particular attitude controllers are used in this research. The vehicle attitude estimation was previously analysed in 2.7.2, and it is used in order to control the stabilisation of the multirotor. The MultiWii project implements a controller based on an inner-outer-loop structure, and its diagram is showed on figure 4.7. The Pixhawk flight controller implements a single-feedback loop as showed in figure 4.8, and inside the controller the working mechanism follows 4.10, where adjusting K_P and K_D will result in desirable close-loop behaviour. One disadvantage of following this structure is

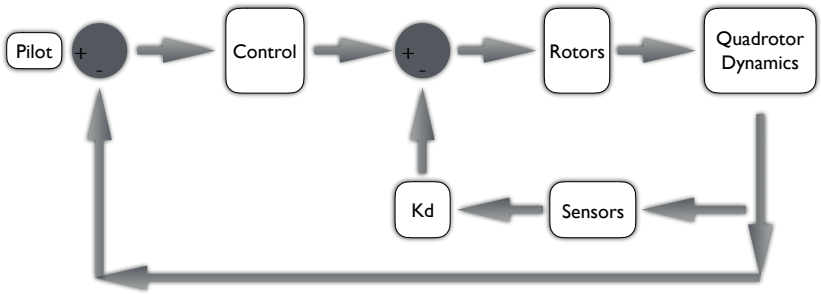


Fig. 4.7.: MultiWii Attitude control block diagram.

that there will be a close-loop *zero* at $s = -\frac{K_D}{K_P}$ and will cause a large overshoot if the plant poles are not dampened. This is one if the main reasons why the MultiWii project uses the inner-outer-loop structure that uses a rate feedback instead of a close-loop pole.

$$\begin{aligned} G_c(s) &= K_P + K_D s \\ \frac{Y}{P} &= \frac{(K_P + K_D s)G_P}{1 + (K_P + K_D s)G_P} \end{aligned} \tag{4.10}$$

The PID structure of the Pixhawk is a standard one as stated before a rate measurement

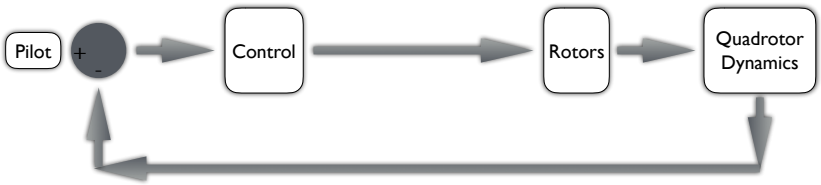


Fig. 4.8.: Pixhawk Attitude control block diagram.

is not used on this controller, the derivative of the error signal is used to provide control inputs to the plant. The MultiWii PID structure on the other hand, removes the derivative term in the forward path.

4.3.2 Position controller

Here we present a position control strategy that uses the pseudo-controls throttle, roll, pitch, yaw, in order for station-keeping or maintaining the position at a desired x, y, and z location. There is similar approaches in Mellinger, 2012 and Khalil, 2002. This approach will be later used by the trajectory controller as well. All of the next controllers and algorithms are implemented using the Python computer language on the framework *DronePilot*. This position controller can also be called *Hover Controller*, and specially on the *DronePilot* framework, the examples refer to it as hover controller. As the quadrotor attitude is stabilised by the on-board flight controller, we are concerned only with defining the attitude commands which drive the quadrotor along a specific position trajectory. We may therefore

consider the quadrotor vehicle and attitude stabilisation controller as a single closed-loop system, with outputs $r = [x, y, z]^T$ and inputs $u = [T_d, \phi_d, \theta_d, \psi_d]$. Here, z describes the height of the quadrotor above the ground, x and y are parallel to the ground and the walls of the flight space. The origin of the inertial frame is located at ground level in the centre of the flight space. The inputs to the stabilised quadrotor vehicle are the desired thrust T_d and desired roll, pitch and yaw angles, ϕ_d, θ_d and ψ_d, respectively. Neglecting the attitude dynamics, which are stabilised by the on-board flight controller, we may describe the position response of the quadrotor by

$$\begin{aligned} \ddot{x} &= \frac{T}{m}(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi) \\ \ddot{y} &= \frac{T}{m}(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi) \\ \ddot{z} &= \frac{T}{m}\cos\phi \cos\theta - g \end{aligned} \tag{4.11}$$

where T is the net thrust from the rotors, ϕ, θ and ψ are the roll, pitch and yaw angles, respectively, m is the quadrotor mass and g is the acceleration due to gravity. We make the assumption that the stabilised quadrotor attitude dynamics are relatively fast with respect to the desired position response. Hence, we assume that

$$T \approx T_d, \qquad \phi \approx \phi^{des}, \qquad \theta \approx \theta^{des} \tag{4.12}$$

giving us a direct relationship between output r and input u. We may then employ dynamic inversion (or feedback linearisation) to invert Equation 4.11 and provide mappings between the inputs u and desired trajectory r^{des}. Using this method, we may determine the control gains via simple pole placement of the linearised closed-loop system, as in Ireland, 2014. We let $r_T(t)$ be the desired trajectory (or location) the vehicle is going to attempt to track. The command accelerations, $\ddot{r}_i^{des}$ are calculated from the PID feedback of the position error ($e_i = (r_{i,T} - r_i)$) by

$$(\ddot{r}_{i,T} - \ddot{r}_{i,T}^{des}) + k_{p,i}(r_{i,T} - r_i) + k_{i,i}\int (r_{i,T} - r_i) + k_{d,i}(\dot{r}_{i,T} - \dot{r}_i) = 0 \tag{4.13}$$

where $\dot{r_{i,T}} = \ddot{r_{i,T}} = 0$ for hovering conditions.

The gains k_p, k_i and k_d are calculated from pole placement and are designed to provide a stable, critically damped response. Having calculated the desired acceleration of the vehicle in equation 4.13, we must then obtain the system inputs u. We do so by inverting the dynamic model in Equation 4.11 to obtain the inputs as functions of the position r.

X-Y Controller

Equations 4.11 and 4.13 are mixed and linearised accordantly in order to obtain the relationship between the desired accelerations with the roll and pitch pseudo-controls.

$$\begin{aligned} \ddot{r}_x^{des} &= g(\phi^{des}\cos(\psi) + \phi^{des}\sin(\psi)) \\ \ddot{r}_y^{des} &= g(\theta^{des}\cos(\psi) - \theta^{des}\sin(\psi)) \end{aligned} \tag{4.14}$$

The relationships above (Eq. 4.14) are then inverted to be able to calculate the desired pseudo-controls commands that are going to be sent to the flight controller.

$$\begin{aligned} u_{2,des} = \theta^{des} &= \frac{1}{g}(\ddot{r}_y^{des}\cos(\psi) + \ddot{r}_x^{des}\sin(\psi)) \\ u_{3,des} = \phi^{des} &= \frac{1}{g}(\ddot{r}_x^{des}\cos(\psi) - \ddot{r}_y^{des}\sin(\psi)) \end{aligned} \tag{4.15}$$

As the flight controller receives attitude commands as PWM values in the range $[1000\ 2000]$, corresponding to the attitude range $[a_{\min}\ a_{\max}]$. We thus scale and offset the attitude commands to provide the corresponding PWM values as follows

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \frac{1000}{a_{\max} - a_{\min}} \begin{bmatrix} \phi_d \\ \theta_d \end{bmatrix} + 1500 \tag{4.16}$$

where the PWM value for zero roll or pitch is 1500. u_1 and u_2 are then the PWM commands corresponding to roll and pitch, respectively. As stated before, this control loop runs at $100Hz$, which is the fastest rate data is received from the MoCap system, while the inner loop control runs at $286Hz$, there is usually a trade-off in optimising the control gains between speed of response and stability.

Height Controller

The height control is very similar to the X-Y control, where equations 4.11 and 4.13 are mixed and linearised accordantly in order to obtain the relationship between the desired acceleration and the throttle pseudo-control.

$$\ddot{r}_z^{des} = \frac{u_{1,des}}{m} \tag{4.17}$$

And then inverted as follows

$$u_{1,des} = m\ddot{r}_z^{des} \tag{4.18}$$

After testing on the real vehicle, a compensation to equation 4.18 is added to improve the behaviour of the controller, this upgrade compensates the throttle pseudo-control u_1 for close-to-large changes in the orientation of the vehicle by using the cosine of the current roll and pitch angles of the vehicle, a gain is added which relates the minimum throttle PWM value u_0 and the throttle PWM value for the vehicle at hover u_{hover}. The final equation to compute the throttle pseudo-control is

$$u_{1,des} = K_t \frac{(\ddot{r}_z^{des} + g)m}{\cos\phi_i \cos\theta_i} + u_0 \tag{4.19}$$

Where K_t is a throttle scale factor that is computed as

$$K_t = \frac{u_0 - u_{hover}}{mg} \tag{4.20}$$

Heading Controller

A simple heading controller is employed. It is important to consider that in order to create the *hover controller*, a heading controller is not necessarily needed. At hover conditions $\psi(t) = \psi_0$, and more over, the X-Y controller compensates for changes due to different headings, as seen in Equation 4.21. This controller is utilised to keep the vehicle *pointed* in a specific direction, in order to visualize the pitch and roll movements in a better way. The heading PWM command u_4 determines the heading speed, thus a simple proportional controller is sufficient,

$$u_{4,des} = k_{p,\psi}(\psi^{des} - \psi) \tag{4.21}$$

Again, the command is scaled to provide the corresponding PWM value. The PWM commands are transmitted in the range $[1000\ 2000]$, corresponding to the heading rate range $[\dot{\psi}_{min}\ \dot{\psi}_{max}]$. The heading PWM command is thus

$$u_4 = \frac{1000}{\dot{\psi}_{max} - \dot{\psi}_{min}} \dot{\psi}^{des} + 1500 \tag{4.22}$$

4.3.3 Trajectory Generation

The trajectory generation is a very simple yet effective method, that involves using the position controller exposed before and simply changing the pseudo-controls for X, Y, Z and heading, depending on the current time-step. Two traditional trajectories (Fig. 4.9) were chosen to test the position controller performance, one is the circular or circle trajectory

and the other one is the figure-of-eight trajectory. Both trajectories use the full motions of the vehicle and can give us an idea of the performance of the controllers and how the entire system attempts to track them. The circle trajectory is generated using

$$\begin{aligned} x_{des} &= r\cos(t) \\ y_{des} &= r\sin(t) \end{aligned} \tag{4.23}$$

Being t a constant that relates the time it will take to complete a circle by multiplying with the step time (update rate, i.e. $0.01s$), if we want a circular trajectory that will be completed in n seconds, then $t = \frac{2\pi}{n}$. The figure-of-eight trajectory used was the *Lemniscate of Bernoulli*, which is a plane curve defined from two points a and b, as showed on parametric equation 4.24. This lemniscate (that resembles a ∞ symbol) was first described in 1694 by Jakob Bernoulli as a modification of an ellipse.

$$\begin{aligned} x_{des} &= \frac{a\sqrt{2}\cos(t)}{\sin(t)^2 + 1} \\ y_{des} &= \frac{b\sqrt{2}\cos(t)\sin(t)}{\sin(t)^2 + 1} \end{aligned} \tag{4.24}$$

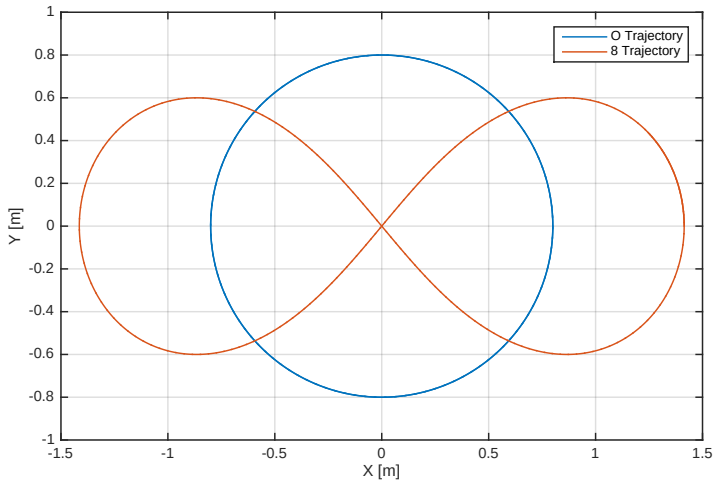


Fig. 4.9.: Circle and figure-of-eight trajectory.

4.4 Experimental results

Thanks to the DronePilot (3.5) framework, it was possible to record data during flight experiments. This data help to tune the parameters of the controllers with a quantitative feedback as well as being able to use this data for Machine Learning experiments, this will be explained in later chapters. In the next figures, the flight tests for attitude and position

controllers will be presented. The plots are divided in two sub-plots to help identifying the trends in the signals and the error boundaries.

4.4.1 Attitude controller performance

The attitude controller plots showed in this section were obtained using DronePilot on-board a Flight Stack compromised by an Odroid U3 as companion computer and a Naze32 (MultiWii) as COTS flight controller, the data-logging thread records data at $100Hz$. The MoCap system update the pose data at $100Hz$, the communication with the flight controller happens at the same rate of the outer-loop with a maximum rate of up to $300Hz$, this is to ensure there is no bottle-neck communication problems in the recording and control thread. The attitude measurement will be compared between the flight controller IMU (Inertial Measurement Unit) and the motion capture system, which is often considered as the true value. It is important to consider that the motion capture suffers from a considerable flaw

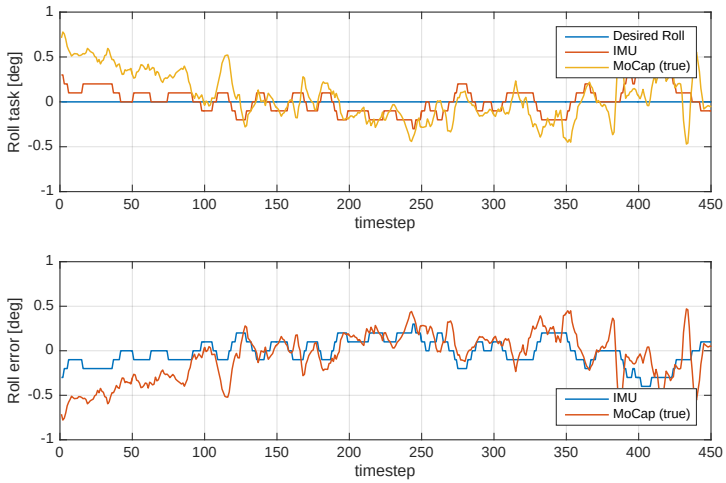


Fig. 4.10.: Roll stabilization performance.

if the system is no calibrated properly and regularly. For the flight sessions it was always intended to have the system properly calibrated. If during the flight tests we observed an abnormal behaviour of the vehicle, like inconsistent disturbances, those indicated that the MoCap system needed to be re-calibrated again, therefore making the flight tests time-consuming. Figure 4.10 displays a flight test for the roll stabilization performance. The test was done by flying the vehicle inside the MAST Lab, making the vehicle track a zero degree pitch and roll command while recording orientation data from the IMU inside the flight controller and the MoCap system. The position of the vehicle will drift due to mechanical and aerodynamic factors (ground effect and wall-vehicle fluid interaction), therefore the test cannot last for long time to safely obtain data from the MoCap system, due to the physical size of the laboratory flight area ($3x2$ meters). It is appreciated on the plot that

the roll error is always lower than one degree. For the pitch part, figure 4.11, a small

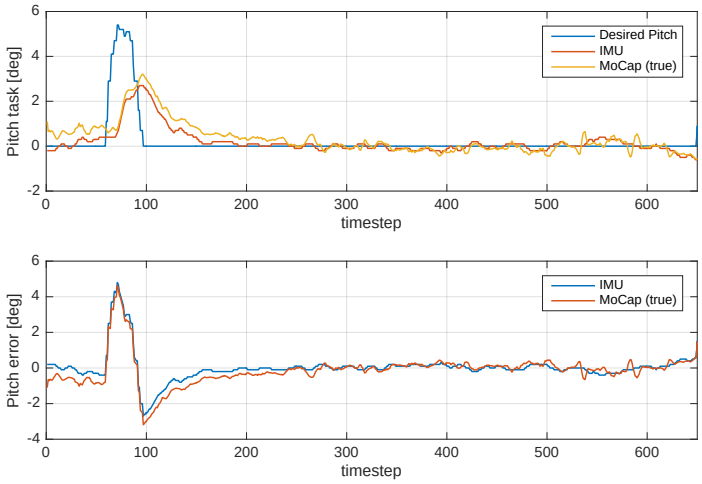


Fig. 4.11.: Pitch stabilization performance.

$+5$ degree pitch input from a human pilot is enforced to the vehicle, followed by a zero degree command. The reaction lag and settling time can be observed. The timestep for the experiments is 0.01 seconds showing around 7 seconds of flight. As mentioned before, in order to keep a stable flight, both roll and pitch errors must be kept low. In hovering conditions if one of the two angles (pitch and roll) are different than zero a longitudinal acceleration will occur and will make difficult to maintain a fixed position without drift, this is the reason why the plots show a small amount of data.

4.4.2 Position controller performance

For the position control the flight test was very similar to the one of attitude; same vehicle, flight stack components and software. The difference is that in this flight test the vehicle was being totally controlled by DronePilot. The control thread will compute the appropriate roll, pitch, yaw and throttle commands in order to keep the vehicle in a specified position. The chosen location inside the MAST Lab was $[1, 1, 1]$ being $[X, Y, Z]$ and meters as units.

X-Y

The initial location of the vehicle was approximately $[0, 0, 0]$. Then DronePilot executes a step-response mission and will fly the vehicle from the initial location to the final location $[1, 1, 1]$ and then it will attempt to hover in that position until the battery runs low or it is asked to land the vehicle. A step-response is important because it informs to the control

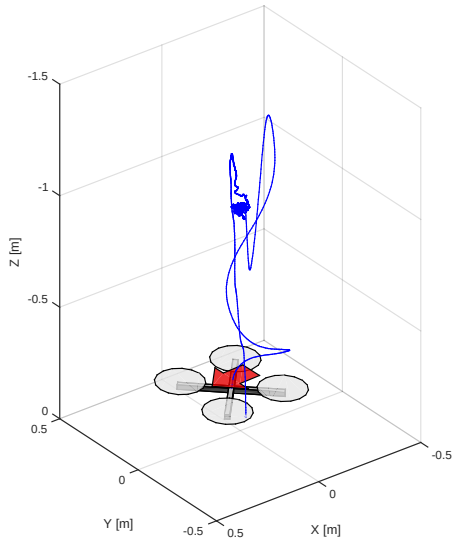


Fig. 4.12.: 3D plot of an entire position hold flight test.

designer how the system responds to a sudden input as well as giving information on the stability of dynamical system (vehicle) and its ability to reach one stationary state when starting from another. Figure 4.13 shows the X-axis step response of the X-Y controller. In such plot we can observe that the reaction time when changing position (within 3 body lengths) is approximately 4 seconds. This settling time is a trade-off between stability and robustness of the controllers with this specific vehicle platform. For the flight experiments, it was preferable to have a much more stable vehicle response than a fast response. The Y-axis step response is showed at figure 4.14, it is noticed that even if the gains were identical between axis, the response on this axis had a bigger overshoot than the previous axis, this might be caused due to the *wall-vehicle fluid interaction* due to the fact that the magnitude of the laboratory Y-axis is significantly lower, and the vehicle ends up being closer to the wall. Another test performed to the vehicle involved how the position controller behave over longer periods of time when attempting to hold a predefined position ($[1, 1, 1]$ meters), as seen in figure 4.15. The X-Y trajectory is plotted from above with a 20 centimetre tolerance circle to show that the control will maintain the commanded position with a maximum error of 0.6 body length.

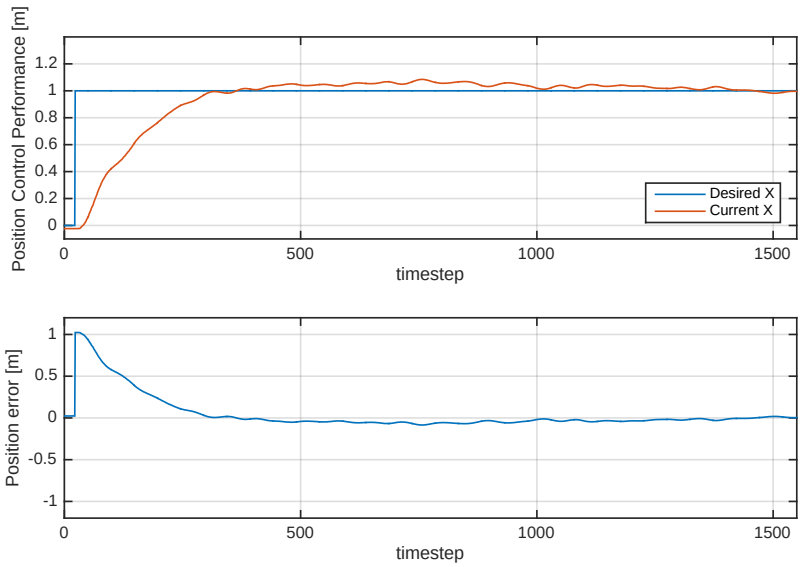


Fig. 4.13.: Position stabilization performance - X axis.

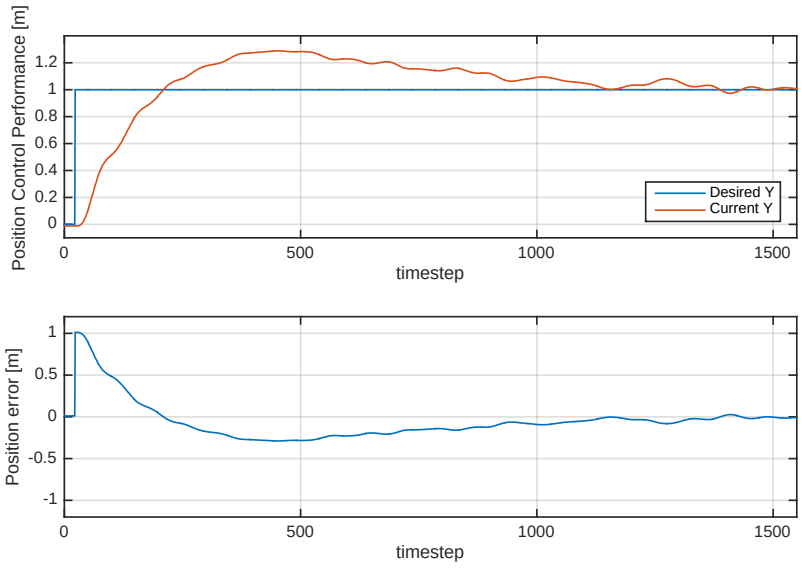


Fig. 4.14.: Position stabilization performance - Y axis.

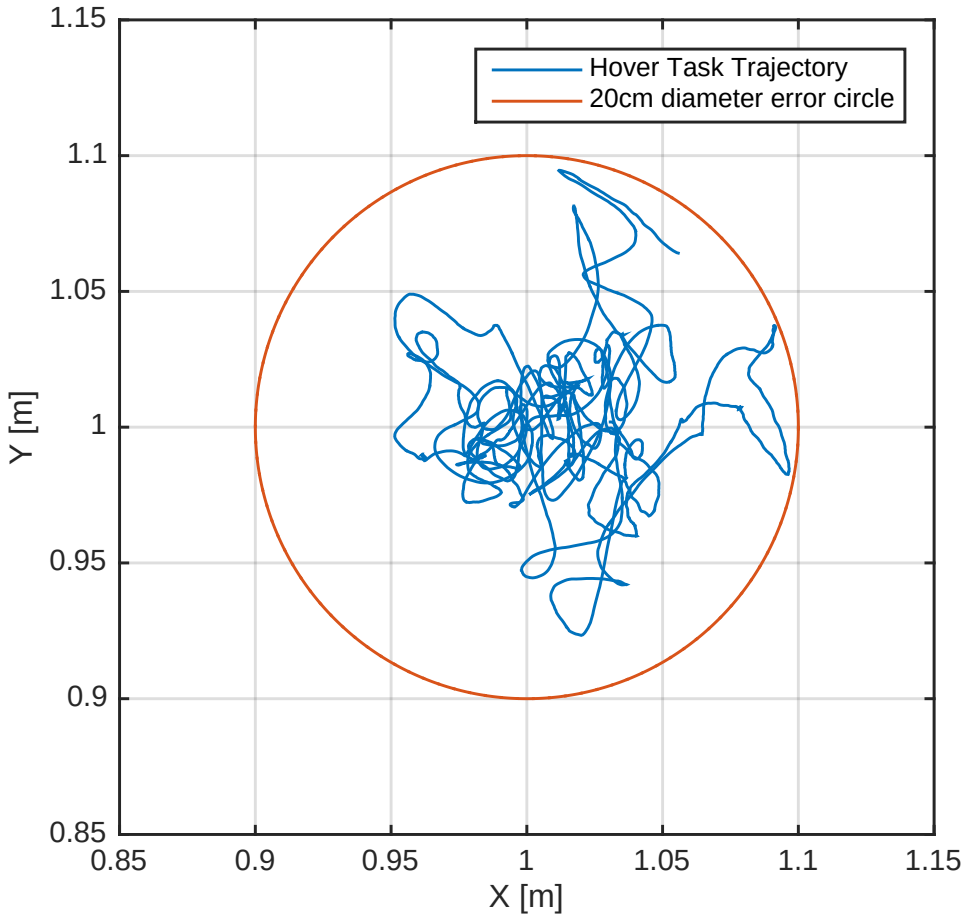


Fig. 4.15.: Position hold performance - Top view.

Height

The height controller step response is showed in figure 4.16. The gains for this controller are different to the X-Y, in the form that the vehicle can have a slightly faster settling time due to response of a matched rotor, this means that the components in the rotor tuple fit in a proper way with the vehicle frame and its weight. For example, if the propellers are changed in this vehicle, the response of this controller will be different to the one previously showed. Important to consider that for this controller, the gains need to be

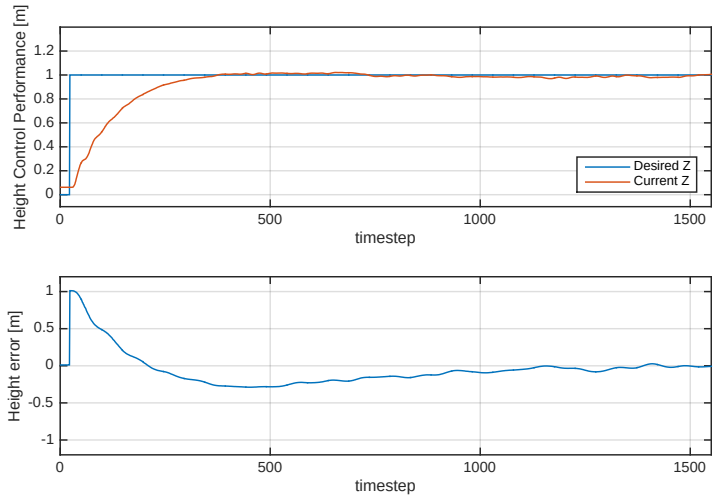


Fig. 4.16.: Position stabilization performance - Z axis.

carefully selected according to the vehicle rotor capabilities and parameters in order to obtain the best response possible of the system.

Heading

The step response of the most simplistic control is showed in figure 4.17. In this case, the vehicle was rotated so that it performs a full 180 degree turn. This rotation is maximum heading change possible. For this one, only the data coming from the MoCap is used, due to the fact that the magnetometers inside the IMU of the flight controller suffer of magnetic interference caused by the wireless communication device among other magnetic fields present inside the MAST Lab. In the plot (Fig. 4.17), three short *disturbances* from the heading of the vehicle can be observed, this is a tracking error coming from the Motion Capture system, it might be caused by damaged reflective markers or bad calibration of the MoCap system.

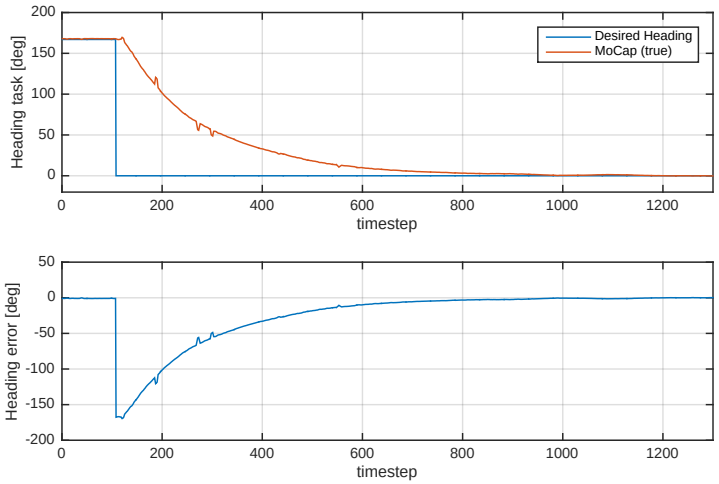


Fig. 4.17.: Position stabilization performance - Heading.

4.4.3 Trajectory flights

As explained before, two trajectories were chosen, the circle and the figure-of-eight. The latter has a motion which is far more aggressive than the circle, therefore more demanding for the position controller. Four different tests for each trajectory were performed, changing the time to complete a circuit in each one of them. As seen on the step response of the X-Y controller (Fig. 4.13) the vehicle reached the desired position in approximately less than 4 seconds, therefore in order to avoid accidents, the minimum time to complete a circuit was 4 seconds, because it could instabilities to the vehicle going faster that the step response.

Circle

For the circular trajectory, the first test was set-up in such way that the vehicle could complete a circumference of radius 0.8 meters in 10 seconds. Figure 4.18 shows the experimental response of the desired trajectory and the response of the vehicle on X-Y, the height was kept constant. It can be appreciated that the tracking error is less than 20 centimetres, which is very similar to the hover performance test. Such figure shows 2000 timestep, which is exactly 20 seconds, the update rate in these tests was $100Hz$. Two circuits are displayed.

Figure 4.19 shows the same test from a top view perspective, the vehicle tracks the circle trajectory in a very successful and precise manner. The 3D view can be appreciated in Fig. 4.20. In this test the height, heading and X-Y controllers were active, the latter one working more actively in order to keep the vehicle in the desired position.

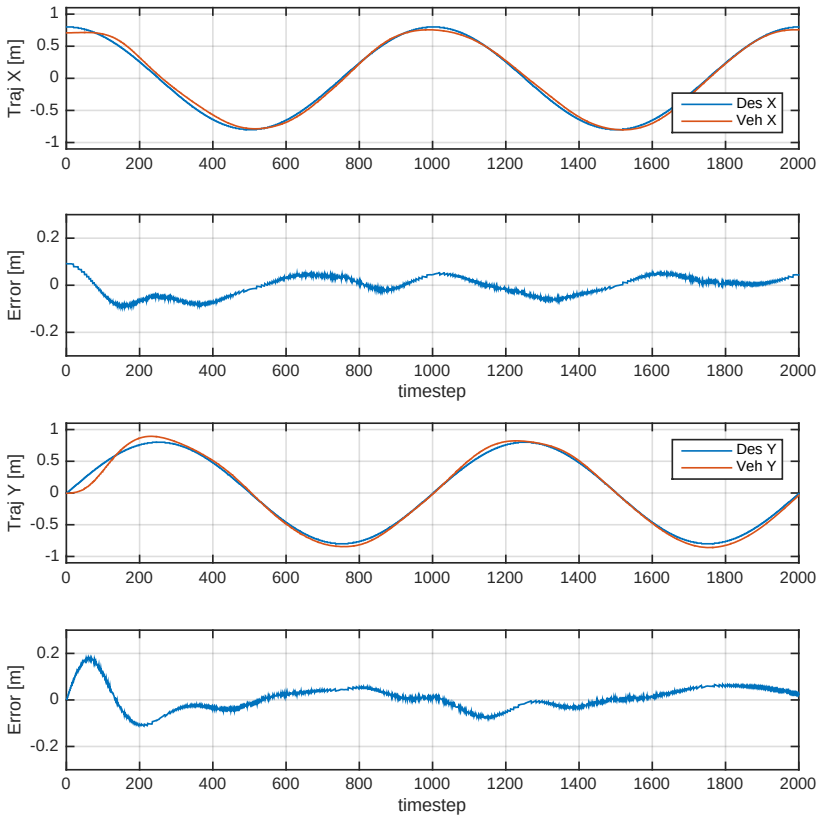


Fig. 4.18.: Circular trajectory tracking performance.

The next test presented, involved changing the time for the vehicle to complete the same circuit. The chosen times were $10, 8, 6$ and 4 seconds. It can be observed in Fig. 4.21 how the performance of the position controller started to decrease when the time is reduced as the time to complete the circuit gets closer to the step response of the vehicle. Important to notice that the tests were done without any tuning of the parameters on the controllers. MSE (Mean Squared Error) is calculated to observe how the tracking error increases when the trajectory is more demanding (faster) to the vehicle.

Figure 4.22 and 4.23 shows a time-lapse photography merged into a single image (time-collapse) in order to show the trajectory progression of the vehicle when performing the autonomous circular flights, it can be appreciated how when the time to complete the trajectory is lower the change in the attitude of the vehicle becomes more aggressive and also *fewer vehicles* appear on the image.

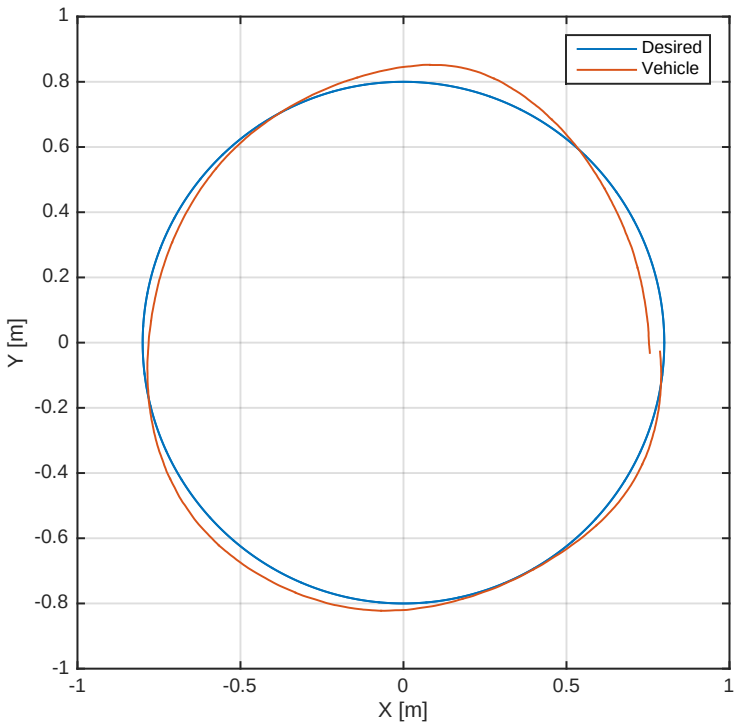


Fig. 4.19.: Circular trajectory tracking performance - Top view.

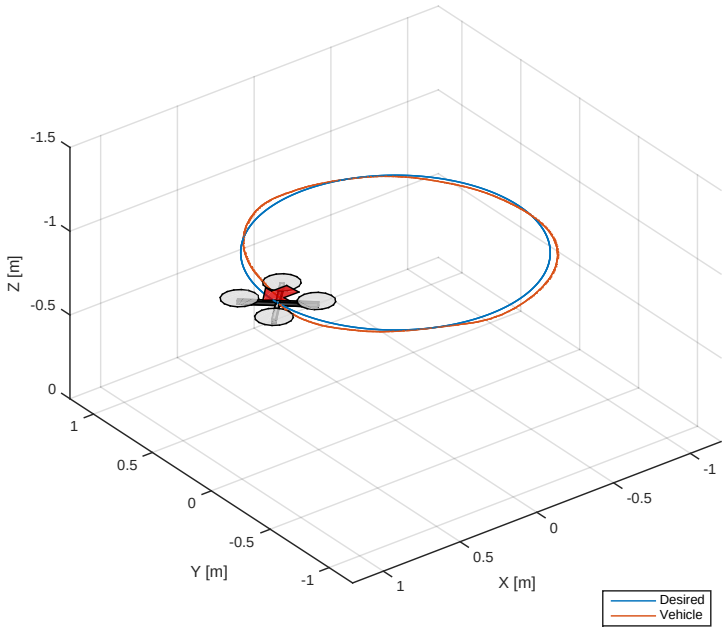


Fig. 4.20.: Circular trajectory tracking performance - 3D view.

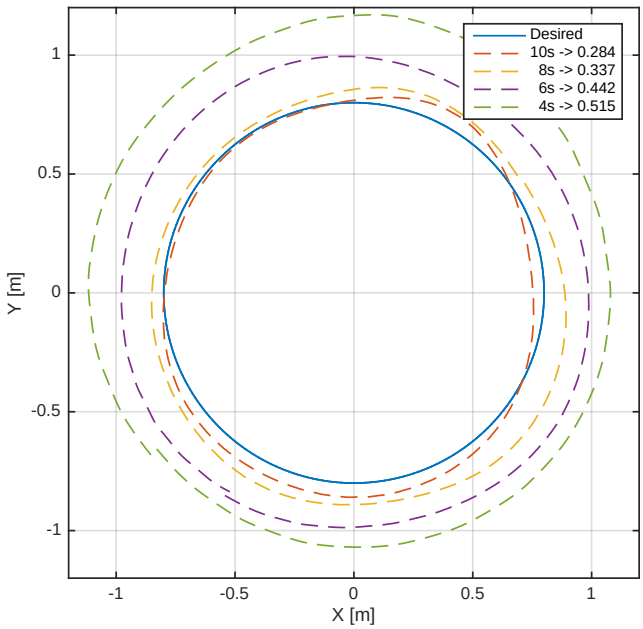


Fig. 4.21.: Circular trajectory tracking performance using different times to complete the circuit.

Fig. 4.22.: Time-collapse photography of a circle trajectory performed in 10 seconds.

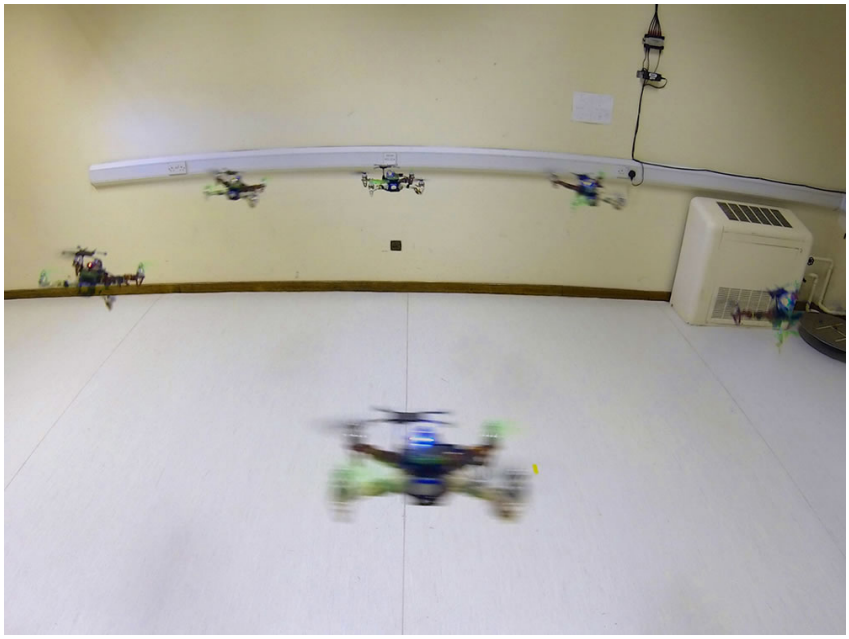


Fig. 4.23.: Time-collapse photography of a circle trajectory performed in 4 seconds.

Figure-of-eight

In the similar manner as with the circular trajectory, the first test with the lemniscate (*a pendant ribbon*) was configured to be completed in approximately 10 seconds. This trajectory is longer to the circular one, therefore the vehicle must travel faster to complete the circuit in comparison that with the circular. The average velocity of the vehicle performing the circular trajectory was $0.5m/s$ while on the lemniscate the average is about $0.59m/s$. Fig. 4.24 shows the performance of the controller and vehicle when attempting to track

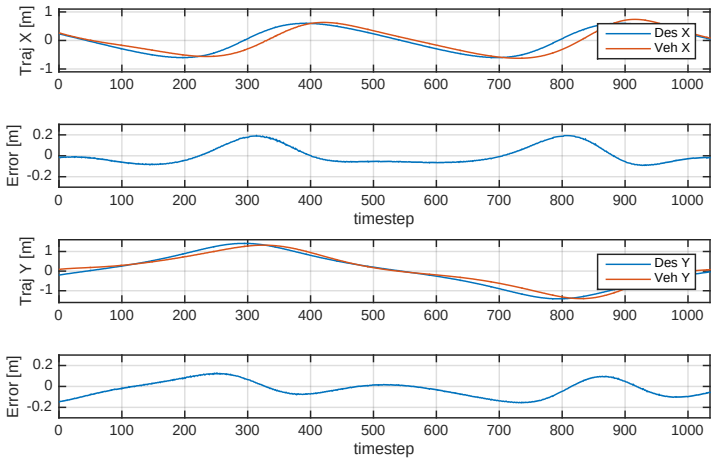


Fig. 4.24.: Figure-of-eight trajectory tracking performance.

a figure-of-eight trajectory in 10 seconds. Regarding the error it can be appreciated that it never surpass 0.2 but the trajectory is more demanding than the one of circular one, therefore the lemniscate contains more error. The top view can be seen in Fig. 4.25 while the 3D

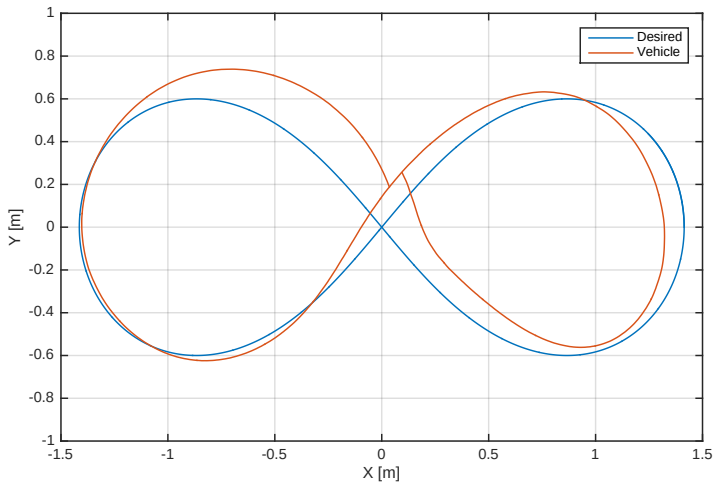


Fig. 4.25.: Figure-of-eight trajectory tracking performance - Top view.

plot is in Fig. 4.26. The top view shows a bigger error when the trajectory finishes, as it is extremely complicated that the vehicle can pass in the same position where it started. The height controller suffer a small performance decrease in order to achieve the desired height, due to the attitude motions required to achieved the trajectory, and as is showed in equation 4.19, the attitude of the vehicle is linked directly to the height controller. In figure 4.27 the different tracked trajectories for several times ($10, 8, 6$ and 4) are showed alongside their correspondent MSE, there is a dramatically increase of error when the time became smaller. The highest error appears when the trajectory must be completed in 4 seconds, important to remember that the step response showed us a settling time very close to 4 seconds, therefore we can state that in the figure-of-eight trajectory at 4 seconds the vehicle is reaching its performance limits, if a faster trajectory is attempted, the vehicle will not be able to track the trajectory as a new position command is due when a previous one is not reached yet. The three controllers actions during tracking of a figure-of-eight trajectory can be seen on Fig. 4.28, the controllers acting on the plot are two similar angle control for position X-Y which uses the pseudo-controls *roll* and *pitch*, heading controller using the pseudo-control rate *yaw* and a height controller using the pseudo-control *throttle*. All pseudo-control units are fed to the flight controller in PWM. It is shown that at 220 and 480 time-step the height controller had a *perturbation* due to a momentarily loss of the *trackable* position from the motion capture system.

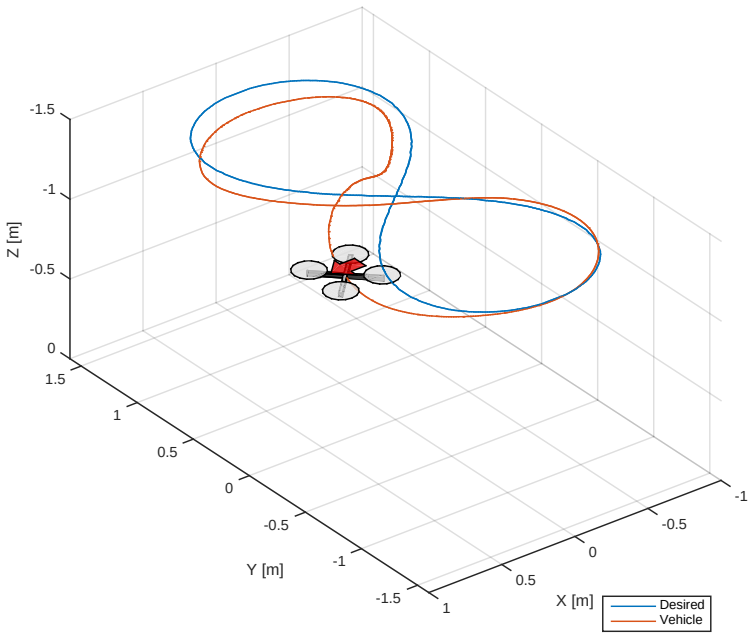


Fig. 4.26.: Figure-of-eight trajectory tracking performance - 3D view.

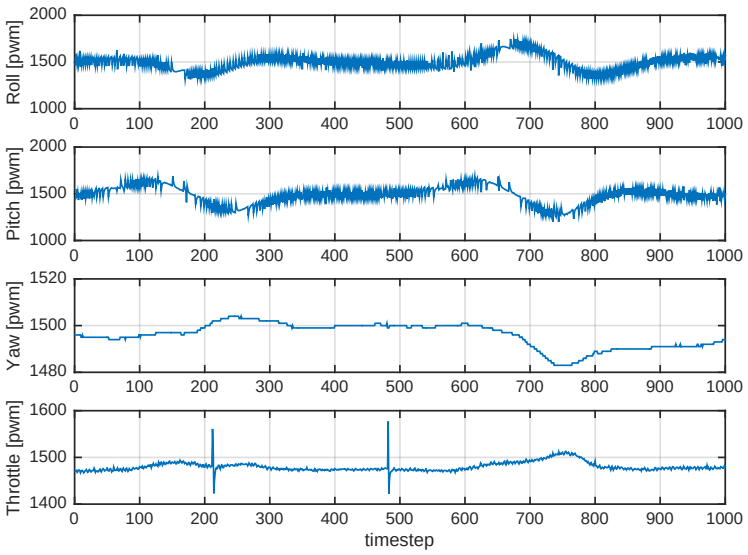


Fig. 4.28.: Controller action during tracking.

Figure 4.29 and 4.30 shows the time-collapse in order to appreciate the physical movements movements of the vehicle when tracking the trajectory, also the much more aggressive

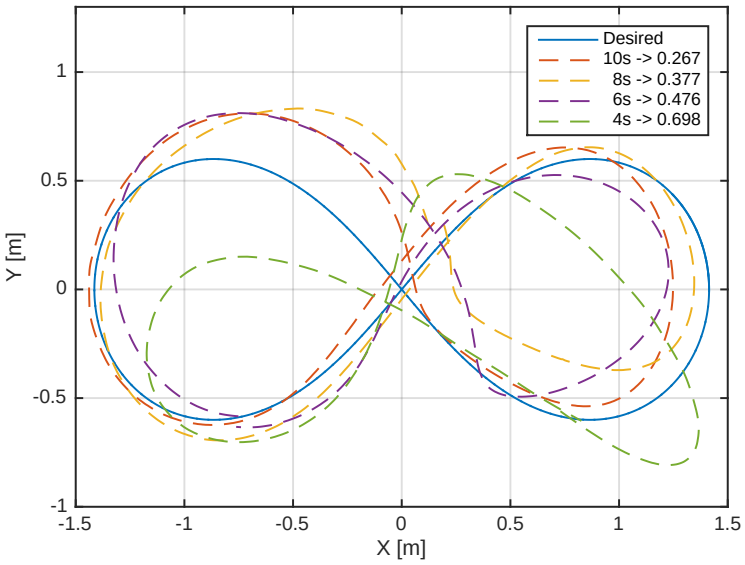


Fig. 4.27.: Circular trajectory tracking performance using different times to complete the circuit.

change in attitude from the 4 seconds (Fig. 4.30) trajectory against the 10 second (Fig. 4.29) trajectory.

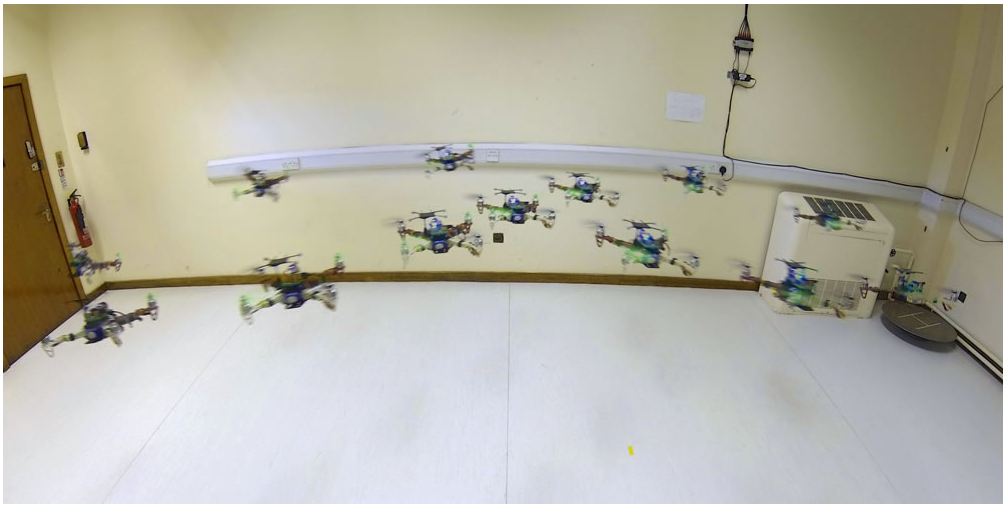


Fig. 4.29.: Time-collapse photography of a figure-of-eight trajectory performed in 10 seconds.

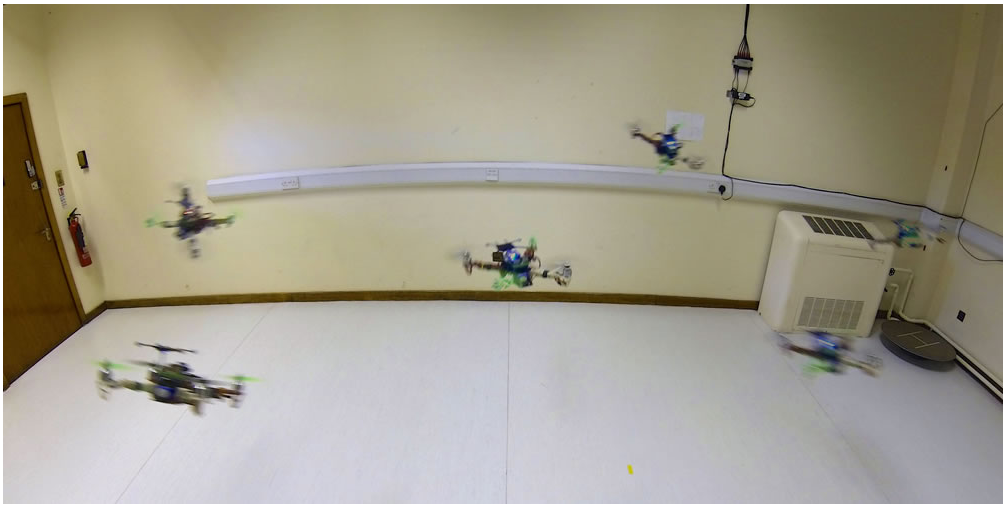


Fig. 4.30.: Time-collapse photography of a figure-of-eight trajectory performed in 4 seconds.

4.5 Summary

In this chapter the kinematic and dynamic modelling of a quadrotor was presented, followed by the contribution of the position control strategy that uses the pseudo-controls throttle, roll, pitch, yaw in order for station-keeping or maintaining the position at a desired location. Such controller strategy is experimentally tested using two types of trajectories (circle and figure-of-eight). The control strategy dynamic inversion which is a popular method for in designing non-linear controllers for the quadrotor platform, using this technique requires an accurate model of the system which is then inverted and placed in closed loop with the plant.

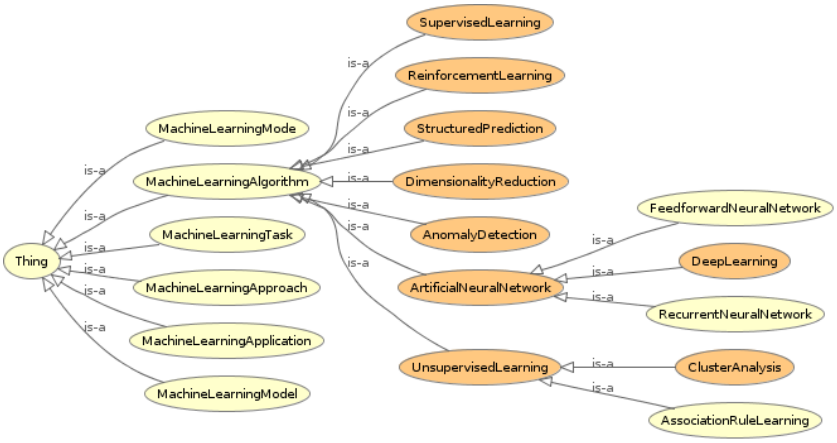
5 Machine Learning

Following a thorough literature review and detailed consideration of the research question, the principle research hypothesis of this thesis is that the dynamics (and ultimately control) of the slung load / MRUAV coupled system can be identified by applying Machine Learning (ML) techniques. ML addresses the question of how to build computer software that improves automatically through experience, not to be confused with Artificial Intelligence (AI) which has the goal of creating a machine that will mimic the human mind, in both techniques there is learning but in the latter is not just about learning, is about knowledge representation, reasoning and abstract thinking. ML is closely related to *data mining* and *statistics*. Recent progress in Machine Learning has been driven by the development of new learning algorithms that use experimental data and low-cost computation. One of the most commonly known machine learning subsets is Artificial Neural Network (ANN), inspired by the structure and functional aspects of biological neural networks. The Recurrent Neural Network (RNN) is a class of ANN that represents a very powerful generic system identification tool, integrating both large dynamic memory and highly adaptable computational capabilities. Reservoir Computing (RC) is another approach to design, train, and analyse RNNs. The main advantages of this paradigm are modelling capacity and accuracy, biological plausibility and their extensibility and parsimony. ML can often be successfully applied on problems that try to establish relationships between multiple features, improving the efficiency of systems and the design of machines. In this section, a discussion about ML will be presented. The applications of ML in this research effort will also be introduced. The goal of this section is to show the key algorithms and theory that is used to help solving the research question and hypothesis (1.1.3).

5.1 Background

ML is a computer program said to learn from experience $'E'$ with respect to some class of tasks $'T'$ and performance measure $'P'$, if its performance at tasks in $'T'$, as measured by $'P'$ improves with experience $'E'$ (Mitchell, 1997). If any computer program, system or script can improve how it performs a certain task based of past experience then we can state that it has learnt, and therefore is a machine learning application. Figure 5.1 shows a general categorization of ML algorithms and their applications. Two of the most typical application

of ML techniques are *Classification* and *Prediction*. Classification is the process whereby a machine can recognize and categorize objects/things from a particular dataset that can include visual or measurement data. Prediction, also known as regression in statistics, is where a machine can guess, predict the value of something based on previous values. The meaning of the data can lead us to another definition of ML, which is the extraction of knowledge from data. This is the main reason why ML is related to statistical analysis and data mining. Machine learning also uses concepts and results from many fields, including philosophy, information theory, biology, cognitive science, computational complexity, and control theory.



Credit: *lpag.de*

Fig. 5.1.: Machine Learning general categorization.

5.2 Categories

ML can be divided into three categories, *Supervised learning*, *Unsupervised learning* and *Reinforced learning* (Fig. 5.2). Supervised learning is about *approximation* while unsupervised learning is about *description* and reinforcement learning is about maximising a numerical *reward signal*. All categories can be valuable and which one is selected should depend on the circumstances, what kind of problem is being solved, how much time is allotted to solving it, supervised learning or unsupervised (clustering) is often faster than reinforcement learning techniques, and whether supervised learning is even possible.

5.2.1 Supervised learning

In this category the algorithm uses a known dataset (which will be called *training dataset*) in order to make predictions based on evidence in the presence of uncertainty. The training dataset includes input and output (response values) data. While the algorithm identify

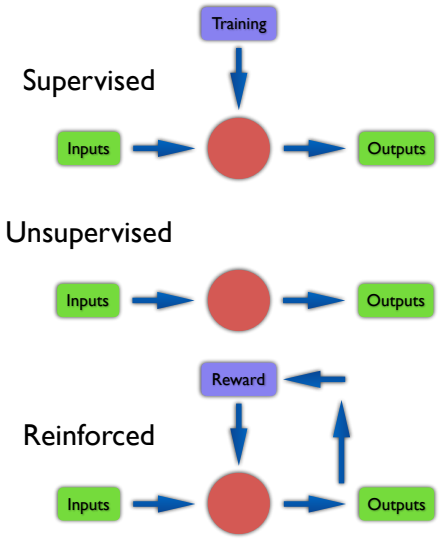


Fig. 5.2.: ML categories diagram.

patterns in data, the computer *learns* from the observations. When exposed to more observations, the computer improves its predictive performance. Using the dataset, the supervised learning algorithm seeks to build a model that can predict the output (response) for a new dataset. The model is prepared through a training process where it is required to make predictions and is corrected when those predictions are wrong. The training process continues until the model achieves a desired level of accuracy on the training dataset. The datasets are usually divided in two, the training and the test dataset, the latter one is used to validate the model. Using larger training datasets often yield models with higher predictive power that can generalize well for new datasets. This is the most used category in this research effort. The first application of ML in this thesis is the one of system identification of multirotor vehicles. Experimental flight test datasets, coming from the DronePilot (Sec. 3.5) framework, are fed into a ML algorithm, in order for it to predict the output to a certain set of inputs. In this way, the ML algorithm will learn how the vehicle moves and reacts to the control inputs. The steps to solve a problem using supervised learning are:

- Obtain and prepare the data
- Choose an algorithm
- Fit the model
- Validate the model
- Examine the fit and update until satisfied
- Use fitted model for predictions

In order to explain the steps, consider a *machine* that receives some sequence of inputs $x_1, x_2, ...$, where x_t is the sensor input at time t, this input is called the data, is usually obtained from sensors, in our case MoCap and/or Flight Controller (IMU). This machine

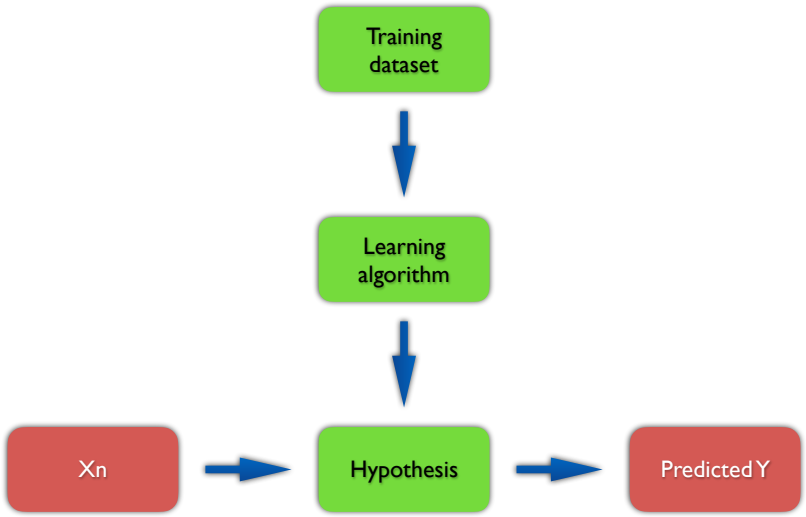


Fig. 5.3.: Supervised learning diagram.

also receives a sequence of desired output $y_1, y_2, ...,$. Both input and outputs are called the *dataset*. Then using either classification or regression (to fit the model until satisfied) given a training set, the goal is to learn a function $h : x \rightarrow y$ so that is a good predictor for the corresponding value y, such function is usually called *hypothesis*, the process is therefore showed in Fig. 5.3. Supervised learning uses two ML techniques, classification and regression (prediction). From figure 5.3, when the predicted variable (y_n) is continuous, the learning problem is a regression one, and when y_n is just discrete values, then its a classification problem. The main objective of **classification** is to assign a *class* from a finite set of classes to an observation, therefore the response are categorical variables. In prediction (**regression**), the objective is to predict a continuous measurement for an observation, therefore the response is numerical values. This is the technique used for system identification of multirotors. Between supervised and unsupervised learning is *Semi-supervised learning*. This approach is used for the same applications as supervised learning. But it uses both labelled and unlabelled data for training. It uses unlabelled data because is less expensive and takes less effort to acquire. This type of learning can be used with methods such as classification, regression and prediction.

Classification

This technique involves the problem of identifying to which set of categories the new observation or measurement belongs. Classification is an example of pattern recognition, often the individual observations are analysed into a set of quantifiable properties (features). The algorithm that implements this classification is called *classifier*, which is a mathemat-

ical function that maps input data into a category. The classification algorithms include several procedures as in Alpaydn, 2014 including frequentist, bayesian, binary, multiclass, feature vector and linear classifiers:

- Neural Networks
- Learning vector quantization
- Decision trees
  - Random forest
- Kernel estimation
  - k-nearest neighbour
- Quadratic classifiers
- Linear classifiers
  - Fishers linear discriminant (Fisher, 1936)
  - Logistic regression
  - Perceptron
  - Naive Bayes classifier
- Support vector machines (SVM)

Decision tree methods construct a model of decisions made based on actual values of attributes in the data. Decisions fork in tree structures until a prediction decision is made for a given record. Decision trees are trained on data for classification and regression problems. Decision trees are often fast and accurate and a preferred method in machine learning. The most used procedure on this research effort are the Artificial Neural Networks. The main reason is because with the recent progress in ANN it provides new tools for modelling, estimation and control of complex non-linear systems. In the case of neural networks, the classification is used to determine the error of the network and then adjust the network to minimize it.

Prediction

Also called *regression*, is a statistical process for estimating the relationships between inputs/outputs, in other words, this analysis understands which among of the input variables are related to the output variables and then it explores the forms of the relationships. However, it is common to find false relationships (*illusions*), therefore caution is needed when using this type of methods because *correlation does not imply causation*. It is often recommended to have knowledge a priori about the possible relationships. Several methods have

been developed, being the most common *linear regression* and *ordinary least squares* which are parametric. The regression function is defined in terms of a finite number of unknown parameters that are estimated from the datasets. A non-parametric regression is the one that allows the function to lie in a specified set of infinite-dimensional functions. The most popular regression algorithms are:

- Ordinary Least Squares Regression (OLSR)
- Logistic regression
- Linear regression
- Stepwise regression
- Locally Estimated Scatterplot Smoothing (LOESS)
- Multivariate Adaptive Regression Splines (MARS)

Prediction methods are a workhorse of statistics and have been cooped into statistical machine learning, this causes confusion because regression can be used to refer to the class of problem and the class of algorithm, and in reality regression is a process.

5.2.2 Unsupervised learning

This algorithm uses no labels (no output data), leaving it on its own to find in its input. A model is prepared by deducing structures present in the input data. It may use a mathematical process to systematically reduce redundancy. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. Algorithms are left to their own to discover and present the structure of the data. There is two commonly used approaches to follow when applying unsupervised learning. The first one is to teach the agent not by giving explicit categorizations, but by using a reward system to indicate success. This type of training will generally fit into the decision problem framework because the goal is not to produce a classification but to make decisions that maximize rewards. The second approach is the *cluster analysis*, which is used on data analysis to find hidden patterns or grouping data. In this type of learning, the goal is not to maximize a utility function, but simply to find similarities in the training data. The assumption is often that the clusters discovered will match reasonably well with an intuitive classification. Unsupervised learning algorithms are designed to extract structure from data samples (Ghahramani, 2008). Unfortunately, unsupervised learning also suffers from the problem of over-fitting the training data, this issue is also called *lack of robustness*. The quality of a structure is measured by a cost function which is usually minimized to infer optimal parameters characterizing the hidden structure in the data. Clustering can be useful when enough data is present to form clusters and especially when additional data about

members of a cluster can be used to produce further results due to dependencies in the data. Classification learning is powerful when the classifications are known to be correct and it is often necessary when the decisions made by the algorithm will be required as input somewhere else.

5.2.3 Reinforced learning

In this approach, the training information supplied to the learning algorithm by the environment (external trainer) is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates. The learning algorithm is not told which actions to take, therefore it must discover which actions generates the best reward, by trying each action in turn. Often, a form of reinforcement learning can be used for unsupervised learning, where the agent bases its actions on the previous rewards and punishments without necessarily even learning any information about the exact ways that its actions affect the world. In contrast, when reinforcement learning involves supervised learning, it does so for specific reasons that determine which capabilities are critical and which are not. This type of learning has three primary components, the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time (Fig. 5.4). The agent will reach the goal much faster by following a good policy. So the goal in reinforcement learning is to learn the best policy. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. The agent must operate despite significant uncertainty about the environment it faces.

Fig. 5.4.: Reinforced learning diagram.

5.3 Considerations

There is three very important factors to consider when applying a learning algorithm method. The first one the heterogeneity of the data, if such data involves features of many different

kinds like discrete, discrete ordered, counts and/or continuous values, some algorithms are easier to apply than others. For example SVM, linear regression, neural networks, require that the input data be numerical and scaled to similar ranges. In comparison, decision trees have the advantage that they can easily use heterogeneous data. The next factor is related to the redundancy of the data, if the data contains highly correlated features some algorithms like linear regression, logistic regression will have a poor performance, because of numerical instabilities. This problem can be overcome or solved by using some form of regularization of the data prior to learning. The last factor is the presence of non-linearities and interactions. If the input data makes an independent contribution to the output, then algorithms based on linear functions such as linear regression, SVM, Naive Bayes will perform generally well. When choosing the algorithm, comparison with multiple learning algorithms is suggested to see which one works best on the problem at hand, such technique is called cross validation. Tuning the performance of a learning algorithm can be very time-consuming, important to mention that in this research effort, several learning and optimising methods have been tested.

5.4 Artificial Neural Networks

A neural net is a machine learning technique modelled trying to replicate the work of neurons inside a biological brain. The idea is that given a number of inputs the neuron will propagate a signal depending on how it interprets the inputs. In machine learning terms this is done with matrix multiplication along with an activation function. ANN are a class of pattern matching that are commonly used for regression and classification problems but are really an enormous sub-field composed of hundreds of algorithms and variations for all manner of problem types. The artificial network approach has superior characteristics when compared with conventional computers Omatu et al., 1996. If we consider the Von Neumann conventional computers approach properties, as showed in Tab. 5.1, the neural network approach has superior characteristics (Tab. 5.2), and therefore when using neural networks we can expect to inherit at least some of the properties and advantages.

Symbolic expression
Logical representation
Local memory
Serial processing architecture
Sequential algorithm programming

Tab. 5.1.: Von Neumann computer properties.

One of the most important properties of neural networks are the parallel, distributed and self-organization. Parallel architecture refers to the capability of information processing on

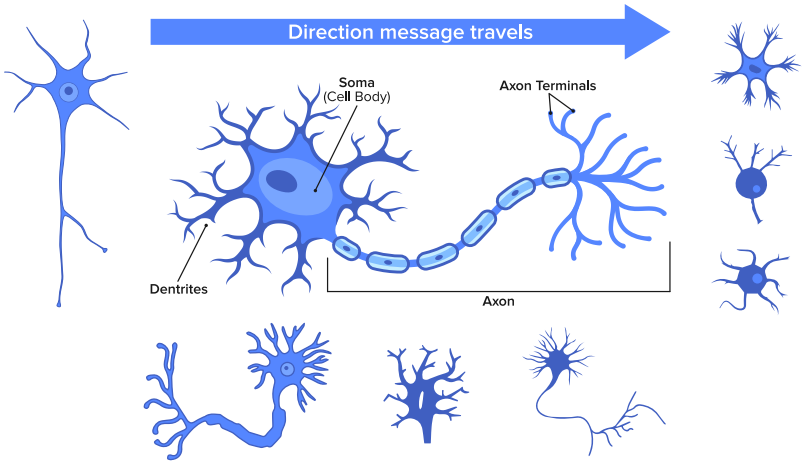
Easier pattern information processing
Self-organization
Distributed memory
Parallel processing architecture
Learning

Tab. 5.2.: Neural network properties.

many central processing units (CPUs), while in conventional computer has only a single CPU. Distributed memory means that information is stored in many addresses in a shared way so that each entity is represented by a pattern of activity distributed over many computing elements and each element is involved in representing many different entities, using such representation architecture, information can be divided and computed into several different parts (Rumeihart et al., 1968). Another peculiar and relevant properties, for this research work, are the self-organization and learning properties of neural network, this means that neural networks can learn static or dynamic properties autonomously based on the past history of measurement data (supervised learning) and then a better solution can be obtained under unknown environmental conditions. This is exactly the reversed case of conventional computers, where they cannot take decisions under new environments that have not been pre-programmed into.

5.4.1 Biological Neural Networks

Artificial neural networks are pretty much inspired by the biological nervous system. Most living creatures have the ability to adapt to different environments, therefore they need a controlling unit which is able to learn (brain). Higher developed animals and humans use very complex networks of highly specialized neurons to perform this task. The brain, or control unit, can be divided in different functional sub-units, each sub-unit have certain tasks like vision, hearing, motor and sensor control. The brain is connected by nerves to the sensors and actors in the rest of the body. Extremely important to take into account is that biological neural networks are recurrent 5.5. A neuron (nerve cell) is a special biological cell that processes information (Fig. 5.5). It is composed of a cell body (soma), axon and dendrites (tree-like branches). The neuron receives signals (impulses) from other neurons via the dendrites (receivers), then it generates a signal in the soma and transmits along the axon, which eventually branches into strands. At the terminal ends of such strands are the *synapses*. The synapse is an elementary structure and functional unit between two neurons. When a impulse reaches the synapse's terminal, there are chemicals called *neurotransmitters* which are released. The synapse's effectiveness can be adjusted by the signals passing through it, in this way the synapse learn from the activities they participate into. The brain (cerebral cortex) consists of a very large number of neurons, about 10^{11} in average.



Source: modified from *vecteezy.com*

Fig. 5.5.: Sketch biological neuron.

Neurons are massively connected between each other, they are more complex and dense than telephone networks (Brunak et al., 1990). Each neuron is connected to 10^3 to 10^4 other neurons. The human brain contains 10^{14} to 10^{15} interconnections. Neurons communicate through a train of short pulses, approximately milliseconds in duration. Complex perceptual decisions e.g. face recognition, takes around a few hundred hertz in frequency. Such type of decisions are made in a network of neurons with operational speed of a few milliseconds, this implies that computations cannot take more than about 100 serial stages, this is know as the *hundred step rule* (Feldman et al., 1988), which states that the brain runs parallel programs that are about 100 step long.

Computational model of a neuron

In McCulloch et al., 1943, a binary threshold unit was proposed as a computational model of a neuron. The mathematical representation can be seen on equation 5.1 and the diagram in Fig.5.6.

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right), \tag{5.1}$$

The model computes a weighted sum of its n input signals, x_j ($j = 1, 2, ..., n,$) and generates and output of 1 if such sum is above a certain threshold u, otherwise an output of 0 will result. In equation 5.1, $\theta(\cdot)$ is a unit step function of 0, and w_j is the synapse weight associated with the j_{th} input. The threshold u is considered as another weight $w_0 = -u$ attached to the neuron with a constant input $x_0 = 1$. McCulloch et al., 1943 proved that with a properly chosen weights, an arrangement of neurons could perform universal compu-

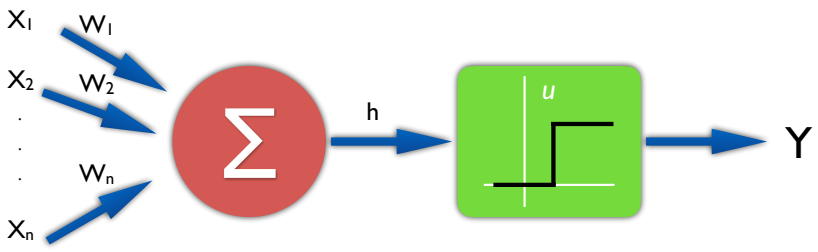


Fig. 5.6.: McCulloch-Pitts model of a neuron.

tations. The analogies with the biological neurons include wires and interconnections with axons and dendrites, connection weights with synapses, threshold functions with the activities in the soma. Such model simplifies assumptions that do not reflect the true behaviour of biological neurons.

5.4.2 ANN Architectures

Based on the connection pattern (architecture) of the ANN, they can be grouped into two main categories: *feed-forward* and *recurrent* networks. ANNs are viewed as directed weighted graphs in which the nodes are the artificial neurons and the directed edges are connections between neuron outputs and inputs. In the feed-forward networks the graphs have no loops while on the recurrent networks (also called feedback) the loop occurs because of the feedback connections. In the most common family of feed-forward networks

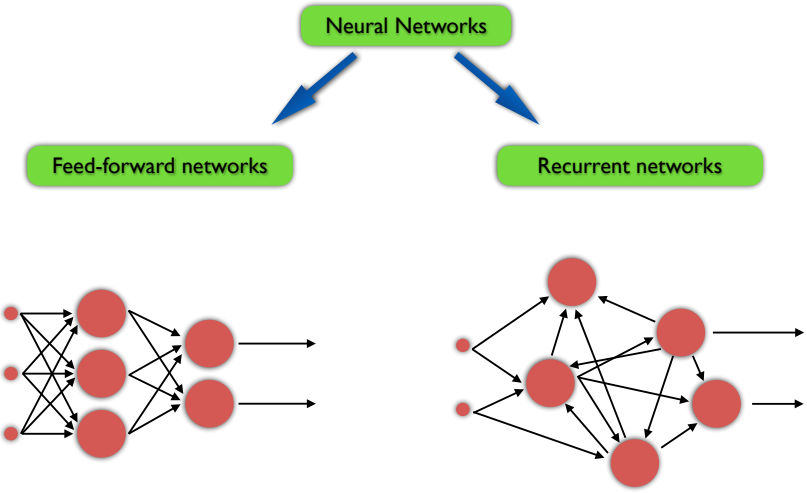


Fig. 5.7.: Architecture of feed-forward and recurrent neural networks.

(multilayer perceptron), neurons are organized into layers that have unidirectional connections between them as showed in Fig. 5.9. Such networks are static, because they produce only one set of output values rather than a sequence of values from a given inputs, they are also memory-less due to the fact that their response to an input is independent of the

previous network state. A modified version or a MLP has being used on this research effort (see Section 6) in order to perform system identification of a slung-load/quadrotor system. Recurrent networks, on the contrary, are dynamic systems, such that when a new input pattern is presented, the neuron outputs are computed. The feedback paths modify the inputs of each neuron causing the network to enter in a new state.

5.4.3 ANN Learning

The learning process of a ANN can be viewed as the problem of updating network architecture and connection weights so that the network can efficiently perform a specific task. The network usually learn the connection weights from available training patterns (supervised training). Performance is improved over time by iteratively updating the weights in the network. One of the most attractive properties is their ability to automatically learn from examples, instead of following a set of rules specified by human experts, this is one major advantage over traditional expert-systems (Myers, 1986). To overview the learning process of an ANN a model of the environment in which the network operates is need, this model is usually called the *learning paradigm*, also the process of how the network weights are updated must be understood. The learning algorithm refers to the procedure in which learning rules are used for adjusting the weights. As explained before (5.2) there is three main learning paradigms: supervised, unsupervised and reinforced (discussed before). The *learning theory* must then address three fundamental and practical issues associated with learning from samples, those are *capacity*, *sample complexity* and *computational complexity*. *Capacity* concerns how many patterns can be stored and the functions and decision boundaries that the network can form. *Sample complexity* determines the number of training patterns needed to guarantee a valid generalization, e.g. if too much patterns are supplied may cause *over-fitting*. *Computational complexity* refers to the time required for a learning algorithm to estimate a solution from the training patterns (data). The latter topic is important and very relevant for the current research work. There are four basic types of learning rules, Hebbian, competitive learning, Boltzmann and error correction.

Hebbian Rule

This learning rule is based on observations from neuro-biological experiments (Hebb, 1949). The main idea is that if neurons on both sides of a synapse are activated synchronously and repeatedly, the synapse's strength is selectively increased. The *Hebbian rule* is described as,

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_j(t) x_i(t) \tag{5.2}$$

where x_i and y_j are the output values of neurons i and j, which are connected by the synapse w_{ij} and η is the learning rate. A property of this rule is that learning is done locally and the change in synapse weight depends only on the activities of two neurons connected to it.

Competitive Learning

In competitive learning, the output units compete among themselves for activation, which provokes that only one output unit is active at any given time, this phenomenon is called *winner-take-all* and is different from the Hebbian learning where multiple output units can be fired simultaneously. The input data in this learning technique is often clusterized and categorized, similar patterns are grouped and represented as a single unit, such grouping is done automatically based on data correlations. A simple competitive rule can be seen as

$$\Delta w_{ij} = \begin{cases} \eta(x_j^u - w_{i*j}), & i = i^*, \\ 0, & i = i^*. \end{cases} \tag{5.3}$$

It is noted that only the weights of the winner unit get updated. The effect of this rule is to move the stored pattern in the winner unit closer to the input pattern. Also, this rule will not stop learning (updating the weights) unless the learning rate η is 0, therefore a particular input pattern can fire different iterations during learning which introduces a stability issue. The system is stable if no pattern in the training changes after a finite number of learning iterations.

Boltzmann Learning

This rule is composed of symmetric recurrent neural networks consisting of binary units, $+1$ for *on* and -1 for *off*, they are also called Boltzmann Machines. This rule is an *stochastic* method derived from information-theoretic and thermodynamic principles as in Anderson et al., 1988. The weight of the connection from unit i to unit j is equal to the weight on the connection from unit j to unit i, therefore $w_{ij} = w_{ji}$. They contain two subset of neurons, visible and hidden. The first ones interact with the environment and the latter do not. Each neuron is a *stochastic* unit that generates an output according to the Boltzmann distribution of statistical mechanics. The change in the connection weight w_{ij} is

$$\Delta w_{ij} = \eta(\bar{\rho_{ij}} - \rho_{ij}) \tag{5.4}$$

where η is the learning rate, $\bar{\rho_{ij}}$ and ρ_{ij} are correlations between the states of units i and j. The values $\bar{\rho_{ij}}$ and ρ_{ij} are estimated using *Monte Carlo* experiments and they can be extremely slow.

Error Correction

The basic principle of the error-correction rule is to use the error signal $(d - y)$ to modify the connection weights to gradually reduce the error. One example of this error-correction principle is the perceptron learning rule. A perceptron is formed of a single neuron with adjustable weights, w_j, being $j = 1, 2, ..., n$, and threshold u, the net input to the neuron is

$$v = \sum_{j=1}^{n} w_j x_j - u \tag{5.5}$$

for an input vector $x = (x_1, x_2, ...x_n)^t$. The output y of the perceptron is $+1$ if $v > 0$, and 0 otherwise. Important to notice that learning occurs only when the perceptron makes an error. The *perceptron convergence theorem* states that the learning procedure converges after a finite number of iterations. The back-propagation learning algorithm is based on this learning rule, and many variations of this algorithm have been proposed in the literature (Hertz et al., 1991).

5.5 Recurrent Neural Networks

Recurrent neural networks are a powerful set of artificial neural network algorithms especially useful for processing sequential data such as sound, time series (sensor) data or written natural language, this is one of the reasons it proved to be a powerful tool to analyse the flight characteristics of unmanned aerial aircraft. Also, they perform well when the training data may contains errors as well as being robust to noise. The information persists in them because they contain at least one feed-back connection, therefore the activations run in a loop. Such behaviour allows them to do temporal processing and perform sequence recognition/reproduction as well as temporal association and prediction. This capability of operation with sequences of vectors, either on the input or in the output, is one of the reasons why RNN are more powerful compared with fixed networks. They are considered *Turing-complete* (Siegelmann, 1995) because they can simulate arbitrary programs (with the proper weights). The diagram of a typical RNN is showed in Fig.5.8, such diagram shows an unfolded version into a full network, that is showing the complete sequence of the network. X_t in the input at time step t. h_t is the hidden state at time step t which is the *memory* of the network, h_t is computed based on the previous hidden state and the input

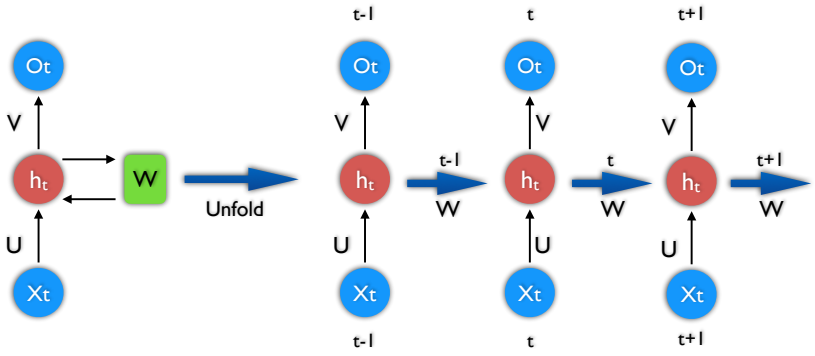


Fig. 5.8.: An unfolded Recurrent Neural Network.

at the current step $h_t = f(U_{X_t} + W_{h_{t-1}})$, the activation function usually is a non-linearity such as the hyperbolic tangent ($tanh$), h_{t-1} is typically initialized to all *zeroes* (first hidden state). O_t is the output at time step t. Note that time must be *discretized*, with the activations updated at each time step, the time scale might correspond to the operation of real neurons, a *delay unit* needs to be introduced to hold activations until they are processed at the next time step. The above diagram has outputs at each time step, but depending on the task this may not be necessary. The main feature of an RNN is its hidden state (h_t), which captures some information about a sequence. It is noticed that at Fig. 5.8 shows a unidirectional flow of information from the input units to the hidden units as well as another unidirectional flow of information from the hidden units to the output units, In some cases, RNN break the latter restriction with connections leading from the output units back to the hidden units, these are called *back-projections*. Mathematically, a RNN can be described as a *State Space Model* (dynamical system), where the *state* of the dynamical system is a set of values that summarizes all the information about the past behaviour of the system that is necessary to provide a unique description of its future behaviour, apart from the effect of any external factors, in the neural network case the state is defined by the set of hidden unit activations $h(t)$ and can be described in:

$$\begin{aligned} h(t) &= f_H(W_{IH}x(t) + W_{HH}h(t-1)) \\ y(t) &= f_O(W_{HO}h(t)) \end{aligned} \tag{5.6}$$

where the inputs and the outputs are the vectors $x(t)$ and $y(t)$, W_{IH}, W_{HH}, W_{HO} the three connection weight matrices and f_H, f_O the hidden unit activation functions. In addition to the input and output spaces, there is also a *state space*. The order of the dynamical system is the dimensionality of the state space which is the number of hidden units. Following on the properties of dynamical systems, in terms of RNN, the *stability* concerns the boundedness over time of the network outputs and the response of the network outputs to small changes while the *controllability* refers to whether it is possible to control the dynamic behaviour of

the recurrent neural network and it is said to be controllable if an initial state is steerable to any desired state within a finite number of time steps. Lastly the property of *observability* concerns if it is possible to observe the results of the control applied to network which is done if the state of the network can be determined from a finite set of input/output measurements. It is important to mention the *universal approximation* capabilities of the recurrent neural networks. The Universal Approximation Theorem (Csaji, 2001) states: *Any non-linear dynamical system can be approximated to any degree of accuracy by a recurrent neural network, with no restriction on the compactness of the state space, provided that the network is given sufficient sigmoidal hidden neurons*. However, knowing that a RNN can approximate any dynamical system does not tell us how to achieve it.

5.5.1 Mathematical Model

There is three formal types of RNN mathematical models, *discrete-time models* are iterated over discrete time steps $n = 1, 2, 3, ...,$, *continuous-time models* are defined through differential equations whose solutions are defined over a continuous time t, *spikes* are especially designed for purposes of biological modelling which are continuous dynamical models that describe activation signals on the level of individual action potentials. If we consider a discrete-time model with K input units with activation vector $\mathbf{u}$, N internal units (neurons), L output units, $\mathbf{W}^{in}$ input connection weights, $\mathbf{W}$ internal connection weights, $\mathbf{W}^{out}$ output connection weights and a back-propagation weight matrix $\mathbf{W}^{back}$, the activation of internal units is updated according equation:

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}(n)) \tag{5.7}$$

Where $\mathbf{u}(n+1)$ is the externally given input, while $\mathbf{f}$ is the individual units transfer function (usually a sigmoid function $f = tanh$), the output is computed as:

$$\mathbf{y}(n+1) = \mathbf{f}^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))) \tag{5.8}$$

Where $\mathbf{u}(n+1), \mathbf{x}(n+1)$ denotes the concatenated vector composed by the input and internal activation vectors, while $\mathbf{f}^{out}$ remains $f = tanh$.

5.5.2 Architectures

Recurrent neural network architectures can have many different forms including Elman Networks, Hopfield Network, Fully Recurrent Network, Echo State Networks (ESN), Jordan Networks, Modified MLP (Multi-Layer Perceptron), Long Short Term Memory (LSTM), Recurrent Multi-Layer Perceptron (RMLP), among others. Although several recurrent neu-

ral network architectures were tested on this research effort, the most important and documented ones are the MMLP and the ESN, this is due to the fact that they proved to converge better when the patterns of the data change through time. These *deep learning* models have a simple structure with a built-in feedback loop allowing them to act as a forecast engine, which is one of the applications used on this research. When a RNN is trained GPU (Graphics Processing Unit) have a tremendous advantage over ordinary CPU (Central Processing Unit), this was validated in Chen et al., 2014 where they compared the speed boost from a GPU training over a CPU. The GPU had a 250-fold increase, which means training a RNN in e.g. 3 hours for the GPU and over a month for the CPU. Another option for training neural networks that use large amounts of data coming from sensors, such as the one explored in this thesis, is to use *grid computing*, which can be defined as a collection of computer resources from multiple locations to reach a common goal. Neural network training can be culminated in an extremely fast time frame if grid computing is used.

Recurrent MLP

A modified version of a Multi-Layer Perceptron (MLP) can exploit the powerful non-linear mapping capabilities of neural networks. Their mode of operation is such that the units each perform a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output, and the units are arranged in a layered feed forward topology. Figure 5.9 shows an example diagram for a normal MLP with two inputs,

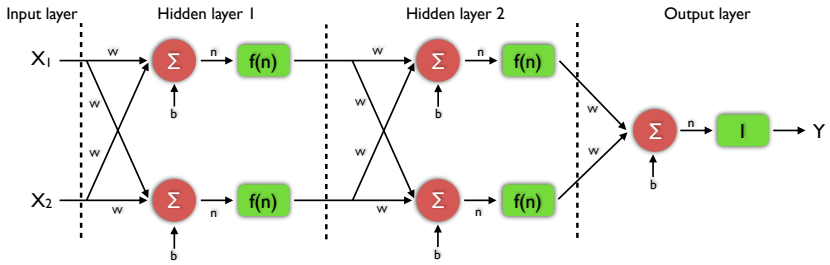


Fig. 5.9.: Example diagram for a MLP.

two hidden layers with two neurons each and a output layer with one neuron, this diagram shows how a MLP is described. The neurons are the same as the one defined in Fig. 5.6 plus a bias weight b. They can have several inputs (X_1, X_2) which are multiplied by the connection weights W and summed up together with the bias weight b to the summation output n. Then the neuron output Y is calculated using the transfer function (activation function) $f(n)$, which can be linear function or most commonly a hyperbolic tangent. To create a recurrent MLP, we could connect the output of a layer with the input of a previous layer, this is done by using a *real-value time-delay*, such technique is called Tapped Delay

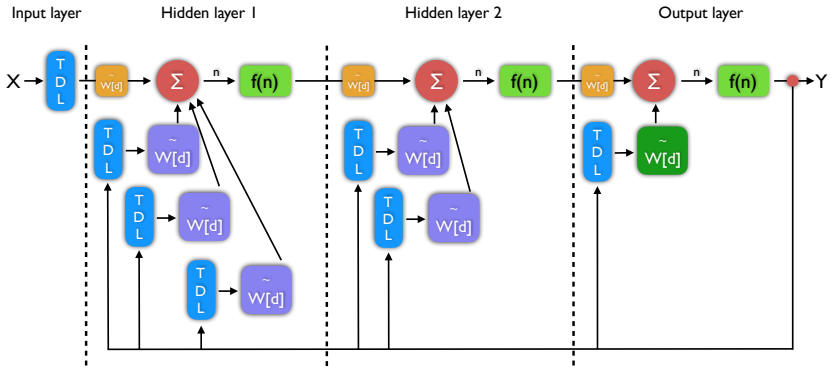


Fig. 5.10.: Example diagram for a Recurrent MLP.

Lines (TDL). A delay-line tap extracts a signal output from somewhere within the delay line and typically sums with other taps to form a TDL output signal (Smith, 2010). Figure 5.10 shows an example of a recurrent MLP with delays on the inputs, hidden and output layer. The TDL contains delay operators z^{-d} which delay time-discrete signals by a value d. For every delay d_i there has to be a connection matrix $\widetilde{W}[d_i]$, consequently there is three types of delays. The *input delays*, allows to delay the inputs X of the neural network by any real-valued time-step d, and the neural network can be used for systems where the output depends not only on the current input, but also previous inputs. The *output delays* add a recurrent connection of the outputs Y of the neural network to the first layer and the neural network can be used for systems where the output depends not only on the inputs, but also on previous outputs (states). The *internal delays* add a recurrent connection from all layers to all previous layers and to it self (except from the output layer to the first layer) and the neural network can be used for systems where the output depends on previous internal states.

Echo State Network

The echo state network consists of a recurrent neural network with a sparsely connected random hidden layer, with the characteristic that only the weights of output neurons are the part of the network that can change and be trained. ESN are good at reproducing certain time series therefore have being using in this research effort to help perform system identification of small unmanned aircraft Vargas et al., 2015b and also in control tasks Vargas et al., 2014. A section is dedicated to this algorithm in 5.6.1.

5.5.3 Training

There is several types of training algorithms which are well known on the literature, there is no a clear winner among those. The most common method to minimize the total error is *gradient descent*, which is a first-order iterative optimisation algorithm. The methods used in this research effort will be described in the next sub-sections, with the exception of the ESN training, which will be described in a different section, due to the importance of it. In the most simplistic approach to train neural networks, we can consider a system that produces a output Y, when a input X is given. Figure 5.11 shows such simple system where (x, y) represents one sample of training data. As discussed in past Sections 5.2.1, training requires

Fig. 5.11.: Example diagram for a system with input X and output Y.

more than one sample of data to obtain good results. Therefore the training data is defined by the matrix $\widetilde{X}$ and the target (output) matrix with $\widetilde{Y}$, both containing N samples of data. Important to notice that the samples have to be in the correct time order and the training data should represent the system as good as possible. Training a neural network (Fig.

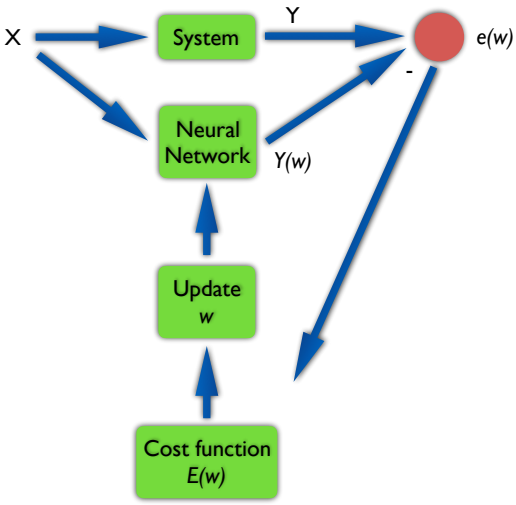


Fig. 5.12.: Example diagram of training a neural network.

5.12) means that all weights in the vector w, which contains the connection weights and biases, are updated step by step, such that the neural network output ($Y(w)$) matches the training output data (Y) target. The objective of this optimisation is to minimize the error $E(w)$ (cost function) between neural network and system outputs and the training repeats adapting the weights of the weight vector w until one of the two termination conditions becomes active, such conditions are the maximal number of iterations (epochs) is reached

and/or the error is minimized to the goal E_{stop}. Recurrent Neural Networks can be trained using several methods which have been explored in Jaeger, 2005a, the most important and used are Back-propagation Through Time (BPTT), Real-Time Recurrent Learning (RTRL) and Extended Kalman Filtering (EKF) based techniques, being the first two methods the ones used in this research.

BPTT

Back-propagation is a gradient-based technique used to train several types of neural networks, not only recurrent. Back-propagation Through Time is an adaptation of the well-known back-propagation training method known from feed-forward networks. It is an extension of perceptrons or multi-layered neural networks, thus it employs at least three or more layers of neurons (input, hidden and output) interconnected to every unit of the previous layer. Detailed work on BPTT can be found at Werbos, 1990 Beaufays et al., 1994 Guo, 2013. In feed-forward, the error derivatives are calculated with respect to the weights of one layer, they can be expressed completely in terms of the error derivatives from the layer above, the issue with recurrent neural networks is that they don't have this ordered layering because the neurons do not form a *directed acyclic graph* (DAG), therefore a transformation is needed for pseudo-converting a RNN into a feed-forward neural network, this process is called *unrolling* or *unfolding* and it was showed in Fig. 5.8. The steps of the training method of BPTT can be seen next.

1. For a sample n, the activations are computed using a forward pass as

$$x_i^{m+1}(n) = f(\sum_{j=1,\dots,N^m} w_{ij}^m x_j(n)) \tag{5.9}$$

2. Backward compute through $m = k+1, ..., 1$, for each unit x_i^m the error propagation term $\delta_i^m(n)$ as seen in equation 5.10 for the output layer and equation 5.11 for the hidden layers,

$$\delta_i^{k+1}(n) = (d_i(n) - y_i(n))\frac{\delta f(u)}{\delta u}\Big|_{u=z_i^{k+1}} \tag{5.10}$$

$$\delta_j^m(n) = \sum_{i=1}^{N^{m+1}} \delta_i^{m+1} w_{ij}^m \frac{\delta f(u)}{\delta u}\Big|_{u=z_j^m} \tag{5.11}$$

where 5.12 is the internal state of the neuron x_i^m, which is the error back-propagation pass, the error propagation term $\delta_i^m(n)$ represents the error gradient with respect to the potential of the neuron x_i^m.

$$z_i^m(n) = \sum_{j=1}^{N^{m-1}} x_j^{m-1}(n) w_{ij}^{m-1} \tag{5.12}$$

$$\frac{\delta E}{\delta u}\Big|_{u=z_j^m} \tag{5.13}$$

3. Finally, an adjustment to the connection weights is made according to

$$\text{new } w_{ij}^{m-1} = w_{ij}^{m-1} + \gamma \sum_{t-1}^{T} \delta_i^m(n) x_j^{m-1}(n) \tag{5.14}$$

After every such epoch, the error is computed as

$$E = \sum_{n=1,\ldots,T} \left\| \mathbf{d}(n) - \mathbf{y}(n) \right\|^2 = \sum_{n,\ldots,T} E(n) \tag{5.15}$$

The training stops when the error falls below a predetermined threshold or when the number of iterations exceeds a predetermined maximum number of them.

The basic gradient descent approach (and its back-propagation algorithm implementation) is notorious for slow convergence, because the learning rate γ must be typically chosen small to avoid instability. Many speed-up techniques are described in the literature (Plagianakos et al., 2001), e.g. dynamic learning rate adaptation schemes. One of the problems with this method is the selection of a suitable network topology (number and size of hidden layers), which can be overcome using prior information, performing systematic search and/or intuition. Like all gradient-descent techniques on error surfaces, back-propagation finds only a local error minimum, such problem can be addressed by adding *noise* during training to avoid getting stuck in poor minima, or by repeating the entire learning from different initial weight settings, or by using task-specific prior information to start from an already plausible set of weights. This method can be easily implemented, yet a considerable expertise/experience is a requisite for good results in non-trivial tasks. An implementation and upgrade for the BPTT was made as a comparison of several training methodologies that were used on this research effort, such implementation made usage of the *Broyden–Fletcher–Goldfarb–Shanno* algorithm (BFGS) and is similar to Nawi et al., 2006. The BFGS algorithm is an iterative method for non-linear optimisation problems. This algorithm is a second order optimisation method that uses rank-one updates specified by evaluations of the gradient to approximate the Hessian matrix. The gradient is calculated

using a BPTT technique based on Werbos, 1990. Experiment results using this algorithms will be showed on Chapter 6.

RTRL

Real-Time Recurrent Learning (RTRL) is a gradient-descent method which computes the exact error gradient at every time step. This method has been analysed and reviewed in Chow et al., 1998 Williams et al., 1989 Cios et al., 1997. If the network activation of the internal (Eq. 5.7) and output units (Eq. 5.8) are differentiated with the weights, the effect of the network dynamics change. The derivative of an internal or output unit with respect to a weight w_{kl} is given by equation 5.16, where w_{kl} includes all input, internal and output units weights. Equation 5.16 constitutes a $(N+L)$-dimensional discrete-time linear dynamical system with time-varying coefficients, with dynamical variable 5.17

$$\frac{\delta v_i(n+1)}{\delta w_{kl}} = f'(z_i(n))\left[(\sum_{j=1}^{N+L} w_{ij}\frac{\delta v_i(n)}{\delta w_{kl}}) + \delta_{ik}v_l(n)\right] \tag{5.16}$$

$$\left(\frac{\delta v_i}{\delta w_{kl}}, ..., \frac{\delta v_{N+L}}{\delta w_{kl}}\right) \tag{5.17}$$

Where $i = 1, ..., N+L$, $z_i(n)$ is the unit potential, $\delta_{ik}v_l(n)$ represents the effect of the weight W_{kl} onto the unit k. Since the initial state of the network is independent of the connection weights, it can be initialized by making 5.17 equal to 0. Thus the computation forward in time can be done by iterating 5.16 simultaneously with the recurrent neural network dynamics 5.7 and 5.8, and the error gradient can be calculated as

$$\frac{\delta E}{\delta w_{kl}} = 2\sum_{n=1}^{T}\sum_{i=N}^{N+L}(v_i(n) - d_i(n))\frac{\delta v_i(n)}{\delta w_{kl}} \tag{5.18}$$

And the new weight update after a complete iteration or epoch can be done with a standard batch gradient descent algorithm as showed in 5.19, where γ is the learning rate.

$$\text{new } w_{kl} = w_{kl} - \gamma\frac{\delta E}{\delta w_{kl}} \tag{5.19}$$

$$w_{kl}(n+1) = w_{kl}(n) - \gamma\sum_{i=1}^{L}(v_i(n) - d_i(n))\frac{\delta v_i(n)}{\delta w_{kl}} \tag{5.20}$$

An alternative update scheme is the gradient descent of current output error at each time step (Eq. 5.20), important to notice that it is assumed that w_{kl} is a constant, therefore a small learning rate must be used. Equation 5.20 is known as real-time recurrent learning.

In RTRL the computational cost for each update step is very high, because a $(N + L)$-dimensional system (Eq. 5.16) must be solved for each of the weights, therefore this method is only recommended for small networks. An implementation of the *Levenberg–Marquardt* (LM) algorithm was made as a comparison of training methodologies for RNN. LM is a second order optimisation method that uses the Jacobian matrix to approximate the Hessian matrix, the Jacobian matrix is calculated using the RTRL method from Williams et al., 1989. The LM algorithm Levenberg et al., 1944, Marquardt, 1963 is used to solve non-linear least squares and is also known as the damped least-squares method (DLS).

5.6 Reservoir Computing

In order to overcome the downsides of traditional RNN training such as BPTT and RTRL (reviewed in the previous Section), a novel paradigm of computation with dynamical systems, named Reservoir Computing (RC), has been proposed in Verstraeten et al., 2007, which can be utilized to achieve efficient training of recurrent neural networks. Traditional neural network models are not inherently able to handle time-varying stimuli or dynamic patterns, usually to tackle this situation these network models treat time as an additional spatial dimension at the same level of the inputs which is not a plausible approach for dynamical *biological* systems. The RNN model approaches the representation of time based on recurrent connections with previous state of the network and the current sensory input Jordan, 1986 Elman, 1990, and it was mentioned in Section 5.5. The reservoir computing model proposes a randomly generated non-linear *dynamical system* with fixed weights that maps the inputs to a high-dimensional space where classification or linear regression are efficiently accomplished. Such dynamical system is called the *reservoir* (Fig. 5.13), the states of this system are linearly combined with the output layer and this is the only part that requires training thus reducing the computational effort. RC differs from traditional neural network design and learning techniques because there is a conceptual and computational separation of the internal states ($\mathbf{X}$) and the output layer ($\mathbf{Y}$), which serve different purposes, $\mathbf{X}$ expands the input history into a rich *ripple* reservoir state space while $\mathbf{Y}$ combines the neuron signals into the desired output. There are three reservoir computing models, *Echo State Networks* (ESN) Jaeger et al., 2004, *Liquid State Machines* (LSM) Maass et al., 2002 and *Backpropagation-decorrelation* (BPDC) Steil, 2004. From a machine learning perspective, a reservoir network, usually randomly generated and sparsely connected, functions as a *temporal kernel*, projecting the input to a dynamic non-linear space (Antonelo, 2011). The reservoir internal states form patterns that are dependant on the current inputs but still contains memory traces of previous stimuli, resembling *ripples* on water. The computation on the output layer occurs by by linearly reading out instanta-

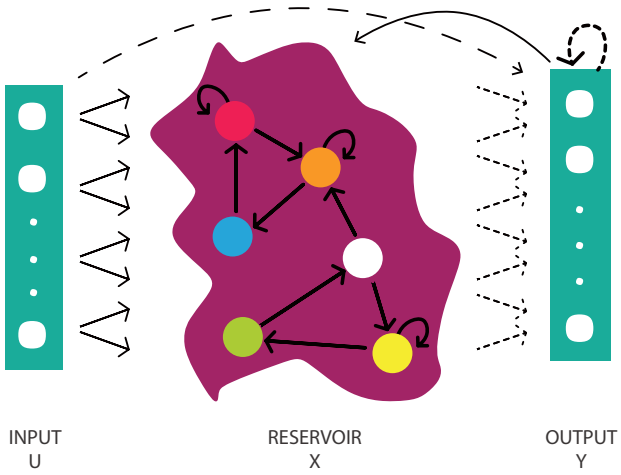


Fig. 5.13.: Reservoir computing diagram.

neous states of the reservoir, this allow reservoir architectures the capability of processing spatio-temporal patterns. The main advantages of this paradigm are modelling capacity and accuracy, biological plausibility and their extensibility and parsimony. RC has outperformed previous methods of non-linear system identification, prediction and classification Jaeger et al., 2004, NN3, 2007, Vargas et al., 2014, Verstraeten et al., 2006, this is one of the paramount capabilities needed on this research. RC architectures are computationally capable of modelling continuous-time, continuous-value real-time systems with bounded resources as in Maass et al., 2003, which is another advantage of using them on systems such as multi rotorcraft. Reservoir computing models offer a functional interpretation of the cerebellar circuitry Yamazaki et al., 2007, it can also provide explanations of why biological brains carry out *accurate* control calculations (e.g. walking or running) using *inaccurate* and *noisy* physical data Buonomano et al., 1995. RC models are not affected by the problem of neural networks known as *catastrophic interference* French, 1999, such problem relates on how new items behave in learned models without impairing or destroying previous representations, in RC the output weights of different output units are independent of each other thus not an issue. In this section, the ESN model will be presented and analysed as a reservoir computing architecture and used as one of the tools for modelling of multirotor aircraft with or without a slung-load attached to it.

5.6.1 Echo State Networks

Echo State Networks is one of the methodologies of RC (Lukoševičius, 2012). The main idea comes from a continuous neural hardware micro-circuitry, ESN have the advantage to overcome the difficulties of traditional dynamic RNN in large-scale training, it can also

approximate a non-linear system excellently and its prediction can get good results. It is practical and conceptually simple, but requires some experience and insight to achieve good performance Vargas et al., 2014. In certain RNNs, the activation of internal states $x(n)$ is a function of the input history $u(n), u(n-1), ...$ Such function $\mathbf{E}$ can be represented as $x(n) = \mathbf{E}(u(n), u(n-1), ...)$ and it can be understood as an *echo* of the input history, which is where their name comes from. An ESN is composed of a discrete hyperbolic-tangent RNN (reservoir) and of a linear readout output layer which maps the reservoir states to the actual output. As shown in figure 5.14, the connections between the neurons are random (not organized into neat sets of layers), such reservoir stays fixed randomly (at the beginning) at all moments and during training only the connections on the output layer are changed. The reservoir of ESN is constituted of analog sigmoidal neurons, such reservoir

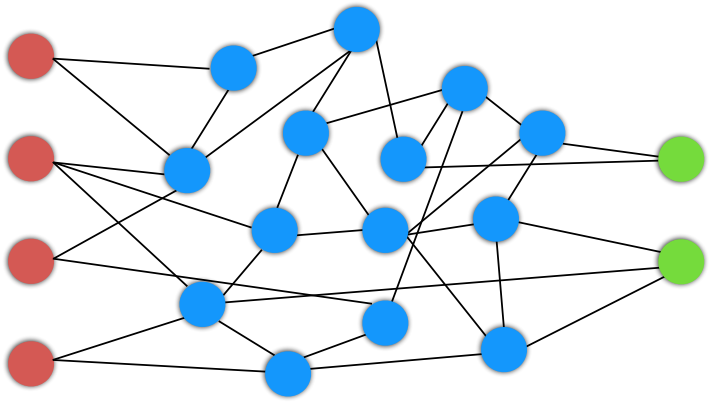


Fig. 5.14.: Example diagram of a Echo State Network.

has a theoretical property called *echo state* which the refers to the fact that the influence of inputs on reservoir states fades away gradually (Jaeger, 2001) and it is a property prior to training. Such property can be described as fading memory of the input, the trajectories of the reservoir state should converge given the same input, irrespective of the previous history, this is ensured by appropriately scaling recurrent connection weights $\mathbf{W}$. Other important parameters are the input weight $\mathbf{W}^{in}$ scaling, the *leaking rate* α, the *spectral radius* ρ and it should be adjusted for an optimal validation performance in a given task, in this research effort, a evolutionary strategy algorithm (CMA-ES) was used to find and adjust some of the parameters of this reservoir computing approach, this is going to be explained in Section 5.7. The basic steps to approach a modelling task using the ESN methodology is as follows:

1. Create a *random dynamical reservoir* RNN with any neuron model, the size $\mathbf{N}$ of it is task-dependant, examples of common sizes for the task of identifying multirotor vehicles will be provided in section 6

2. After the creation of the reservoir, the input units must be attached to the reservoir by creating random all-to-all connections

3. Create output unit layer, some tasks (with input and output data) might require output feedback if so, create randomly generated output-to-reservoir connections (all-to-all)

4. Run the reservoir with the training data $\mathbf{D}$ for times $n = 1, ..., n_{max}$, this means writing the input $u(n)$ into the input unit and the target output $y(n)$ to the output unit, this process is also called *teacher forcing*

5. Estimate the output weights as a linear regression of the teacher outputs $y(n)$ on the reservoir states $x(n)$, these newly computed weights are used to create the reservoir-to-output connections, which are the dotted arrows in figure 5.13

Important to remark that after the creation of the dynamical reservoir, the determination of optimal weights becomes a linear unique solvable task of MSE (mean squared error) optimisation, which measures the average of the squares of the errors or deviations. MSE is a risk function, corresponding to the expected value of the squared error loss or quadratic loss, the difference occurs because of randomness or because the estimator doesn't account for information that could produce a more accurate estimate (Lehmann et al., 1998).

5.6.2 Mathematical model

Let n_i represent the number of input units, n_r the reservoir units and n_o the output units, $\mathbf{u}(n)$ the n_i-dimensional external input, $\mathbf{x}(n)$ the n_r-dimensional reservoir internal activation states and $\mathbf{y}(n)$ the n_o-dimensional target output. The discrete time dynamics of the ESN is give by the state update equation 5.21, which is similar to equation 5.7 RNN internal activation.

$$\mathbf{x}(n+1) = tanh(\mathbf{W}_r^r \mathbf{x}(n) + \mathbf{W} r_i \mathbf{u}(n) + \mathbf{W}_o^r \mathbf{y}(n) + \mathbf{W}_b^r) \tag{5.21}$$

Where the weights $\mathbf{W}_{from}^{to}$ elements are described in table 5.3 and represents the connection weights between the nodes of the complete network (Fig. 5.15). b, i, r, o denotes bias, input, reservoir and outputs respectively. The output is computed as

$$\begin{aligned} \mathbf{y}(n+1) &= g(\mathbf{W}_r^o \mathbf{x}(n+1) + \mathbf{W}_i^o \mathbf{u}(n) + \mathbf{W}_o^o \mathbf{y}(n) + \mathbf{W}_b^o) \\ &= g(\mathbf{W}^{out}(\mathbf{x}(n+1), \mathbf{u}(n), \mathbf{y}(n), 1)) \\ &= g(\mathbf{W}^{out} \mathbf{z}(n+1)) \end{aligned} \tag{5.22}$$

where g is a *post-processing* activation function and $\mathbf{z}(n+1) = (\mathbf{x}(n+1), \mathbf{u}(n), \mathbf{y}(n), 1)$ is the extended reservoir state which includes the previous input, output vectors and a bias term. The weight matrices that represent the connections to the reservoir $\mathbf{W}^r_{\cdot}$ are randomly initialized and represented by solid arrows on figure 5.15. The output weights $\mathbf{W}^o_{\cdot}$ are trained and represented by dashed arrows in figure 5.15. Output feedback is given by the projection $\mathbf{W}^r_o\mathbf{y}(n)$ and bias $\mathbf{W}_b$ As stated before, the non-trainable weights $\mathbf{W}^r_{\cdot}$ are generated

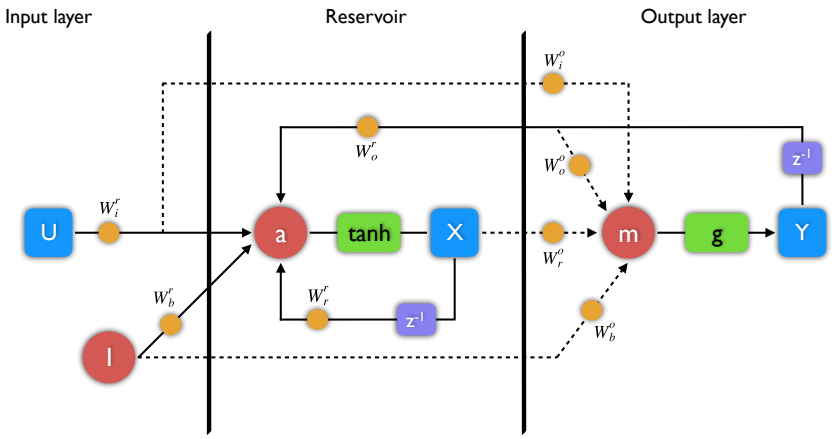


Fig. 5.15.: Echo State Network mapping scheme.

Signals	
$\mathbf{u}$	input signal
$\mathbf{y}$	output signal
$\mathbf{x}$	reservoir state
a	weighted sum for reservoir units
m	weighted sum for output units
Weights	
$\mathbf{W}^r_i$	input to reservoir connection matrix
$\mathbf{W}^r_b$	bias to reservoir connection matrix
$\mathbf{W}^r_r$	reservoir connection matrix
$\mathbf{W}^r_o$	output to reservoir connection matrix
$\mathbf{W}^o_i$	input to reservoir connection matrix
$\mathbf{W}^o_r$	reservoir to output connection matrix
$\mathbf{W}^o_o$	output to output connection matrix
$\mathbf{W}^o_b$	bias to output connection matrix

Tab. 5.3.: Elements of figure 5.15

using a sparse uniformly distributed random matrix function with a certain added *connectivity* which corresponds to the percentage of non-zero weights in the respective connection matrix $\mathbf{W}^{to}_{from}$. Also a scaling factor to the weights is applied and it corresponds to the scaling of the respective connection matrix $\mathbf{W}^{to}_{from}$ such that all weights are rescaled according to the multiplication of the scale factor and the weight matrix. The reservoir connection matrix $\mathbf{W}^r_r$ must be rescaled to comply with stability of such *dynamical system*, which states

that in some cases the rescaling have a few eigenvalues that are situated slightly outside the unity circle, but the reservoir should still exhibits rich dynamics. Also, the reservoir should guarantee the ESP (Echo State Property) (Jaeger, 2001) which means the reservoir should have a fading memory. The *spectral radius* ρ is the largest absolute eigenvalue of the reservoir connection matrix $\mathbf{W}_r^r$ and should be less than unity or else the ESP will be violated. For most applications, the best performance is attained with a reservoir that operates at the edge of stability $\rho(\mathbf{W}_r^r) = 0.99$, different values will be shown in Chapter 6.

5.6.3 Training

With the introduction and mathematical modeling presented in previous sections, we proceed to show the formal method for training a ESN network for the task that it will be used in this research effort, in such task we assume that the output units are sigmoid units, the output layer must contain feedback connections and a supervised training methodology is also assumed. The formal process it is showed next:

1. Data processing. Create/obtain input and data outputs of training and testing samples, such data must be consistent with the network structure.

2. Reservoir creation. Randomly generate the dynamical reservoirs $\mathbf{W}^{in}$, $\mathbf{W}$ and $\mathbf{W}^{back}$, such reservoir must comply with the echo state property and a spectral radius $\rho_{max} < 1$ and also they should be sparse with a rich variety of dynamics. The number of neurons $\mathbf{N}$ should reflect both the length T of training data, and the difficulty of the task (difficult tasks require a larger $\mathbf{N}$). $\mathbf{N}$ should not exceed an order of magnitude of $\frac{T}{10}$ to $\frac{T}{2}$, just to prevent *over-fitting*. The spectral radius ρ should be small for fast teacher dynamics and large for slow teacher dynamics.

3. Sample training. Enter the network input and output data samples and update the network status using equation 5.23 and collect the concatenated input/reservoir/previous-output states $(\mathbf{u}(n), \mathbf{x}(n), \mathbf{y}(n-1))$ as a new row on a *state collecting matrix* $\mathbf{M}$. Also the teacher output $tanh^{-1}\mathbf{y}(n)$ should be saved as a new row on a *teacher collecting matrix* $\mathbf{T}$.

$$\mathbf{x}(n+1) = tanh(\mathbf{W}^{in}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{5.23}$$

4. Compute output weights. Calculate values of the output by multiply the pseudo-inverse of $\mathbf{M}$ with $\mathbf{T}$, as showed in equation 5.24. To obtain the desired output weight $\mathbf{W}^{out}$, $(\mathbf{W}^{out})^t$ should be transposed.

$$(\mathbf{W}^{out})^t = \mathbf{M}^{-1}\mathbf{T} \tag{5.24}$$

5. Usage. The network $\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{back}$ and $\mathbf{W}^{out}$ is ready to be exploited and it can be driven with novel data (testing data) sequences using equations 5.25 and 5.26. The MSE for training data and testing data should be calculated to ensure the ESN is working properly, if not, the process can be repeated or optimised until a desired MSE is founded on the testing data.

$$\mathbf{x}(n+1) = tanh(\mathbf{W}^{in}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{5.25}$$

$$\mathbf{y}(n+1) = tanh(\mathbf{W}^{out}(\mathbf{x}(n+1), \mathbf{u}(n+1), \mathbf{y}(n))) \tag{5.26}$$

If stability problems are encountered when using the trained network, it very often helps to add some small noise during the sampling step, such noise should be a uniform white noise function $\nu(n)$ of sizes $[0.0001 - 0.01]$. In a experimental test, the noise was optimised to produce the lowest MSE possible on the testing data. It should be added to the weights inside the activation function in equation 5.23. This technique was proven in Jaeger, 2002a. If the system is highly non-linear, the system can be improved by adding *augmented* network states for training and in usage. The modified update augmented equation can be seen in 5.27. This method is showed and used in Jaeger, 2002b.

$$\mathbf{y}(n+1) = tanh(\mathbf{W}^{out}(\mathbf{x}(n+1), \mathbf{u}(n+1), \mathbf{y}(n), \mathbf{x}^2(n+1), \mathbf{u}^2(n+1), \mathbf{y}^2(n))) \tag{5.27}$$

5.7 Optimisation

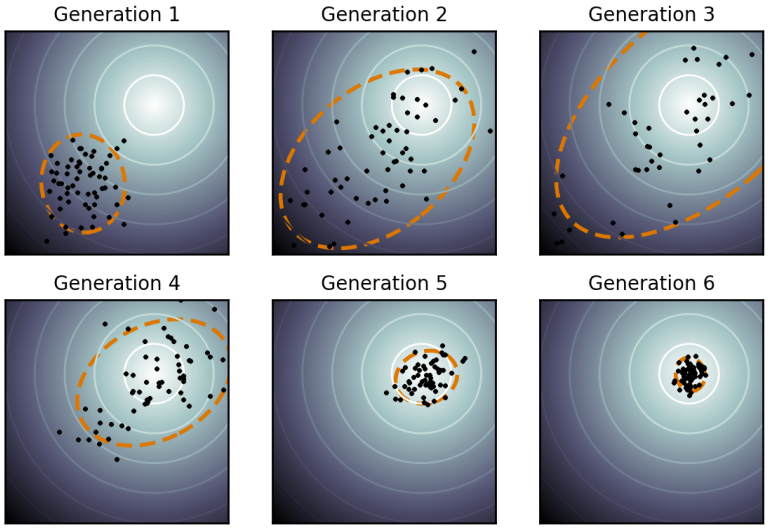
In this section we discuss practical issues around the optimisation of the various global parameters of the recurrent neural network that where analysed in 5.5.3, 5.5.3 and 5.6.1. *Optimisation* refers to the goal of achieving a minimal training error (reducing the MSE for training and testing steps). Achieving a minimal test error is delegated to cross-validation schemes which need a method for minimizing the training error as a substep. Two main optimisation algorithms were used on this research, *Genetic Algorithms* (GA) and *Evolutionary Algorithm* (EA), having always better results with the latter as shown in Vargas et al., 2014, Lukoševičius et al., 2009 and Jiang et al., 2008. This section describes briefly the evolutionary algorithm.

5.7.1 CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a state-of-the-art evolutionary algorithm in continuous domain evolutionary computation, it was presented in Hansen et al., 1996, Hansen et al., 2001, Hansen et al., 2004 and Hansen, 2016. CMA-ES is a population based algorithm for black box optimisation, stochastic, derivative-free methods for numerical optimisation of *difficult* non-linear or non-convex continuous optimisation problems, and it has been proven to have extremely good performance with deep neural networks, recurrent neural networks and reservoir computing Loshchilov et al., 2016 Tanaka et al., 2016 Jiang et al., 2008. This EA is broadly based on the principle of biological evolution, namely the repeated interplay of variation (via recombination and mutation) and selection and it works on a similar way as GA Anderson, 2011 because it encodes possible solutions as genes. Each generation (iteration) new individuals (candidate solutions x) are generated by variation, usually in a *stochastic* way, of the current parental individuals. CMA-ES estimates parameters of a Gaussian distribution for a gene x such that the distribution is concentrated in a region with high values of $f(x)$ as showed in:

$$x_k^{(g+1)} \quad \sim \quad m^{(g)} + \sigma^{(g)}\mathcal{N}\left(0, C^{(g)}\right) \quad \text{for} \quad k = 1, ..., \lambda \tag{5.28}$$

where $x_k^{(g+1)} \in \mathbb{R}^n$ is the individual offspring from generation $g+1$, $\sim$ denotes the same distribution both sides of the equation, $m^{(g)} \in \mathbb{R}^n$ is the mean value of the search distribution at generation g, $\sigma^{(g)} \in \mathbb{R}_{>0}$ is the overall standard deviation at generation g, $\mathcal{N}\left(0, C^{(g)}\right)$ is a multi-variable normal distribution with zero mean and covariance matrix of the search distribution $C^{(g)} \in \mathbb{R}^{nxn}$ at generation g. The number of offspring is denoted by $\lambda \geq 2$. Figure 5.16 shows an illustration of an actual optimisation run with covariance matrix adaptation on a simple 2-dimensional problem, where the solid lines represent the equal $f-$values spherical optimisation landscape and the dotted lines represent the distribution of the population that changes through generations (optimisation). One of the main advantages of the CMA-ES algorithm is that it does not require a *tedious* parameter tuning for its usage. Finding good strategy parameters is considered as part of the algorithm design, and not part of its application. The default λ population size is comparatively small to allow for fast convergence. In the experiments section, for a task of identifying the black-box model of a multirotor Vargas et al., 2015b, considering an optimisation of the spectral radius ρ, reservoir size N and noise added $\nu(n)$ parameters (always making sure the ESP is fulfilled) it took approximately 2000 iterations to *evolve* and create an ESN with a drastic drop on the MSE for the testing samples. Now that the mathematical model and the training algorithm has been discussed, we can characterize ESN as RNN with randomly scattered internal interconnected topology with restriction of its eigenvalue (maximal singular) where the output



Source: *wikipedia.org*

Fig. 5.16.: CMA-ES optimisation run on a simple 2-dimensional problem.

weights are the only ones being trained. The algorithmic complexity is reduced in comparison with classical RNN, they are highly adaptable and of fast teaching. To train ESN, the user must choose a small number of parameters or optimised them using evolutionary strategies like CMA-ES. ESN do suffer of common RNN issues like data over fitting, lack of generalization and model stability.

5.8 Summary

This chapter introduced the ML techniques, description of the algorithms used and required additional tools so that the following sections tackle the problems stated in the introduction chapter.

Firstly, by identifying black-box models for quadrotor vehicles and secondly, with the development of a slung load position estimator that uses ML at its core.

6 System Identification of MRUAV

In this chapter, the machine learning methods discussed in Chapter 5 are applied on real world data stored during numerous flights with multirotor vehicles to identify system models and demonstrate that the methodologies work and are capable of converging with the non-linear dynamics of MRUAV. This section is based in Vargas et al., 2015b coming from techniques used in Vargas et al., 2014.

Dynamic models of quadrotors can be obtained through several techniques. Grey-box modelling involves measuring system properties and dynamic relationships through experimentation. In this way, it can be used to derive non-linear models, however obtaining such parameters can be difficult and expensive with the required level of accuracy and precision. This is just one example of system identification. *System identification* is a collection of techniques for extracting an accurate mathematical model of a dynamic system from experimental input-output data. This can range from parameter identification only (light-grey-box modelling) or to full parameter/structural identification of the non-linear mapping (known as black-box). When flying, a quadrotor is an underactuated non-linear dynamically unstable system (Das et al., 2008). With this point of view, the RNN black-box modelling approach was chosen because they are especially powerful when approximating fast changing dynamics. As showed in Sections **??** and **??** the kinematic and dynamic models, respectively, where presented. The *pilot* or pseudo-controls were discussed, this are the ones used to *fly* the vehicle manually inside the MAST Lab and also are the ones that when carefully driven, via controllers presented in Section 4.3, can fly the vehicle autonomously. Using the framework DronePilot, presented in Section 3.5, real world data can be saved and stored for further use. Also, we can ensure this data is properly timestamped and the communication rates are always respected, which is extremely important when using this data for the intended experiments of this chapter. Schematically figure 6.1 displays the data flow in our robot architecture. In such architecture, the *Ground Station* is in charge of gathering (Motion Capture) the position and attitude of a specific set of markers that are contained on the vehicle, it also reads an input device (Joystick) that is used to map the pilot inputs to the pseudo-controls of the vehicle, once that data is collected it is sent via a wireless network to the vehicle. In the *MRUAV* a companion computer is reading the data being sent from the ground station, also it gathers the current state of the vehicle (IMU) and then it computes pseudo-controls with a position controller for a desired flight mode selected by

the pilot via the joystick. The current flight modes include: manual flight, position hold and trajectory tracking mode. The entire flight data is saved on *csv* files on the companion computer. Such data flow was explained in more detail in Section 3.5. It is important to notice

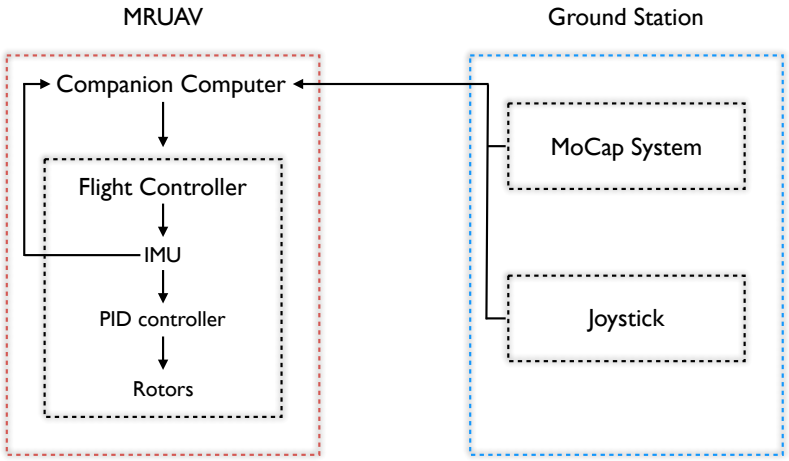


Fig. 6.1.: Data flow block diagram to control a MRUAV.

that several experiments were made just to know if the data being used was correct (with rich dynamics), if the algorithms worked, if the nets were converging, if the results were promising and if the output of the networks was making sense. Hundreds of hours of high performance CPU time were utilised before even attempting the first promising result. Regarding flight-data, approximately $533Mb$ of data was collected using DronePilot over the entire first stage of this research effort, this means around 96 hours of flight time have been performed with the MRUAV. Dozens of propellers (~ 60), 11 ESCs, 6 glass-fibre frames, 5 flight controllers, 3 companion computers and 2 motors were broken, on occasions, due to vehicle/code malfunction. In this chapter system models that are tuned using machine learning algorithms. The system model described in Section 6.1 uses real flight data and considers the pilot control as inputs $u(n)$ and as outputs $y(n)$ the vehicle pose.

6.1 Methodology

In this system identification experiment, the black-box model of a multirotor unmanned aerial vehicle is created using training data and three neural network approaches which will serve as universal dynamical system representation. Figure 6.2 shows the system block model of what it is intended to achieve, the inputs to the *black-box* are the pseudo-controls required to fly the vehicle either manual or with a automatic trajectory following, the outputs of the *black-box* will be the pose data which includes position (x, y, z) in a North-East-Down (NED) inertial reference frame and the orientation angles (ϕ, θ, ψ) in the body-fixed frame of the vehicle. The system is controlled via four inputs $[u_1, u_2, u_3, u_4]$, where u_1 is the

Fig. 6.2.: Pilot to Pose black-box model.

thrust along the z axis, u_2, u_3 are roll, pitch angle commands and u_4 is the yaw rate command. The pose data is from two sources, the MoCap system and the flight controller attitude complementary filter computation. The MoCap system provides the (x, y, z) inside the MAST Lab (Fig. 6.3), therefore only flights inside the laboratory are used. For orientation, the information is taken directly from the flight controller complementary filter (Section 2.7.2), it is also possible to use the orientation that the MoCap system delivers, but there could be data problems/corruption if the MoCap system loses track of the markers even for a few microseconds, causing a big error on the orientation measurements and this error would make more difficult the neural network training. Figure 6.3 show an example of a

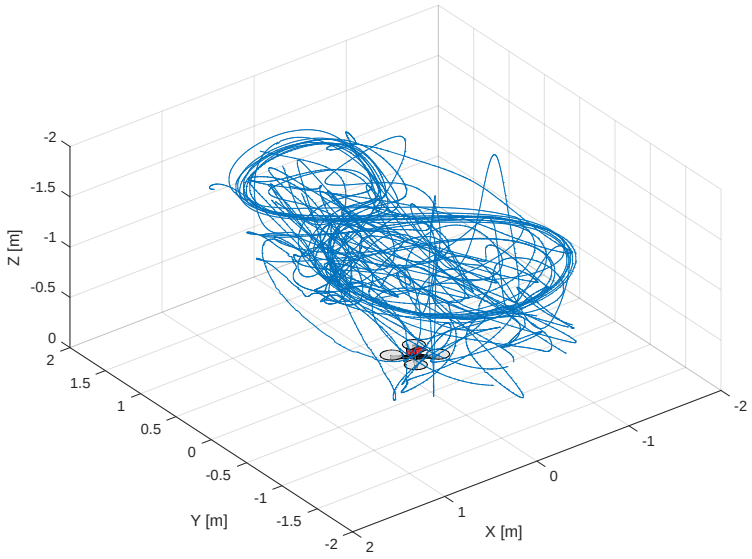


Fig. 6.3.: 3D trajectory plot of a training flight.

very rich flight test, that specific flight had a duration of approximately eight minutes (draining the battery until $\sim 10\%$). After dozens of hours of different flight styles/experiments, the best data results for training neural networks comes from trying to *excite* all modes of

the vehicle. Therefore the tests contain manual *slow* flights, manual *aggressive* flights, *step-responses* and automatic trajectory flights. In Fig. 6.3, the figure of eight trajectory (Section 4.4.3) can be appreciated along side all of the manual responses.

6.2 Data processing

With the help of the DronePilot framework explained in Section 3.5, we can save all data at a very precise time-rate, this is important when training neural networks from experimental data. The data is saved on *csv* files that then are preprocessed to check several factors such

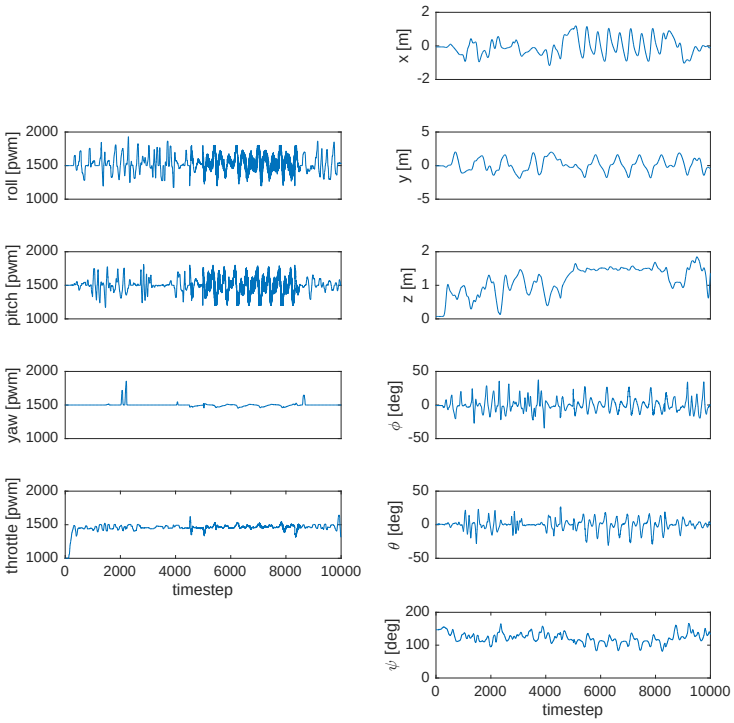


Fig. 6.4.: Example plot of inputs and outputs for the Pilot to Pose experiment.

as data health, figure 6.4 shows the inputs (left) and the outputs (right) half of a training flight. The data processing in this case, involves removing the data from where the vehicle is not flying and after it landed (disarming/arming process). Regarding data health, we need to make sure the reading are close to true values, as stated before, if the MoCap system is not properly calibrated, big errors or discrepancies will appear on the position of the vehicle. Another step that visually helps to check data consistency is making a 3D *replay* animation of the entire flight and compare it with real videos taken on the laboratory. In the MAST Lab when a vehicle is flown, either one or two cameras are recording the flight just for the purpose of checking the data health afterwards. Important to notice that in the

supervised learning method, two or more sets of *csv* files (flight experiments) are needed, one set is always used for training the neural network and an entire different set is used to run the validation of the training process (testing). In this particular experiment, we have several flights available for usage. Attempts on different flight modes for testing were used, this is just to prove that the *black-box* actually learned the dynamics of the system.

6.3 Training

For consistency on this test, two flights are used to train two different RNN approaches with three training methods. The RNN architectures used are the *Recurrent MLP* and *ESN*. The training methods include the common *Back-propagation Through Time* with the *BFGS* algorithm, *Real-Time Recurrent Learning* with the *LM* algorithm and the training method for the *Echo State Network*. All methods are described in Section 5.5.3. For simplicity of terminology we will call the networks by the main learning algorithm, that is *BPTT*, *RTRL* and *ESN*. In the first attempt the networks are trained with their own specific *standard* parameters, not *tuned*. Such parameters are different for each neural network architecture, they were found by running training experiments manually, in a trial-and-error method combined with the all of the literature mentioned on the Machine Learning Chapter 5. After getting *decent* results with more simplistic problems such as cosine generators (not presented in this thesis) it is proceeded to use them on the configuration for this specific task before attempting a optimisation. Figure 6.5 shows an overlay of the training pose output data of the three neural networks after the training process using the standards parameters. It can be observed that the topologies are roughly converging and tracking the real output. BPTT is the network that appears to be having the most problems to predict the correct output. 1000 time-steps are showed on the figure, if more data is showed is not easily distinguishable to the bare eye. In order to know if the networks are learning the task from experience (past data), a performance measurement is needed. MSE computation is used for this purpose. Table 6.1 shows the MSE of each network architecture for this first training process. It is very clear the ESN is outperforming BPTT and RTRL on this task, even

Network Architecture	**MSE**
BPTT	44.16
RTRL	13.54
ESN	0.905

Tab. 6.1.: Pilot to Pose training performance measurements.

with standard *not-optimised* parameters. This is of course the training data, this data is used in the algorithm to learn from it, therefore the error is always going to be lower than using testing data. Another important factor was the internal training time that the algorithms

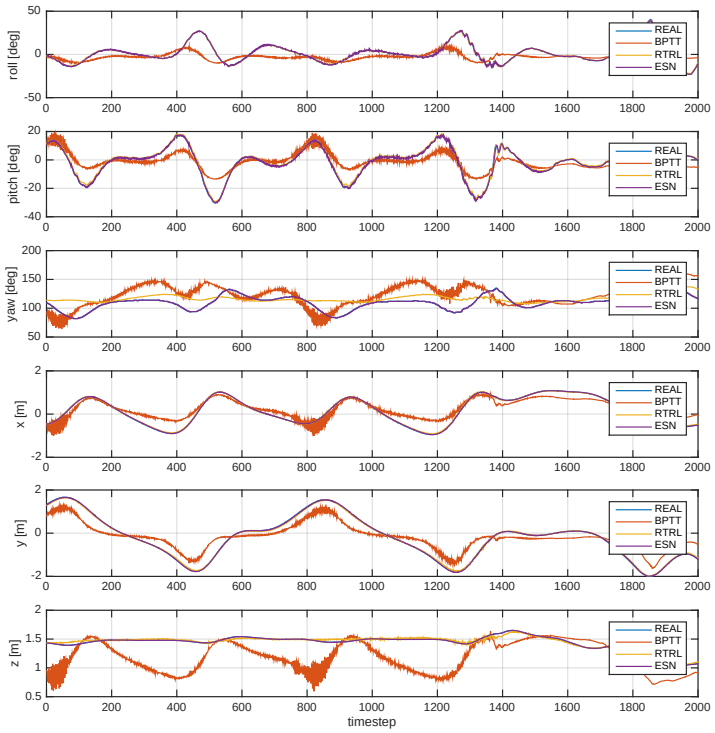


Fig. 6.5.: Pilot to Pose network output after training with standard parameters.

took to complete the learning task or iterations. On the BPTT it took around 585.79 seconds to complete 50 iterations with the *BFGS* algorithm while the RTRL the training time was 480.72 seconds to complete the same iterations, both networks had two internal layers of 5 neurons each, input-output delays but internal delays. The ESN took an impressive 3.21 seconds to train the network using 100 neurons (internal units).

6.4 Testing

Figure 6.6 shows an overlay of the testing pose output data of the three neural networks. This is done with activating (evaluating) the network output when inputting new data, such data we call it *test data* and its originated from an entire different flight, this is to prove the real performance of the networks. It can be appreciated (Fig. 6.6) that the networks are having more problems predicting the real output, this is an expected/common behaviour, but its the real application it is expected from this experiments, therefore it is important to optimise the network in such way that the MSE is reduced when attempting new data. Table 6.2 shows the errors calculated on the networks output with the test data. Even that in the three networks the error went bigger, the ESN is still outperforming the other two architectures. It is also appreciated that all architectures have problems with the heading

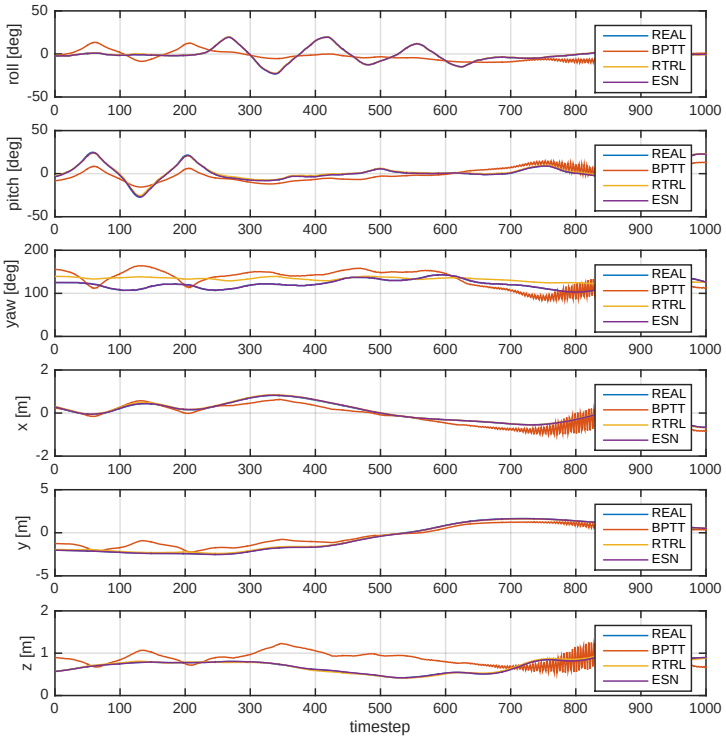


Fig. 6.6.: Pilot to Pose network output with *test* data.

Network Architecture	**MSE**
BPTT	45.74
RTRL	19.22
ESN	1.507

Tab. 6.2.: Pilot to Pose testing performance measurements.

output (yaw), this could indicate that there is no enough yaw data available to understand it. In figure 6.7 only one of the six outputs is showed, next to the instant error for each time-step, it is more easily appreciated the network behaviours. Overall, RTRL and ESN are *understanding* the task, but are having problems predicting the real output. BPTT is not converging into favourable trends using the standard parameters, it can be made better, but more training time is needed. One of the objectives of this experiment was to identify the best architecture that required the least optimisation and training time possible.

6.5 Optimising

After finding some rough level of convergence with the three different methodologies, an optimisation of the parameters must be made in order to make the testing MSE closer to

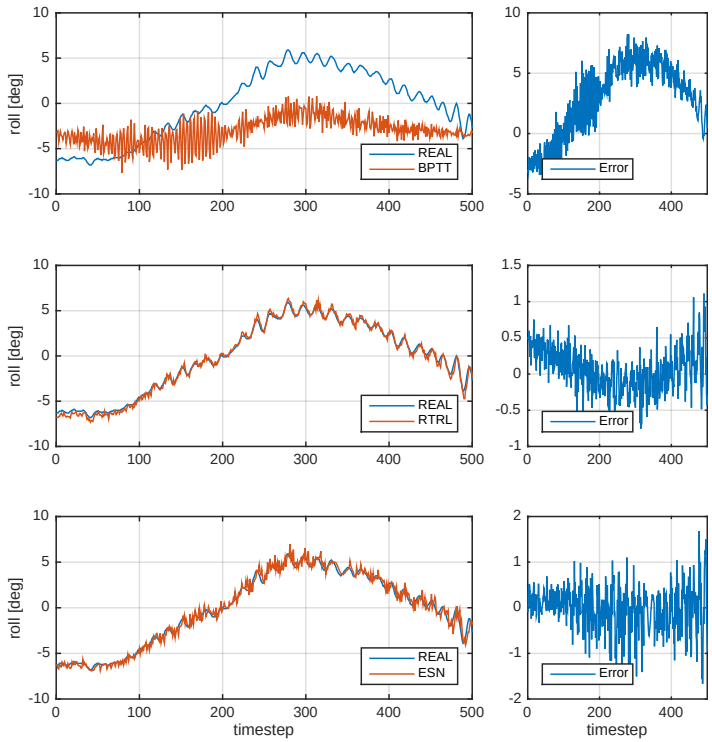


Fig. 6.7.: Pilot to Pose *zoomed* Roll network output .

zero therefore making the network output better. For BPTT and RTRL the process involved optimising the number of neurons, avoiding large numbers because the full-iteration time grows exponentially, also a big number of neurons does not insure a better performance, it can cause over-fitting and larger MSE on the testing data. CMA-ES was used to find the proper number of neurons, input-output delays was also increased, but still no internal delays. The optimisation took around 2 days on a *Intel(R) Xeon(R) CPU W3520 @ 2.67GHz*. For the ESN case, the optimised parameters were: #neurons (reservoir size), spectral radius, input scaling, output scaling, shift and noise added. The evolution algorithm was let to run about 2000 iterations, considering the time to train was much lower than the other two methods. On the training part, the MSE decreased an average of 94.41% for the three networks, figure 6.8 shows the network output after optimising the parameters. It can be appreciated in figure 6.8 that the three architectures are now converging with the dynamics of the vehicle, still the BPTT is the network that has more problems predicting the output while RTRL and ESN are *not-noticeable* alongside the real values. The MSE values are shown in table 6.3. It its noticeable that the network that was more beneficiated by the optimisation process was the RTRL. The optimisation process was done by creating a *fitness* function that received the parameters from the CMA-ES algorithm, created the network, train it and computed back a MSE. Then the CMA-ES is in charge of evolving such parameters until

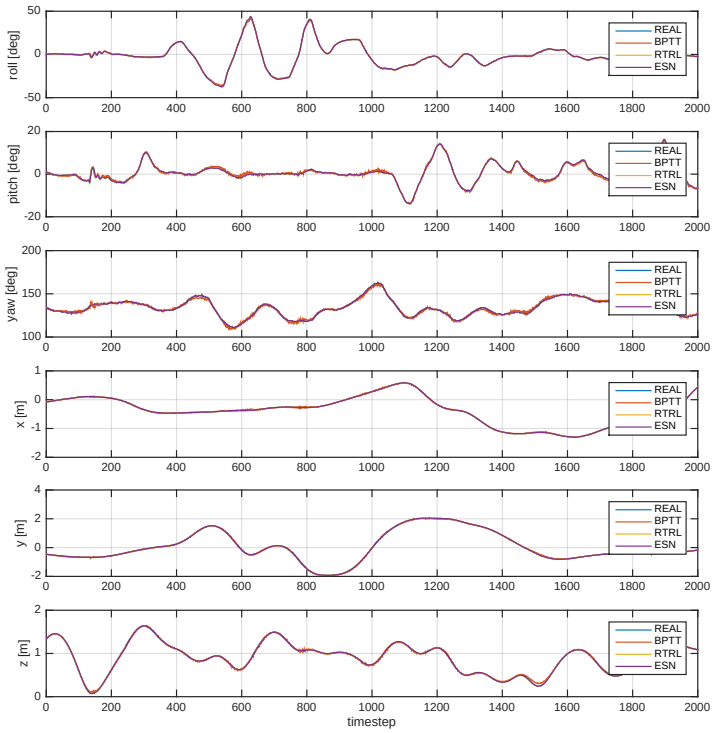


Fig. 6.8.: Pilot to Pose network output after training with optimised parameters.

Network Architecture	**MSE**
BPTT	0.7041
RTRL	0.0133
ESN	0.0127

Tab. 6.3.: Pilot to Pose training performance measurements after optimisation.

the MSE is decreased. Some tests were interrupted due to the fact that it was noticeable that it will never converge or it was lowest MSE possible with that set of data, but even so, the results showed in Tab. 6.3 proved that CMA-ES is a great candidate evolutionary algorithm to *tune* recurrent neural networks. Figure 6.9 shows the historic progression of the parameters evolution for 1000 iterations of the fitness function for the ESN network parameter optimisation. Figure 6.10 shows the network output predicting a new entire flight, which is the testing data. $5/6$ of the network outputs are very close to each other, therefore the networks are fully understanding the new data and the system dynamics. The heading output is not being predicted in a very good manner. After optimising parameters, the iteration and training times changed, in comparison with the experiments with standard parameters, this is due to a bigger number of neurons. Table 6.4 shows a comparison of training times with standard parameters against optimised. ESN remain the fastest method that converge in understanding the system dynamics of the MRUAV. These times does not

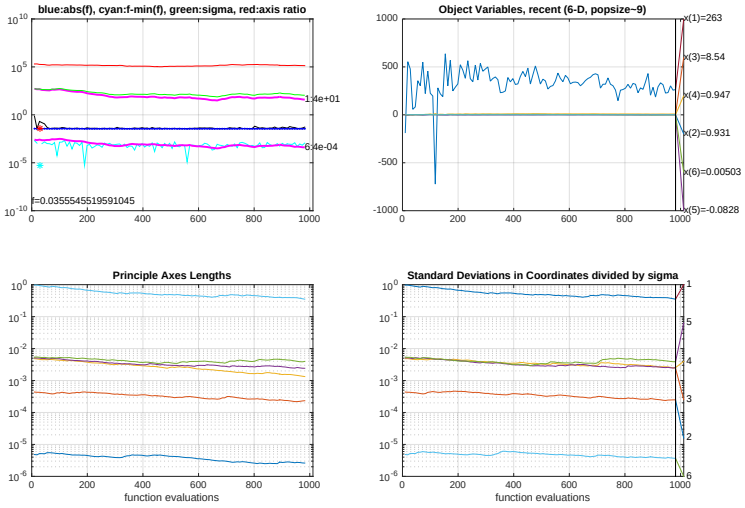


Fig. 6.9.: CMA-ES evolution process for optimising ESN parameters.

include the consumed time for optimisation of the parameters. The performance for the

Network Architecture	Standard Param [sec]	Optimised Param [sec]
BPTT	585.79	4450.11
RTRL	480.72	6086.85
ESN	3.21	18.29

Tab. 6.4.: Pilot to Pose training times comparison.

testing data is showed in table 6.5. Once optimised, ESN still got a better performance that the other two networks, making it a very good tool for system identification of multirotor vehicles. RTLR is extremely close to the ESN network performance, which also makes it a great tool addition for working with MRUAVs.

Network Architecture	MSE
BPTT	0.9023
RTRL	0.0164
ESN	0.0157

Tab. 6.5.: Pilot->Pose testing performance measurements after optimisation.

The zoomed individual network output is showed in figure 6.11. The difference between signals is almost imperceptible on the RTRL and ESN cases. The three proposed optimised networks are now following the trend.

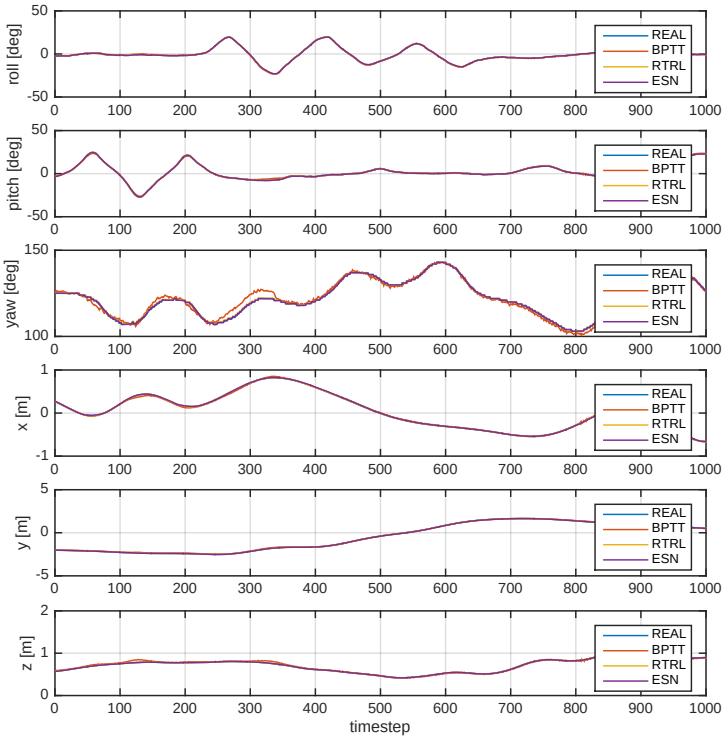


Fig. 6.10.: Pilot to Pose network output with *test* data and optimised parameters.

6.6 Results

In this chapter three network architectures are used with real flight data for performing black-box system identification. Using real flight input and output information, the RNN can identify the system dynamics and produce a black-box model that can be used for creating new state-of-the-art controllers, trajectory tracking algorithms or even optimise current way-point following position controllers. It can be stated that two of the three networks tested are ideal for system identification of MRUAV. Figure 6.12 shows four quadrotors following the same trajectory (offset in X and Y for discrimination), the blue path corresponds to the actual recorded data from the original flights (the testing data), while the other three trajectories correspond to the three proposed neural network architecture output to the pilot inputs of the same flight. If our classifier has succeeded, the trajectories must be equal. The MSE showed in table 6.5 shows that, although there exist errors between the real flight data and the neural network model output, the identified model using the ESN have an acceptable accuracy and can reflect the trend of the quadrotor. The improvement after the optimisation of the parameters on the networks output is clearly visible, therefore we can state that the three proposed architectures understood the full non-linear dynamics of the quadrotor MRUAV. The evolutionary strategy CMA-ES helps us improve the parameters of

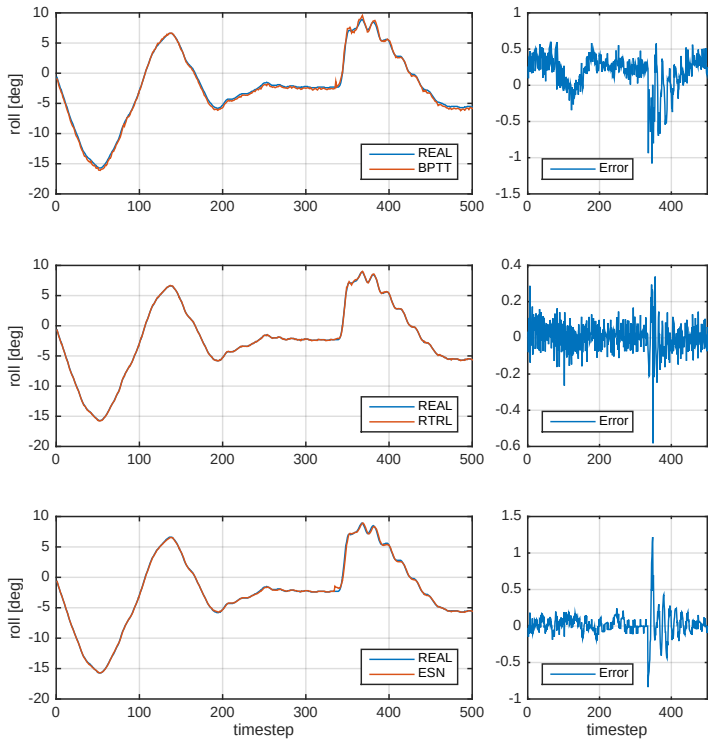


Fig. 6.11.: Pilot to Pose *zoomed* Roll optimised network output .

the ESN, decreasing the error by 99.2%, and it also highlights that the optimal spectral radius for our application must be greater than stated at the beginning of the research. The task of identifying a quadrotor MRUAV therefore requires a longer memory of the input when using echo state networks. With the results showed in table 6.5 we can distinguish that RTRL and ESN are the network architectures that performed the best even when not being optimised, therefore used in a real flight test. In such test the neural network was adapted on the DronePilot thread structure to predict the vehicle pose alongside the real information provided by the motion capture system, this is the final performance test of the proposed neural network architectures. Due to the processing power of the companion computer, only one architecture can be tested at a time, this is due to prevent an excessive time increase on the iteration loop that could lead to sluggish behaviour of the vehicle. Figure 6.13 shows only the position output of the RTRL neural network, this flight test was a short one, close to one minute of flying, the flight was a combination of manual piloting and two laps of a circular trajectory. Is noticeable that for height, the difference is no bigger than 10 cm, which means we can potentially replace this height for the one supplied by the motion capture system. This is potentially a solution for short indoors *GPS-denied* flights because the vehicle can predict, with a margin of error, its own position from current flight data. The overall MSE for the outputs on this test (Fig. 6.13) is 0.0245. The ESN flight test

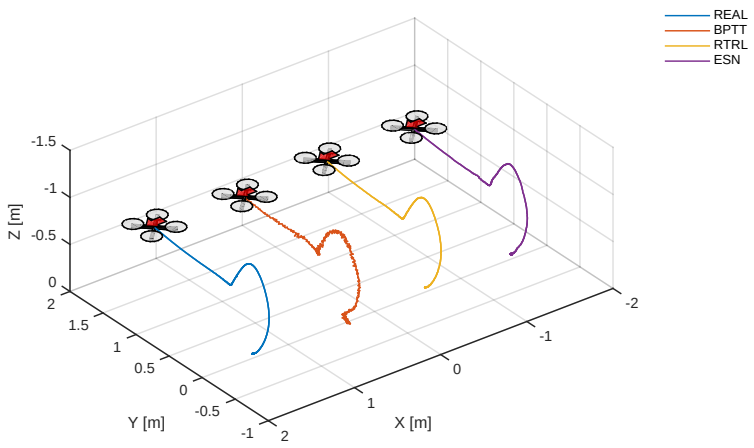


Fig. 6.12.: Pilot to Pose Trajectory comparison.

is shown in figure 6.14, as on previous tests, the ESN performs better, with a smaller MSE, being 0.0101 on this flight test, looking closely to the plot (6.14) on the 4200 time-step for the X output, 2800 time-step for the Y output and on 4100 time-step for the Z axis there is a sizeable *disturbance* on the prediction, which can potentially be a random state coming from learned from data that contained motion capture training errors, therefore reducing the reliability of this network architecture in this specific task. The flight test was roughly one minute length, similar to the RTRL. The experiment results indicate that recurrent neural networks provide generalization capabilities and are able to learn the dynamics of a MRUAV with excellent accuracy. More importantly, it was demonstrated that the learned dynamics can be used effectively on-board of the system, inside the Flight Stack.

6.7 Summary

With the results shown on this chapter it can be stated that the necessary machine learning tools are sufficiently mature and viable to continue using them on more complex generalization tasks, like a slung load position estimator. In the case of system identification of MRUAV the ESN approach showed better system modelling capabilities. On the next chapter, the techniques learned and applied on tasks such as system identification of MRUAV will be applied to the slung-load MRUAV coupled dynamics to be able to estimate and control a load.

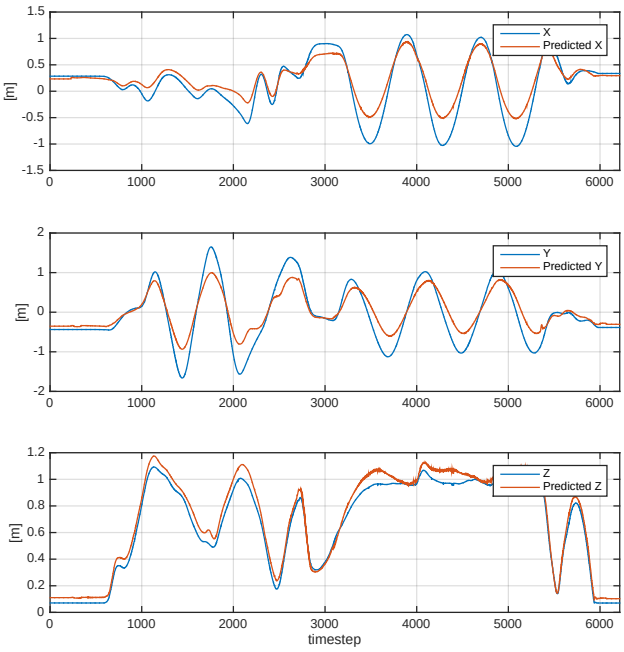


Fig. 6.13.: Pilot to Pose RTRL on-board flight test.

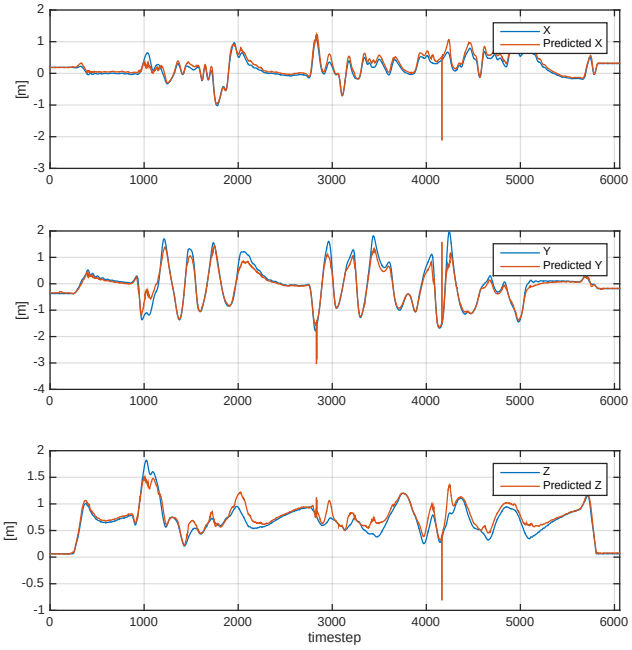


Fig. 6.14.: Pilot to Pose ESN on-board flight test.

7

MRUAV carrying a Slung Load

This chapter describes the analysis of the slung load dynamics when coupled with the quadrotor dynamics. Firstly, an introductory section details the behavior of a slung load using a trivial analogy of a simple pendulum, which was used to obtain the minimum cable length that the system is able to stabilise. Secondly, the development of the system model is presented based on Sreenath et al., 2013. Two methodologies to estimate slung load position are then described and analysed. The first method uses a vision system, while the second method uses machine learning techniques to tune an unstructured (black-box) estimator. Both methods are tested experimentally. For the second method, experiments are conducted to generate input/output data that is used to tune the estimator, three neural network techniques are then tested and verified by comparing results. The last part of the chapter introduces the controller. The objective of the controller is to dampen the oscillation of the load while the vehicle is attempting to track a trajectory. The suitability and accuracy of the controller are tested experimentally and results are presented at the end of the chapter. Sections of this chapter are developments of the work presented on Vargas et al., 2014, Vargas et al., 2015a and Vargas et al., 2017.

Fig. 7.1.: Quadrotor carrying a slung load.

Flying with a suspended load, also known as *slung load* or *sling load* is a very challenging task as the suspended load significantly alters the flight characteristics of the MRUAV (Fig. 7.1). Dynamically, attaching a load via a cable to the underside of the aircraft alters the mass distribution of the combined "*airborne entity*" in a highly dynamic fashion. The load will be subject to inertial, gravitational and unsteady aerodynamic forces which are transmitted to the multirotor via the cable, providing another source of external force to the MRUAV platform and thus altering the flight dynamic response characteristics of the vehicle. Similarly the load relies on the forces transmitted by the multirotor to alter it's state, i.e. we have moved from a single to two-body system, which is much more difficult to control. Aggressive trajectories and maneuvers can be disastrous if not performed correctly, therefore, an effective GNC architecture must maintain control and stability of the aircraft at all times. At the end of the transport motion, the slung load naturally continues to swing. Suppression of residual oscillations has been a topic of research for many years as shown in Faille et al., 1995, Frost et al., 2000, Zameroski et al., 2008, and Starr et al., 2005. Several model-based and non-model-based approaches have been proposed (as shown in Sec.1.2.3) in order to reduce the swing angle of the load when a rotorcraft follows a desired trajectory or comes to a stop (El-Ferik et al., 2013 and Omar, 2009). Some include an actuated suspension point or some other form of active load stabilization as in Smith et al., 1973. Another approach produces exciting transient oscillations in the system such that it reduces the oscillations induced by the load using input shaping methods; this has been used successfully in practice (Singer et al., 1997, Khalid et al., 2006). Using a neuro-predictive trajectory generation architecture (De La Torre et al., 2013b), it was shown that the effect of system uncertainty could be mitigated by the use of neural networks. A fuzzy logic method adds additional displacements to the helicopter trajectory in the longitudinal and lateral directions in order to suppress the swing of the suspended load (Omar, 2009).

7.1 Introduction

The suspended load can be initially considered as a basic pendulum, that is, a mass at the end of a string that swings back and forth, it consists of a mass m hanging from a string of length L and fixed at a pivot point P (Fig. 7.2). When displaced to an initial angle θ and released, the pendulum will swing back and forth with periodic motion. When applying Newton's Second Law for rotational systems (Eq. 7.1) (where the rate of change of the angular momentum is proportional to the net torque), the equation of motion for the pendulum can be obtained as showed on equation 7.2 and rearranged on its differential equation form in 7.3.

$$\tau = \mathbf{I}\alpha \tag{7.1}$$

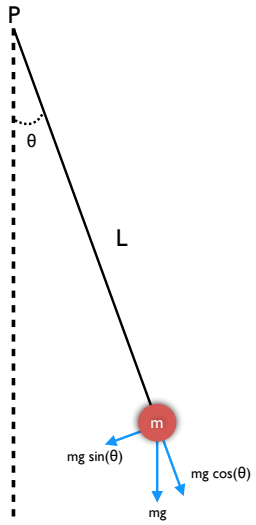


Fig. 7.2.: Free body diagram for a basic pendulum.

$$mL^2\frac{d^2\theta}{dt^2} = -mg\sin\theta L \tag{7.2}$$

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin\theta = 0 \tag{7.3}$$

7.1.1 Small angle approximation

If the amplitude of angular displacement is small enough that the small angle approximation holds true, then the equation of motion reduces to the equation of simple harmonic motion:

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\theta = 0 \tag{7.4}$$

And the harmonic solution is showed on equation 7.5 where the natural frequency of the motion is given by equation 7.6.

$$\theta(t) = \theta_0\cos\omega t + \phi \tag{7.5}$$

$$\omega = \sqrt{\frac{g}{L}} \tag{7.6}$$

With the assumption of small angles, the frequency and period of the pendulum are independent of the initial angular displacement amplitude. All simple pendulums should have the same period regardless of their initial angle and regardless of their mass m. The period for a simple pendulum does not depend on the mass m or the initial angular displacement

θ, but depends only on the length L of the string and the value of the gravitational field strength g, according to:

$$\mathbf{T} = 2\pi\sqrt{\frac{L}{g}} \tag{7.7}$$

When the angular displacement amplitude of the pendulum is large enough that the small angle approximation no longer holds, then the equation of motion must remain in its non-linear form showed on Eq. 7.3 which is usually solved using numerical methods.

7.2 Model of Slung Load Quadrotor System

The quadrotor with a slung load can be considered as a multi-body dynamical system with eight degrees of freedom and four degrees under-actuation as showed in Sreenath et al., 2013. Such system consists of two rigid bodies connected by a *mass-less* straight-line links which support only forces along the link. Figure 7.3 shows a quadrotor carrying a slung load following a particular trajectory. The system is characterized by the mass and inertia

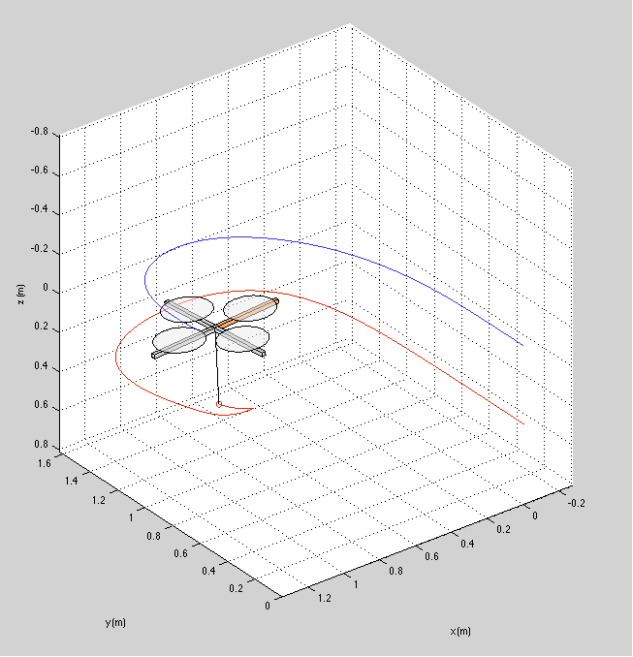


Fig. 7.3.: Quadrotor carrying a slung load.

parameters of the rigid bodies, and the suspension attachment point location (quadrotor centre of mass), some assumptions must be made in order to simplify the suspended load system, but sufficient for realistic representation.

- Both bodies are assumed to be rigid. Quadrotor rigidity is assumed as mentioned in Sec. 4.2 and for the slung load it is excluding non-rigid load (e.g. liquid tanks and flexible loads).

- Tether cable considered inelastic.
- The mass of the tether and aerodynamic effects on the load are neglected.

A load with position $\mathbf{r}_L$ may be described with respect to the quadrotor position $\mathbf{r}_Q$ by

$$\mathbf{r}_L = \mathbf{r}_Q + L\mathbf{q} \tag{7.8}$$

where $\mathbf{q}$ is the unit direction vector of the load, relative to the quadrotor position, and L is the length of the tether.

7.2.1 Quadrotor Attitude

The quadrotor attitude dynamics can be described simply by

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_\eta \boldsymbol{\omega}_Q \tag{7.9}$$

$$\dot{\boldsymbol{\omega}}_Q = \mathbf{I}^{-1}\left(\mathbf{M}_Q - \boldsymbol{\omega}_Q \times \mathbf{I}\boldsymbol{\omega}_Q\right) \tag{7.10}$$

7.2.2 Slung Load Attitude

The attitude dynamics of the combined quadrotor-load system are derived from Newton-Euler formalism. The centre of mass of the system exists at a point along the tether at distance r_0 from the quadrotor centre of mass. The load lies a distance L along the tether from the quadrotor centre of mass. The system thus has moment of inertia

$$I = m_Q r_0^2 + m_Q(L - r_0)^2 \tag{7.11}$$

The angular momentum of the system is given by

$$\begin{aligned} \mathbf{L} &= I\boldsymbol{\omega}_L \\ &= \left(m_Q r_0^2 + m_Q(L - r_0)^2\right)\boldsymbol{\omega}_L \end{aligned} \tag{7.12}$$

The moment is then obtained by differentiating the angular momentum, giving

$$\begin{aligned} \mathbf{M} &= \left(m_Q r_0^2 + m_Q(L - r_0)^2\right)\dot{\boldsymbol{\omega}}_L \\ &= \left((m_Q + m_L) r_0^2 + m_L L^2 - 2m_L L r_0\right)\dot{\boldsymbol{\omega}}_L \end{aligned} \tag{7.13}$$

If the quadrotor has generic specific force $\mathbf{F}_Q$ and the load $\mathbf{F}_L$, the net moment acting on the system is the sum of the moments caused by these forces and the gravitational force of each mass, which is

$$\begin{aligned}\mathbf{M} &= (L - r_0)\mathbf{q} \times (\mathbf{F}_L + m_L\mathbf{g}) - r_0\mathbf{q} \times (\mathbf{F}_Q + m_Q\mathbf{g}) \\ &= (L - r_0)\mathbf{q} \times \mathbf{F}_L - r_0\mathbf{q} \times \mathbf{F}_Q \\ &\quad + (L - r_0)\mathbf{q} \times m_L\mathbf{g} - r_0\mathbf{q} \times m_Q\mathbf{g}\end{aligned} \tag{7.14}$$

Note that the equation of motion for the unforced system is

$$0 = (L - r_0)\mathbf{q} \times m_L\mathbf{g} - r_0\mathbf{q} \times m_Q\mathbf{g} \tag{7.15}$$

which provides the solution for the position of the centre of mass

$$r_0 = \frac{Lm_L}{m_Q + m_L} \tag{7.16}$$

Equating Equations (7.13) and (7.14) gives

$$L\mathbf{q} \times \mathbf{F}_L - r_0\mathbf{q} \times (\mathbf{F}_Q + \mathbf{F}_L) = \left((m_Q + m_L)r_0^2 + m_L L^2 - 2m_L L r_0\right)\dot{\boldsymbol{\omega}}_L \tag{7.17}$$

and substituting Equations (7.15) and (7.16) gives

$$\begin{aligned} & L\mathbf{q} \times \mathbf{F}_L - \frac{Lm_L}{m_Q + m_L}\mathbf{q} \times (\mathbf{F}_Q + \mathbf{F}_L) \\ & \qquad = \left((m_Q + m_L)\frac{L^2 m_L^2}{(m_Q + m_L)^2} + m_L L^2 - \frac{2L^2 m_L^2}{m_Q + m_L}\right)\dot{\boldsymbol{\omega}}_L \\ \Rightarrow \quad & (m_Q + m_L)\mathbf{q} \times \mathbf{F}_L - m_L\mathbf{q} \times (\mathbf{F}_Q + \mathbf{F}_L) \\ & \qquad = \left(m_L L(m_Q + m_L) - Lm_L^2\right)\dot{\boldsymbol{\omega}}_L \\ \Rightarrow \quad & m_Q\mathbf{q} \times (m_Q\mathbf{F}_L - m_L\mathbf{F}_Q) = Lm_Q m_L\dot{\boldsymbol{\omega}}_L \end{aligned}$$

which provides the solution

$$\dot{\boldsymbol{\omega}} = \frac{1}{Lm_Q m_L}\left(\mathbf{q} \times (m_Q\mathbf{F}_L - m_L\mathbf{F}_Q)\right) \tag{7.18}$$

The evolution of $\mathbf{q}$ is easily found to be

$$\dot{\mathbf{q}} = \boldsymbol{\omega} \times \mathbf{q} \tag{7.19}$$

The position dynamics of both the quadrotor and slung load are strongly coupled. From Newton-Euler formalism, the net linear momentum of the system is given by

$$\mathbf{p} = m_Q \dot{\mathbf{r}}_Q + m_L \dot{\mathbf{r}}_L \tag{7.20}$$

The net momentum may expressed purely in terms of the load velocity by considering Equation 7.8, giving

$$\begin{aligned} \mathbf{p} &= m_Q(\dot{\mathbf{r}}_L - L\dot{\mathbf{q}}) + m_L \dot{\mathbf{r}}_L \\ &= (m_Q + m_L)\dot{\mathbf{r}}_L - m_Q L \dot{\mathbf{q}} \end{aligned} \tag{7.21}$$

Substituting Equation (7.19) then gives

$$\mathbf{p} = (m_Q + m_L)\dot{\mathbf{r}}_L - m_Q L(\boldsymbol{\omega} \times \mathbf{q}) \tag{7.22}$$

The net force acting on the system is then found from the derivative of the linear momentum

$$\mathbf{F} = (m_Q + m_L)\ddot{\mathbf{r}}_L - m_Q L(\dot{\boldsymbol{\omega}} \times \mathbf{q} + \boldsymbol{\omega} \times \dot{\mathbf{q}}) \tag{7.23}$$

Substituting (7.18) and Equations (7.19) and expanding gives

$$\begin{aligned} \mathbf{F} &= (m_Q + m_L)\ddot{\mathbf{r}}_L + \frac{1}{m_L}\left((\mathbf{q} \cdot (m_Q \mathbf{F}_L - m_L \mathbf{F}_Q))\mathbf{q} + m_L \mathbf{F}_Q - m_Q \mathbf{F}_L\right) \\ &\quad - m_Q L\left((\boldsymbol{\omega} \cdot \mathbf{q})\boldsymbol{\omega} - (\boldsymbol{\omega} \cdot \boldsymbol{\omega})\mathbf{q}\right) \\ &= (m_Q + m_L)\ddot{\mathbf{r}}_L + \mathbf{F}_Q - \frac{m_Q}{m_L}\mathbf{F}_L - m_Q L(\boldsymbol{\omega} \cdot \mathbf{q})\boldsymbol{\omega} \\ &\quad + \left(\frac{1}{m_L}(\mathbf{q} \cdot m_Q \mathbf{F}_L - \mathbf{q} \cdot m_L \mathbf{F}_Q) + m_Q L \boldsymbol{\omega} \cdot \boldsymbol{\omega}\right)\mathbf{q} \end{aligned} \tag{7.24}$$

The force vector is the sum of all forces acting on both bodies, described generally as

$$\mathbf{F} = \mathbf{F}_Q + \mathbf{F}_L + m_Q \mathbf{g} + m_L \mathbf{g} \tag{7.25}$$

Equating the forces described by Equations (7.25) and (7.24) gives

$$\begin{aligned} & \mathbf{F}_Q + \mathbf{F}_L + (m_Q + m_L)\mathbf{g} = (m_Q + m_L)\ddot{\mathbf{r}}_L + \mathbf{F}_Q - \frac{m_Q}{m_L}\mathbf{F}_L - m_Q L(\boldsymbol{\omega} \cdot \mathbf{q})\boldsymbol{\omega} \\ & \qquad + \left(\frac{1}{m_L}(\mathbf{q} \cdot m_Q \mathbf{F}_L - \mathbf{q} \cdot m_L \mathbf{F}_Q) + m_Q L \boldsymbol{\omega} \cdot \boldsymbol{\omega}\right)\mathbf{q} \\ \Rightarrow \quad & (m_Q + m_L)(\ddot{\mathbf{r}}_L - \mathbf{g}) = \left(1 + \frac{m_Q}{m_L}\right)\mathbf{F}_L + m_Q L(\boldsymbol{\omega} \cdot \mathbf{q})\boldsymbol{\omega} \\ & \qquad - \left(\frac{1}{m_L}(\mathbf{q} \cdot m_Q \mathbf{F}_L - \mathbf{q} \cdot m_L \mathbf{F}_Q) + m_Q L \boldsymbol{\omega} \cdot \boldsymbol{\omega}\right)\mathbf{q} \end{aligned} \tag{7.26}$$

The translational dynamics of the system may then be described in terms of the load acceleration by

$$\ddot{\mathbf{r}}_L = \mathbf{g} + \frac{1}{m_Q + m_L}\left[\left(1 + \frac{m_Q}{m_L}\right)\mathbf{F}_L + m_Q L(\boldsymbol{\omega}\cdot\mathbf{q})\boldsymbol{\omega} - \left(\frac{1}{m_L}(\mathbf{q}\cdot m_Q\mathbf{F}_L - \mathbf{q}\cdot m_L\mathbf{F}_Q) + m_Q L\boldsymbol{\omega}\cdot\boldsymbol{\omega}\right)\mathbf{q}\right] \tag{7.27}$$

7.3 Slung Load Position Estimation

The strongly coupled, non-linear dynamics of the *Slung-Load/Quadrotor* system makes it extremely complicated to estimate the position of the load experimentally and for anti-swing control purposes. On previous experimental research work with a *Slung-Load/Quadrotor* system Palunko et al., 2012 Tang et al., 2015 Mellinger et al., 2014 Sreenath et al., 2013, the position of the suspended load is estimated using motion capture system such as the one used in the MAST Lab (Sec. 3.2). This approach will only work inside a laboratory. As seen in the literature review Chapter 1, there is several alternatives to estimate the position of the load, in this section two methods will be presented, firstly an optical approach using computer vision approach and secondly a machine learning prediction one.

7.3.1 Computer Vision Estimation

Computer vision (CV) is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. It helps in tasks that include methods for acquiring, processing, analysing and understanding digital images, and in general, deal with the extraction of high-dimensional data from the real world in order to produce numerical or symbolic information. The CV estimator uses a vision based system as the only sensor input and therefore it does not require any mounting of sensors on the load. The CV system uses images from a downwards looking camera to calculate a position vector of the load in the MRUAV fixed frame pointing from the camera to the load. This makes it ideal for augmenting an already autonomous MRUAV with slung load capabilities. This estimation system requires a slightly bigger MRUAV platform (Fig. 7.4) that the one presented in Section 3.5, in order to be able to carry a gimballed camera sensor.

Sensor system

The vision system is a camera mounted on the MRUAV frame looking down on the load. A two-axis gimbal was designed to keep sensor system downwards. If the camera is only fixed to the vehicle, there can be a huge amount of *false detection* estimations of the position of

Fig. 7.4.: Test-beds size comparison, higher: CV platform, lower: Standard platform.

the slung load when the vehicle is *pitching* and *rolling* (drastic change in attitude). The gimbal is made with 3D printed parts and contains two rc-servos to compensate the pitch and roll angles of the MRUAV attitude when travelling, it can be seen in figure 7.5. With a gimbal, the camera will *always* be pointing down regardless the attitude of the vehicle. The rc-servos are commanded by the flight controller using the reversed attitude angle and transformed to PWM, which is the input signal to the rc-servo. One disadvantage of this sensor system is that requires a larger MRUAV (Fig. 7.5) to be able to carry and accommodate properly the extra equipment. Having a larger vehicle, approximately $450mm$ from rotor to rotor centre, will limit the trajectories that can be performed inside the MAST Lab due to the size of the aircraft, but for outdoors flying is a very good estimation option.

Computer vision algorithm

To easily identify the load amongst other objects that appear in the camera view the load has a specific colour; in this case the slung load is predominantly black while the ground is a light colour. Therefore a *colour tracking* algorithm is needed. The location of the *colour area* in the image is thus an estimate of the position of the load relative to the orientation of MRUAV. Several colour algorithms were created to achieve the goal of finding the slung load position, more detailed information of the algorithms can be seen in Appendix A.5. Such algorithms use the OpenCV (Open Source Computer Vision) library of programming

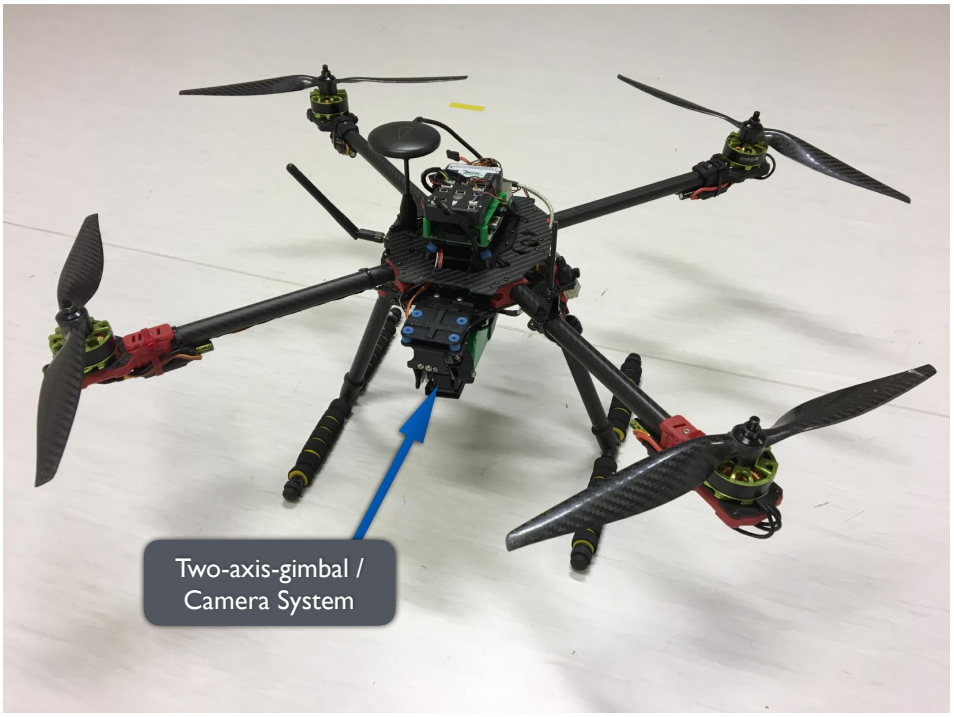


Fig. 7.5.: Test-bed v2 Quadrotor with gimbal/camera system mounted.

functions (Bradski, 2000). This library is mainly aimed at real-time computer vision. The algorithm steps are as follows:

1. Grab an image frame from the camera video stream
2. Convert that image from BGR to HSV format
3. Check if the converted HSV array elements lie between the elements of two other HSV arrays (colour selection is a range of black colour)
4. Apply two very common morphology operators, dilation and erosion
5. Find contours in the binary output image based on Suzuki et al., 1985, with a result similar to the image in Fig. 7.6
6. If the identified area is larger than the minimum area pre-established, calculate the centroid of the area and report back the pixel position

If the area found is to small it is assumed that the algorithm has made a false detection and the measurement is discarded. This could happen if the load is outside the field of view (FOV) of the camera. Note that the camera is fixed in the MRUAV which means that both load swing and the MRUAV roll and pitch can result in the load disappearing from FOV. When the vision algorithm has detected the pixel position of the load this measurement

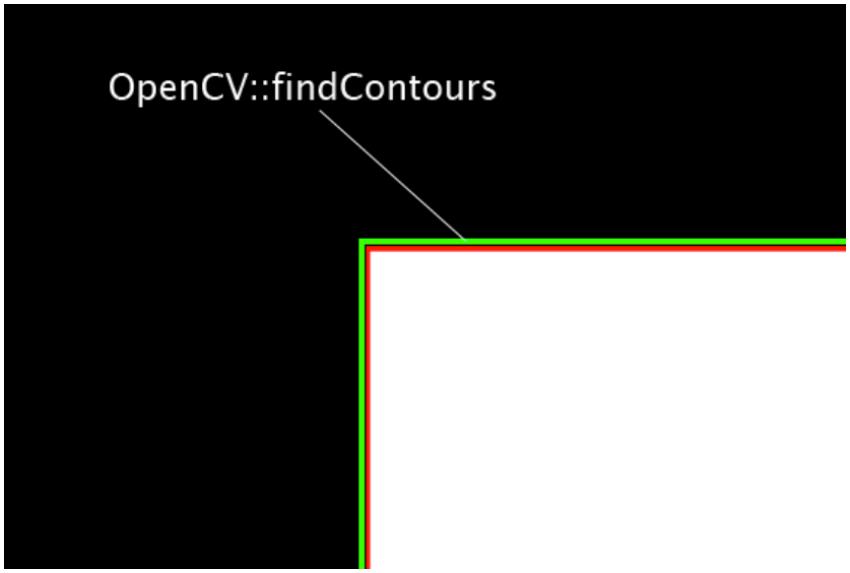


Fig. 7.6.: Graphical description of the *findContours* algorithm.

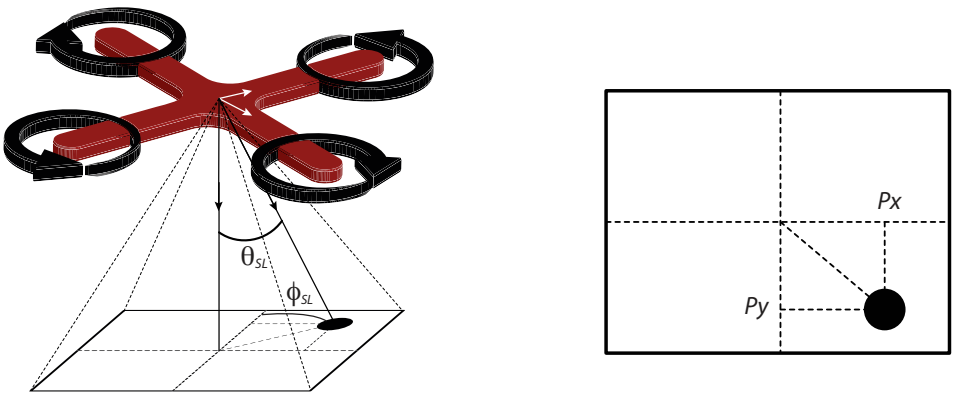


Fig. 7.7.: 3D spatial location perspective view (left) and 2D camera view (right).

must be mapped to a 2D slung load position. This is done by first transforming the pixel position to two angles θ_{SL} and ϕ_{SL} as shown in figure 7.7. The angles are calculated as:

$$\begin{aligned} \phi_{SL} &= K_{SL} P_x \\ \theta_{SL} &= K_{SL} P_y \end{aligned} \tag{7.28}$$

Where K_{sl} is the relationship between the field of view (FOV) of the camera and the number of diagonal pixels of the selected resolution (Eq. 7.29) and [P_x, P_y] are the reported pixel position of the centroid of the area colour target found.

$$K_{SL} = \frac{FOV}{D_{pix}} \tag{7.29}$$

Because the camera is gimballed and in the same position of the *CoG* of the vehicle a rotation is needed (Eq. 7.30) and the position of the camera it is considered the same as the vehicle (Eq. 7.31).

$$\mathcal{R}_C^E = \begin{bmatrix} \cos\psi_Q & \sin\psi_Q & 0 \\ -\sin\psi_Q & \cos\psi_Q & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.30}$$

$$\Gamma_C^E = \Gamma_Q^E \tag{7.31}$$

The position of the slung load can be calculated as:

$$\Gamma_{SL}^E = \Gamma_C^E + \mathcal{R}_C^E \Gamma_{SL}^C \tag{7.32}$$

Where the position of the slung load in regards with the camera view is given by Eq. 7.33 and the rotation of the camera against the earth frame by Eq. 7.30.

$$\Gamma_{SL}^C = \begin{bmatrix} L_{SL}\sin\phi_{SL} \\ L_{SL}\sin\theta_{SL} \\ L_{SL}\cos\phi_{SL}\cos\theta_{SL} \end{bmatrix} \tag{7.33}$$

The 2D slung load position calculated with Eq. 7.32 is used by a swing-free controller to prevent aggressive oscillations of the load. Such controller is described in section 7.4.

Implementation

The proposed algorithm must run on-board the companion computer, so that the swing-free controller can use the estimation of the position and then reduce the oscillations of the load, it is important to keep the delay in the vision system low. It is noticed that almost any kind of computer vision algorithm consumes a considerable amount of CPU time, therefore it must be designed to run on a multi-threaded structure so that the other tasks running in parallel with the CV algorithm do not suffer from performance. The DronePilot framework is therefore ideal for this implementation, the CV algorithm is then added as an extra thread that runs in parallel with the control thread (that flies the vehicle), the communications one and the rest of the add-on functions. Several colour tracking algorithm implementations were created and tested, the best results are achieved with the *Color-6* [1] algorithm, that is contained on the *rpi-opencv* open source repository. The rates can be seen in table 7.1. In figure 7.8 the result of the algorithm can be appreciated in a graphic manner with a rectangle being drawn surrounding the area found with the *findContours* function. On the

[1] `https://github.com/alduxvm/rpi-opencv/blob/master/color-6.py`

Device	Detection @ 640x480px [sec]	Detection @ 500x500px [sec]]
MBPR	0.005	0.003
RPI2	0.15	0.09
RPI3	0.12	0.05

Tab. 7.1.: Performance of the CV algorithm implementation using different CPUs.

real implementation this step is not needed (as no monitor is connected when flying) and it can reduce the load of CPU when removed. Even with the current (2016) most *powerful* on-

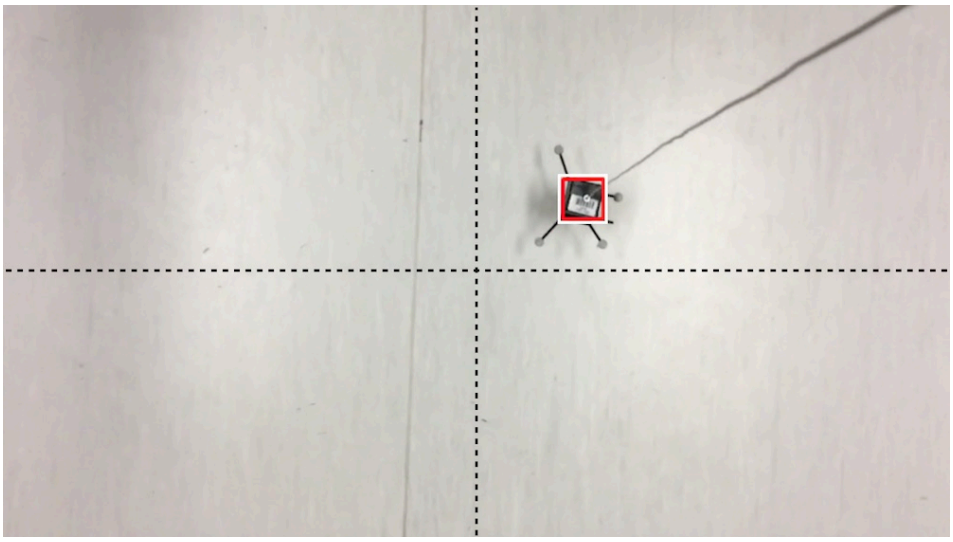


Fig. 7.8.: Frame of a positive colour area found, the slung load position can then be estimated.

board credit-card-size companion computer, the algorithm is extremely heavy for the CPU. Achieving $20Hz$ to get a new slung load position, which in some cases is not fast enough to compensate or attempt a control action to reduce the oscillations.

7.3.2 Machine Learning Estimation

Using similar methods that were tested and proved in Chapter 6, a slung load position prediction is created to be used later on a simple anti-swing slung load controller. Such technique will use data from experimental flights, where a *load* is attached to the vehicle. The *DronePilot* framework is used again in order to control the vehicle and log all of the experimental data. The data required are *pilot commands* and *vehicle pose* as inputs, while the outputs are only the *slung load position* as showed on Fig.7.9.

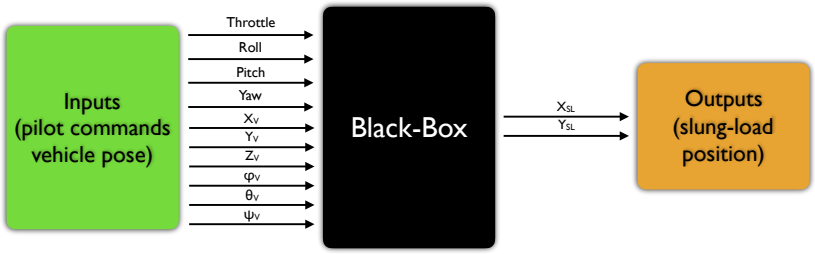


Fig. 7.9.: Slung-Load estimator black-box model.

Data Collection

Several flight modes were designed for making the test flights easier, repeatable and safe. As explained in the introduction of the section, flying with a slung load alters the dynamics of the vehicle, provokes oscillations and can easily cause accidents. Flying inside a confined space (MAST Lab) adds a challenge when gathering the data, due to the possibility of the load making contact with one of the walls, corrupting the entire flight data. A common flight test to gather the data for making the system identification of the *quadrotor/slung-load* system involves the following steps:

- Pre-takeoff security tests (propeller nuts securely tied, battery monitor attached, Mo-Cap markers on position, SSH login to companion computer, GroundStation sending information to the companion computer, among others)
- Takeoff the vehicle manually using joystick attached to the GroundStation
- Activate *position hold* flight mode for the vehicle to remain at a specified position at the centre of the MAST Lab
- Carefully approach the vehicle and connect the slung load to the bottom of the vehicle (ensuring it is securely attached)
- Activate *altitude hold* or *manual* flight mode and carefully move the vehicle around the flight area

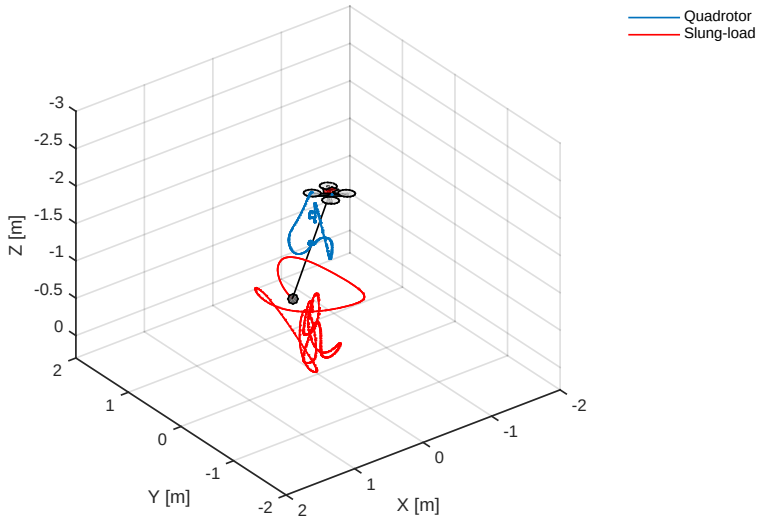


Fig. 7.10.: 3D trajectory plot from a training flight of the quadrotor slung-load system.

- When test is over, activate *position hold* flight mode and remove the slung load from the vehicle
- Land the vehicle manually
- Collect the data for further analysis

If during the flight test one of the next conditions occurs, then the entire test is rendered as invalid:

- Slung load detaching from vehicle
- Slung load tether changes form noticeably
- Slung load makes contact with wall or another object inside flight area, e.g. columns, floor
- Slung load or vehicle goes outside flight area
- Slung load attitude is perpendicular to any axis of the vehicle position (extreme slung load oscillation)

Video of the flight test must be recorded so that it can be used to check that the above conditions did not happened. Also, prior to using the data for the machine learning experiments, it must be plotted and analysed to check if the conditions above were not broken. Replay animations are also made in order to ease visualization of the quadrotor/slung-load system, as shown in figure 7.10 a 3D plot of the quadrotor/slung-load system trajectory is showed, only minimal time-steps are displayed to avoid saturation of the trajectory path. Figure 7.11 shows the exact frame of the above-mentioned videos where the slung load detaches from the vehicle due to extreme oscillations while performing a flight to gather data

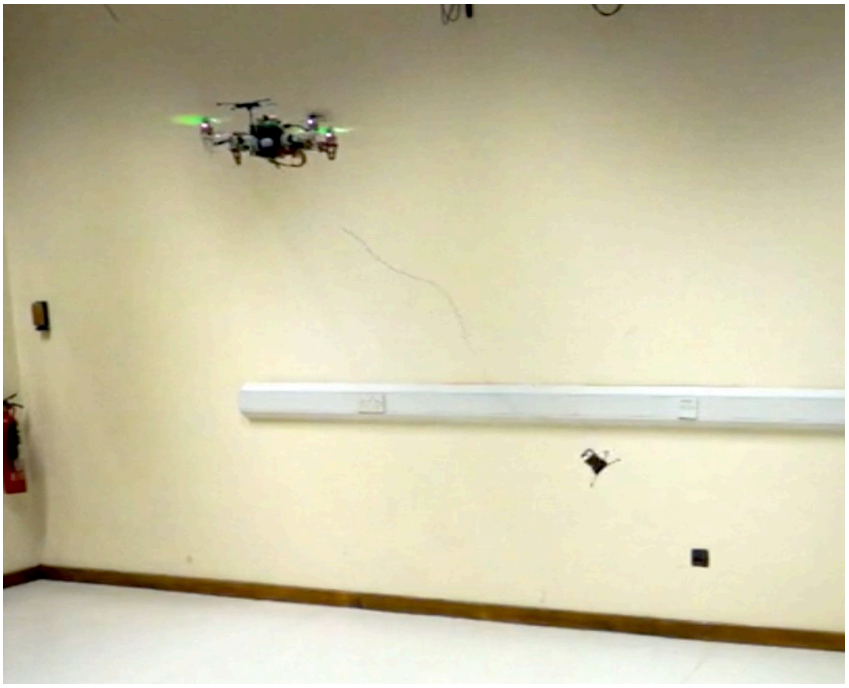
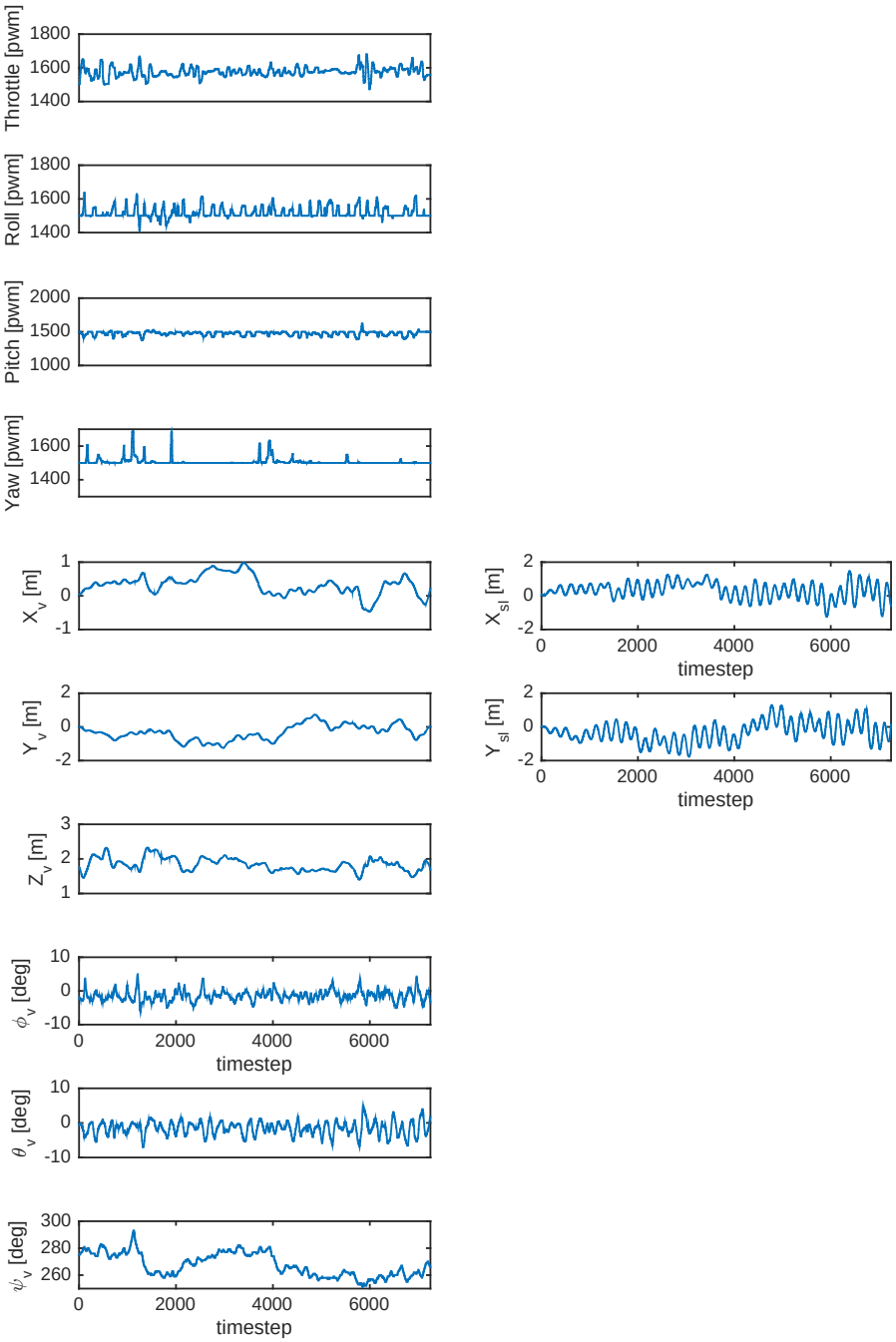


Fig. 7.11.: Video-frame of the slung load detach moment due to extreme oscillations.

for the machine learning experiments. Approximately $200MB$ of flight data was gathered during all of the slung load experiments. Due to the difficulty of flying while exciting the dynamics of the quadrotor/slung-load system, 7 out of 10 experimental flights resulted in a crash. Collecting the experimental data was one of the most challenging elements of this section.

Data Processing

With the help of the DronePilot framework, the flight test data is saved on-board the vehicle so that it can be preprocessed prior neural network training. Figure 7.12 shows the inputs on the left column and the outputs on the right column of a flight test. The data processing in this case, in comparison with the experiments in Sec. 6.2, involves removing all data from when the vehicle does not have a slung load attached to it. To achieve this purpose in an easier way, the black-box flight logs contain the current flight mode in which the vehicle is engaged, therefore the data under the *slung load flight mode* is the one that it will use for training the machine learning algorithms.



* Values only as representation

Fig. 7.12.: Example plot of inputs (left) and outputs (right) for the quadrotor/slung-load system.

Training and Testing

In the same manner as previous machine learning experiments in Section 6.3, three machine learning approaches are used. For naming simplicity we will refer to them using their training methodologies, that is, *Back-propagation Through Time* (BPTT), *Real-Time Recurrent Learning* (RTRL) and *Echo State Network* (ESN). In the first training process, a small number of training iterations are done until a good convergence of the dynamics is being produced by the neural networks. This process helps discard inadequate network configurations. Figure 7.13 shows the result of the first promising estimations from the machine

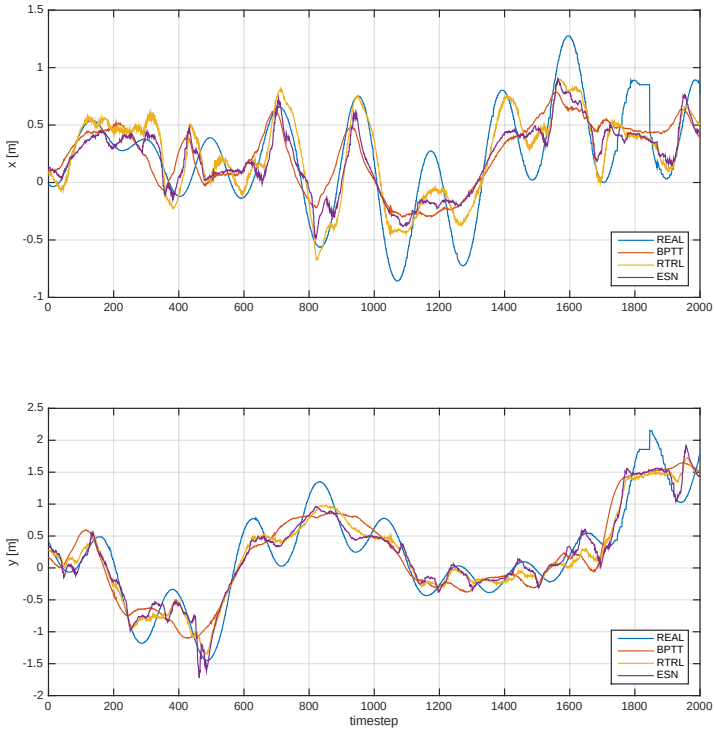


Fig. 7.13.: ML Slung load position estimation after training.

learning approaches. It is appreciated that the neural networks are slightly understanding the movements of the slung load, although oscillations are not being perceived properly and more training is needed to tackle this situation. To ensure the configuration of the neural networks is properly chosen, the same training is repeated with different datasets and checked for convergence similarities. Similarly, as seen in the system identification chapter, two different sets of data are used to ensure the neural networks are properly *understanding* the dynamics of the slung load, those are the training and the testing datasets, the first ones are used to train the NN and the latter for testing the NN with data that is different from the initial iterations, therefore ensuring a full approximation of the dynamical system

behaviour. The test results of the first estimations are displayed in figures 7.14 and 7.15. In these plots, the outputs of the neural networks are *separated* and *zoomed* to facilitate the analysis. The prediction error in testing data is always bigger than the one for training data, and it can easily be observed in the plots. At this stage, with no optimisation, the RTRL

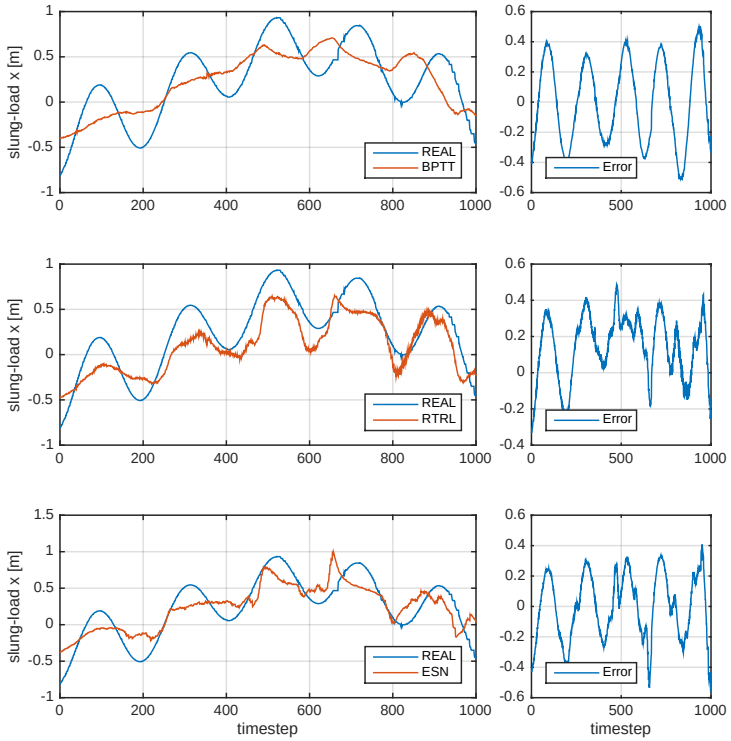


Fig. 7.14.: ML Slung load X-axis position estimation with testing data.

and ESN are performing better than the BPTT. The ESN is much faster to converge when training, while the RTRL is the winner, with a lower MSE as shown on table 7.2.

ML technique	Training time [sec]	Training MSE	Testing MSE
BPTT	404.4	0.077	0.089
RTRL	343.3	0.050	0.064
ESN	1.84	0.060	0.065

Tab. 7.2.: Performance of the ML slung load estimation.

Optimising

After having good convergence results of the neural networks using standard parameters, an optimisation is needed to get the best estimation possible so that later this position estimation can be used to control the oscillations of the load using the controller proposed

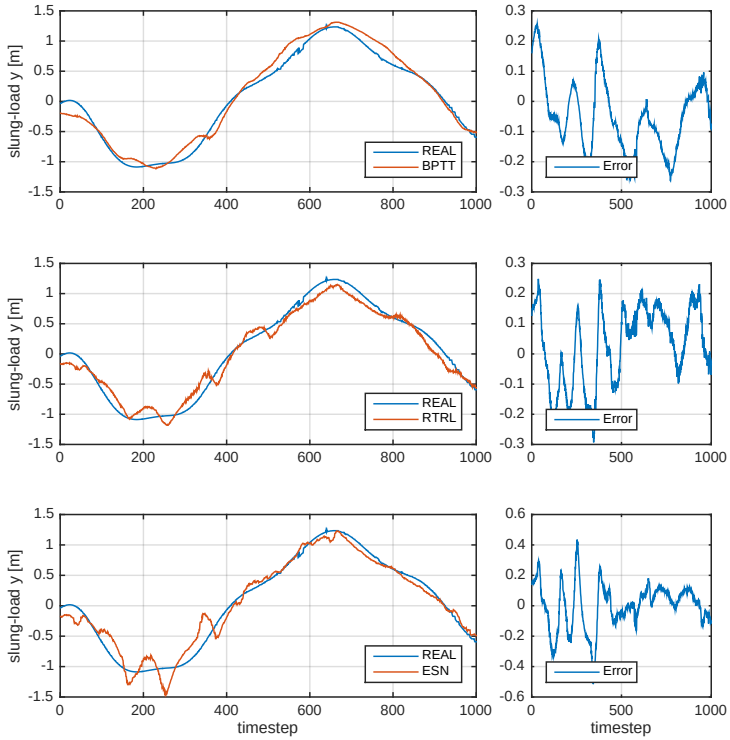


Fig. 7.15.: ML Slung load Y-axis position estimation with testing data.

in Sec. 7.4. In the slung load estimation application, it is noted that if the number of iterations is increased, the errors decrease until a certain point, above that point the errors will start increasing. The best results were found at 50 iterations (as shown on Tab. 7.3), the MSEs increased when doubling this number of iterations. One of the reasons behind this behaviour is over-fitting. Over-fitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been over-fit has poor predictive performance, as it overreacts to minor fluctuations in the training data. The neural network output when predicting test data can be seen in Fig. 7.16, the overall behaviour of the networks is greatly improved in comparison with the non-optimised networks. On previous experiments, RTRL and ESN constantly showed the best performance among the topologies, while in the slung load case RTRL and BPTT had the lowest MSEs. On figures 7.17 and 7.18, the machine learning estimation of the position of the slung load with test data is shown.

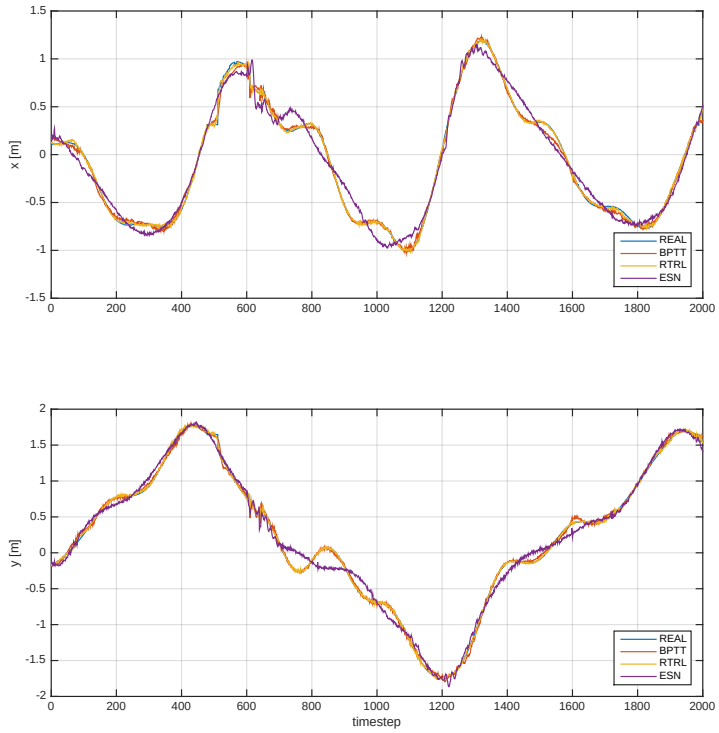


Fig. 7.16.: ML Slung load position estimation after optimising with training data.

	Type	Iterations	Time	Neurons	MSE train	MSE test
Normal	BPTT	10	386.8	5	0.091633	0.09997
	RTRL		354.7	5	0.048233	0.05996
	ESN		3.61	10	0.052887	0.06384
Best	BPTT	***50***	2350.7	20	0.022227	0.03259
	RTRL		***4197.8***	***20***	***0.000311***	***0.00542***
	ESN		12.4	10	0.041639	0.04495
Longest	BPTT	100	4929.4	25	0.03225	0.03663
	RTRL		8252.8	25	0.00017	0.02102
	ESN		12.5	10	0.06014	0.06621

Tab. 7.3.: Machine learning slung load prediction experiment results.

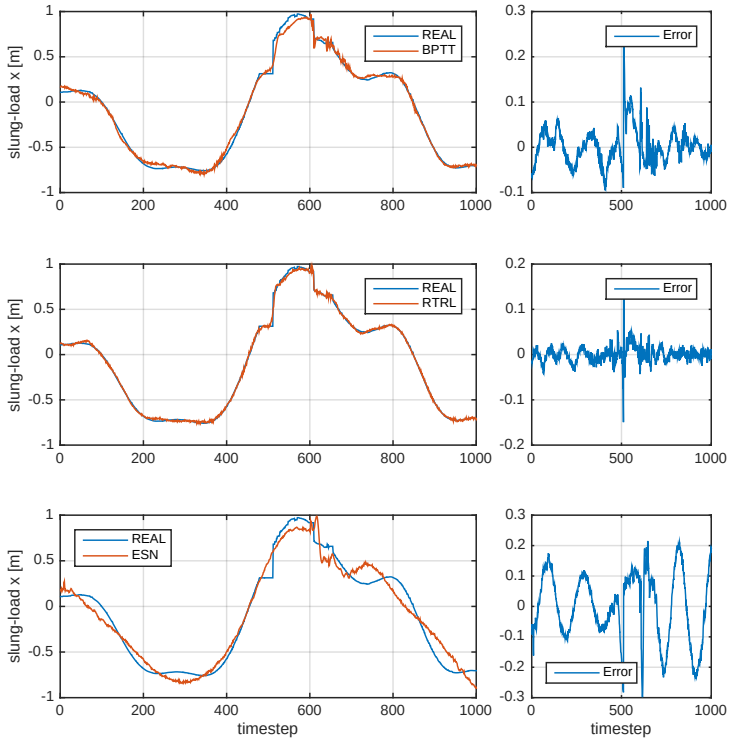


Fig. 7.17.: ML Slung load X-axis position estimation with testing data.

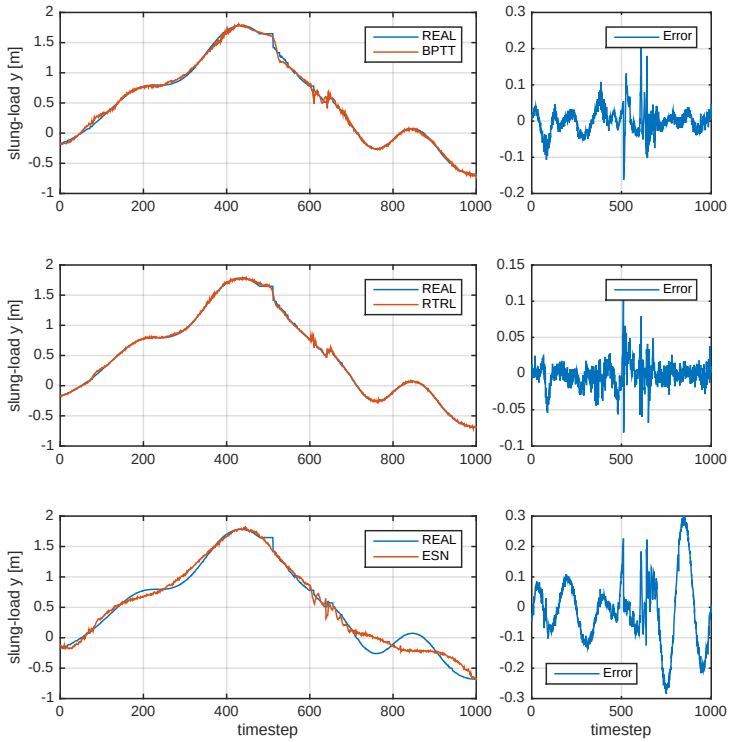


Fig. 7.18.: ML Slung load Y-axis position estimation with testing data.

On-board Tests

Once the networks have reached acceptable low MSE values using test data after running the optimisation routines, the networks must be tested in *real-time* on-board the vehicle. This is the final step to corroborate the adequate estimation of the technique and so that it can be decided if the networks are usable for applications such as control.

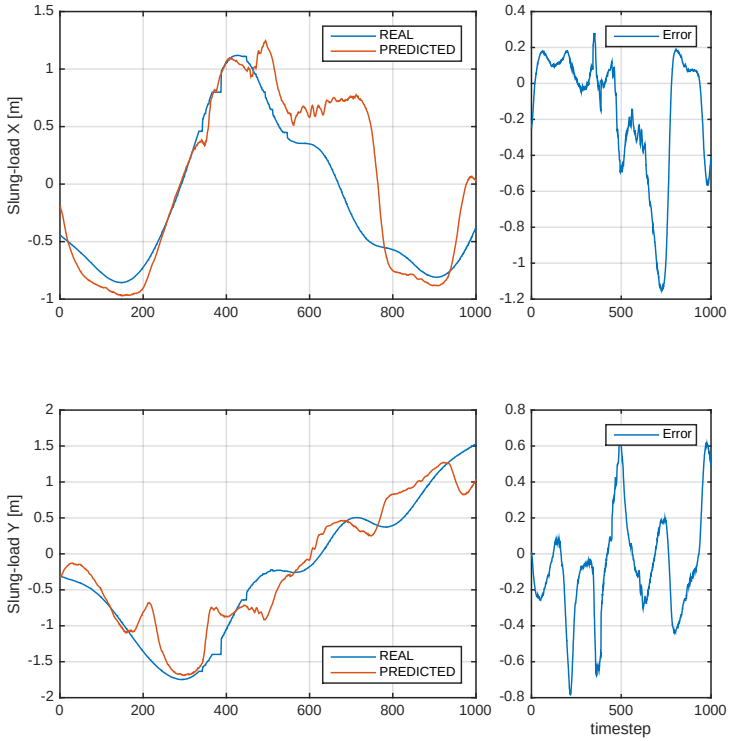


Fig. 7.19.: ML ESN architecture real-time on-board performance predicting the position of the slung load.

The DronePilot framework is designed to achieve the purpose of running experiments in parallel. The selected neural network architecture is executed in parallel with the rest of the necessary processes (*comms, control, black-box*). Due to the companion computer CPU capabilities, only one network can be run at a time, therefore three (or more) different flight tests must be performed to obtain the estimation data from the optimised machine learning architectures (ESN, BPTT, RTRL). It is important to remember that these real-time on-board flight tests were performed in the same entire work regime as the training data was obtained, because it encapsulates part of the research objective which is *reducing the oscillations of a slung load during transport*.

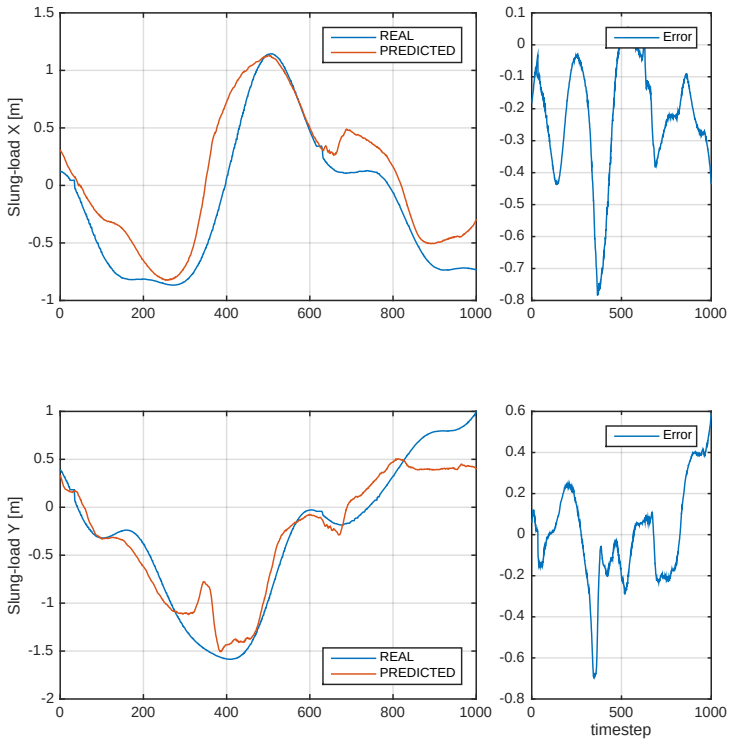


Fig. 7.20.: ML BPTT architecture real-time on-board performance predicting the position of the slung load.

As shown on Table 7.3, the machine learning technique that scored the highest MSE was ESN (high MSE shows poor performance), therefore it was expected that this trend prevailed on the real-time on-board tests. Figure 7.19 shows 1000 time-steps ($10seconds$) of the ML ESN architecture prediction of the slung load position, it is easily appreciated that the architecture is having a high number of *perturbations*.

The machine learning architecture BPTT performance is shown in figure 7.20, showing $10seconds$ as well. This architecture was the second lowest MSE of the three methodologies, it can be appreciated that the prediction follows the real position closer than the ESN, but there is still *perturbation* although not at the same degree of the ESN.

The most accurate (lowest MSE) architecture after the optimisation prediction tests was the RTRL machine learning architecture and this trend continued on the on-board tests, the performance can be observed at figure 7.21. It is appreciated that it follows the trend of the position and the number of perturbations are reduced significantly. Although not entirely free of perturbations, in the $10seconds$ that the plot covers we can appreciate no perturbations, discontinuities or misplaced behaviour. The MSEs of the tests in these $10seconds$ time-frame for the three machine learning architectures can be seen on Table 7.4.

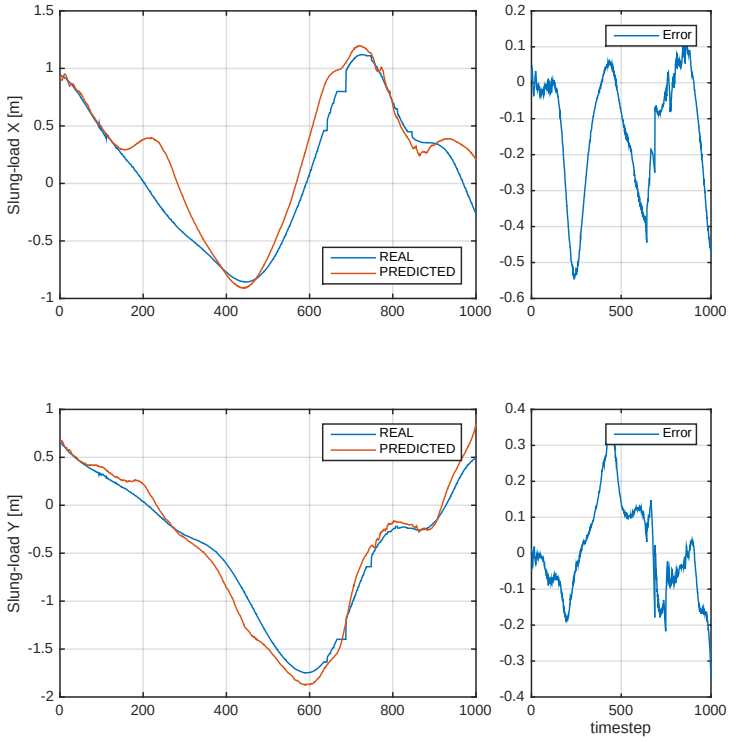


Fig. 7.21.: ML RTRL architecture real-time on-board performance predicting the position of the slung load.

ML technique	SL X-axis MSE	SL Y-axis MSE
ESN	0.0704	0.0447
BPTT	0.0447	0.0287
RTRL	0.0231	0.0091

Tab. 7.4.: Real-time on-board performance of the ML architectures doing the SL position estimation.

To avoid possible vehicle crashes, only the most accurate slung load position estimation tool is used on real control tests, there is no need to risk the vehicle trying to control the position of a slung load that it is not close to the real position values. Figure 7.22 shows a 3D plot comparison of real slung load position with the ML RTRL estimation next to it. On such figure, the position of the vehicles is the same but moved $1m$ apart on the X-axis to easy visualization of the comparison. The objective is that the *PREDICTED* line be similar to the *REAL* line.

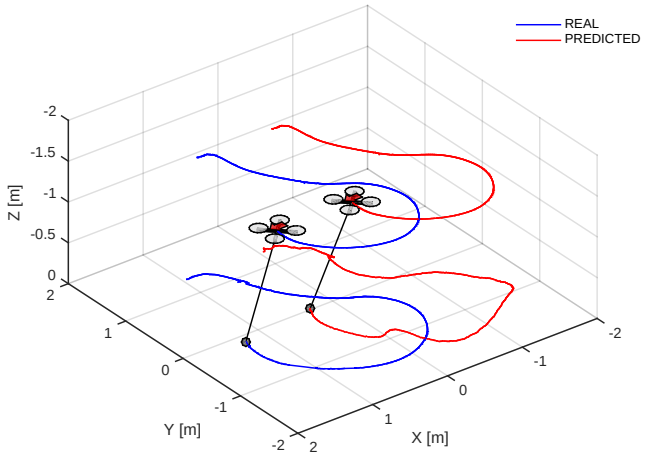


Fig. 7.22.: 3D replay of a flight comparison of the winner machine learning SL position estimation.

7.4 Controller design

The first step in designing any controller is to construct a mathematical model of the equations of motion of the system. Adding additional mass alters, at the very least, the values of the parameters in the equations of motion that define the model - gains, time constants, mode coupling etc. In control theory, there are two approaches to dealing with this problem: treat it as an uncertainty in the feedback loop and apply a robust controller synthesis technique such as H^{∞} or use an adaptation mechanism to alter the underlying mathematical model and controller. For flight with a suspended load the primary impact of adding the load is to induce lateral pendulous oscillations, which can become unstable. This prominent pendulous oscillatory motion affects the response in the frequency range of the attitude control of the vehicle. After designing and incorporating the vehicle stability and tracking controller in Section 4.3, the effect of the load swing forces are taken into account for a new trajectory controller that will attempt to follow the load in order to re-position it to the reference position.

7.4.1 Swing-Free Position Controller

The swing-free controller will attempt to reduce the oscillations of the load when the system is near hover conditions (when the system reached the end of transport). The configuration of the proposed controller is shown in figure 7.24. A linear feedback control system is chosen due to its simplicity and from similar successful applications such as overhead gantry cranes as in Aoustin et al., 2003 and Iwan Solihin et al., 2007. The swing-free position

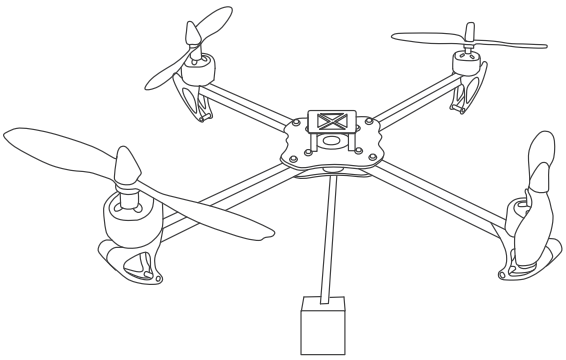


Fig. 7.23.: Diagram of quadrotor with a slung load.

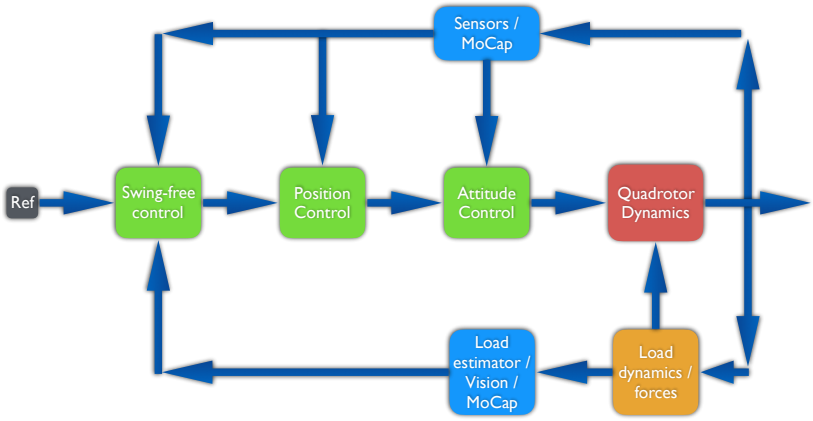


Fig. 7.24.: Configuration of the proposed Swing-free controller.

controller consists of two main control systems, a *MRUAV position controller* and a *slung load position controller*. The slung load controller will generate a trajectory that the quadrotor position controller must track. So, the controller design of the whole system can be divided into two stages. In the first stage, the position controller for the quadrotor alone is designed by neglecting the effect of the slung load on the quadrotor dynamics. The function of this controller is to stabilize the vehicle and follow a reference generated by the anti-swing controller. In the second stage, the whole system is integrated by augmenting the dynamics of the controlled vehicle with the dynamics of the slung load. Then the proposed slung load position controller is added to the integrated system and the performance of the whole system is evaluated. The position controller was presented in Section 4.3 and for the slung load position controller we assume a *reference position* (x, y, z) and then a computation of error signal as in Eq. 7.34. Where Γ_{sl} is the position of the slung load and Γ_q the current position of the quadrotor.

$$e = \Gamma_{sl} - \Gamma_q \tag{7.34}$$

Then, the goal of the slung load controller is to minimize the error signal, only a proportional gain is needed to achieve this goal, due to the cascade of controllers acting after this one. It is important to notice that the error signal is filtered before being sent in order to avoid big changes that can produce bigger errors on the position controller, that could potentially lead to instabilities in the system. Several crashes occurred when testing in the laboratory the proposed controller, due to several factors including space in the room, MoCap system confusing the optical markers and *layer 8* issues. The proposed controller has a limitation, it is linked to the natural frequency of the slung load (Eq. 7.6) which means that it cannot reduce the oscillations of loads that have a higher natural frequency than the step response of the position control (Fig.4.14), simply because the vehicle cannot re-position itself as fast as the slung load displaces. From the experimental results shown in Section 4.4.2 and particularly in figure 4.13, the MAST Lab research MRUAV has a settling time of close to 4 seconds, which means that it will be able to safely reduce the oscillations of loads with a cable length $L \leq 60cm$.

7.4.2 Swing-Free Trajectory Controller

Once the swing-free controller is implemented, a trajectory is added for the quadrotor/slung-load system in order to track it while reducing the oscillations of the load. This is the desired behaviour for an autonomous flight of a slung load under a quadrotor. The overall control concept is a classical cascaded scheme where the outer loop controller (the swing-free trajectory controller) generates references to the inner loop controller (the MRUAV position controller). As seen in Section 4.3.3, the chosen trajectories are a circle and a figure-of-eight. This is because the necessary movements to achieve such trajectory encapsulates the working regime of the experimental data gathering tests, producing high oscillations that then must be suppressed by the proposed controller. The error signal that goes to the position controller is:

$$e = \Gamma_{traj} + k_{psl}(\Gamma_{sl} - \Gamma_q) - \Gamma_q \tag{7.35}$$

Where k_{psl} is the proportional gain of the swing-free controller and Γ_{traj} is the desired trajectory.

Figure 7.25 shows the proposed configuration of the trajectory controller (dashed rectangle), it comprises a similar layout as the trajectory controller developed for the DronePilot framework. For this controller configuration, a flight mode selection module is added to the system, which is in charge of changing the current mode according to the pilot input, from the flight modes available on DronePilot. The flight modes added to DronePilot for this section can be seeing in table 7.5.

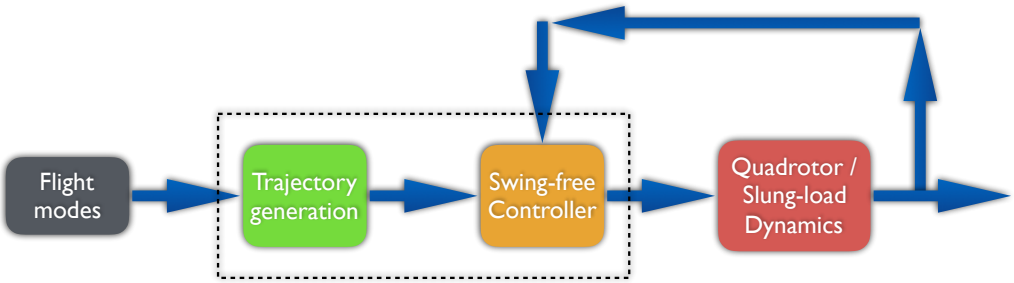


Fig. 7.25.: Configuration of the proposed Swing-free trajectory controller.

Flight mode	**Description**
Manual	Allows to fly the vehicle manually, but self-levels the roll and pitch axis
Position-hold	Automatically attempts to maintain the current position, heading and altitude
Swing-free loiter	Automatically attempts to maintain the current position and reduce the oscillations of the slung load
Swing-free trajectory	The MRUAV will autonomously attempt to track a trajectory while reducing the oscillations of the slung load

Tab. 7.5.: DronePilot flight modes for the Swing-Free Controller.

The *position hold* mode will maintain the vehicle's position while the user attaches the slung load to the vehicle. Once the load is attached the *Swing-free loiter* mode will reduce the oscillations of the load while remaining at the current vehicle position and lastly in the *Swing-free trajectory* mode the vehicle will perform a swing-free pre-selected trajectory, that includes a circle and a figure-of-eight. The experimental results will be shown in the next section.

7.5 Experimental results

When conducting experiments with the slung load, necessary precautions must be taken into account to prevent serious accidents, this is due to the degree of difficulty involved. One disadvantage of doing this test in a confined space such as the MAST Lab is the reduced flight volume, which means small error will grow larger extremely fast, because the walls or roof are very close to the vehicle or load. Fig. 7.26 shows the vehicle stranded on the safety net after a momentarily loss of communication with the vehicle caused by a very aggressive bounce of the slung load.

Fig. 7.26.: MRUAV stranded on safety net during a gathering data flight test.

85% of the accidents with the slung load where caused by an error in the Motion Capture system. This MoCap error had to do with the array of trackables, when performing the initial experiments the position of the slung load Γ_{sl} was given by the second trackable of the MoCap system, but when the first trackable (attached to the vehicle) was very close to the safety net, it disappeared from the view of *some* cameras, therefore the MoCap system

considers it out of view, and displaced the second trackable as the first one in the array. Every $100Hz$ the companion computer received the position of the vehicle, but when the error occurred, the first trackable was now the physical slung load position rather than the vehicle position, causing the height controller to accelerate the vehicle towards the ceiling in an effort to reach the desired altitude requirement, without knowing the current position was the slung load rather than the vehicle. Such an error is easy to fix within the MoCap software by adding IDs to each trackable, but without being able to modify this system (or update it). It became extremely complicated to fix it in the DronePilot framework, therefore flying close to the limits of the MoCap system was not an option.

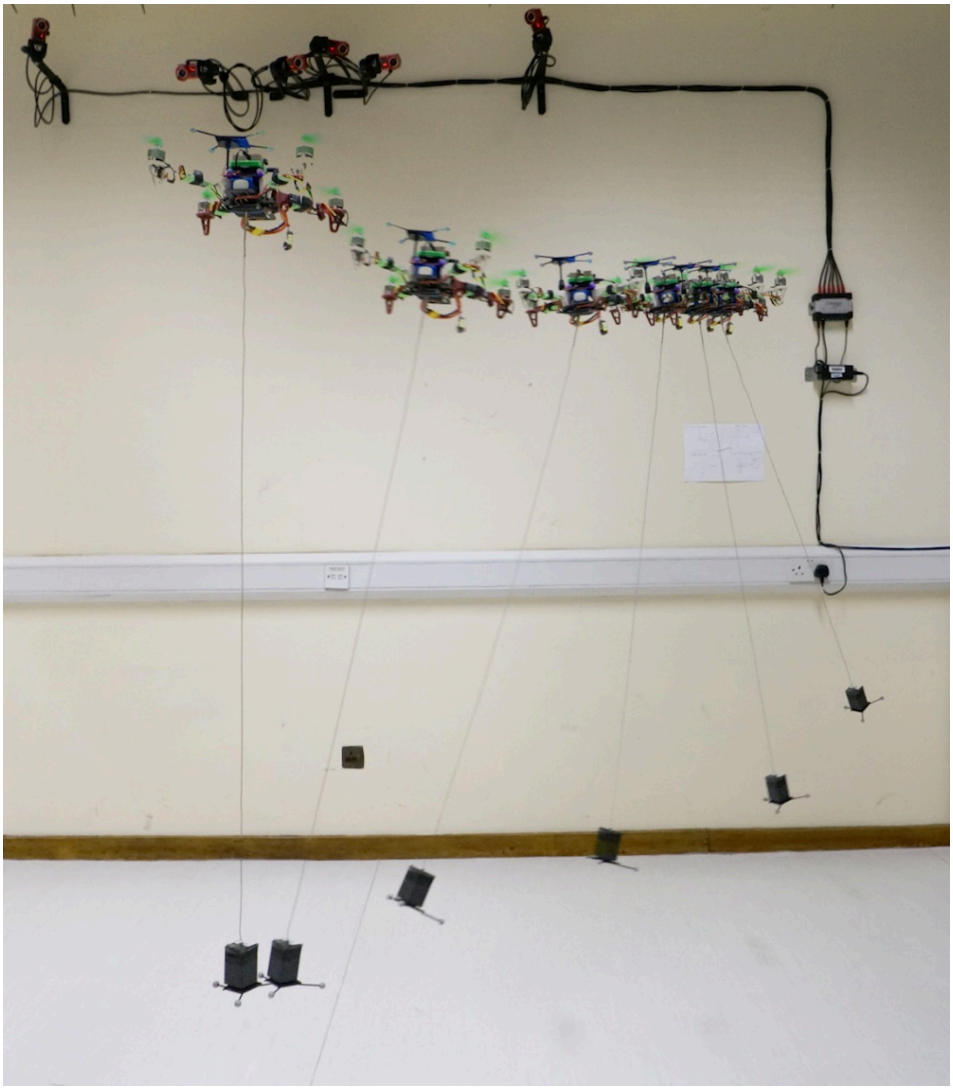


Fig. 7.27.: Time-collapse image of the first oscillation of the step response.

7.5.1 Controller verification

To illustrate the damping effect of the swing-free controller on the MRUAV an aggressive manoeuvre is used. The aggressive manoeuvre is simply a step in position reference. From the initial position of the vehicle with slung load already mounted on it $(0, 0, 1.8)meters$, the vehicle is commanded to track an aggressive change on its position to $(1, 1, 1.8)meters$. The results will only be shown for the X-axis dynamics but the process is similar for the Y-axis dynamics. In Fig. 7.27, a *time-collapse* photography shows the movements of the quadrotor/slung-load system up to the first-maximum value of the first oscillation of the slung load, which occurs around 2 seconds after the $1m$ step response has been applied to the vehicle. The step-response tests are performed both without and with the swing-free

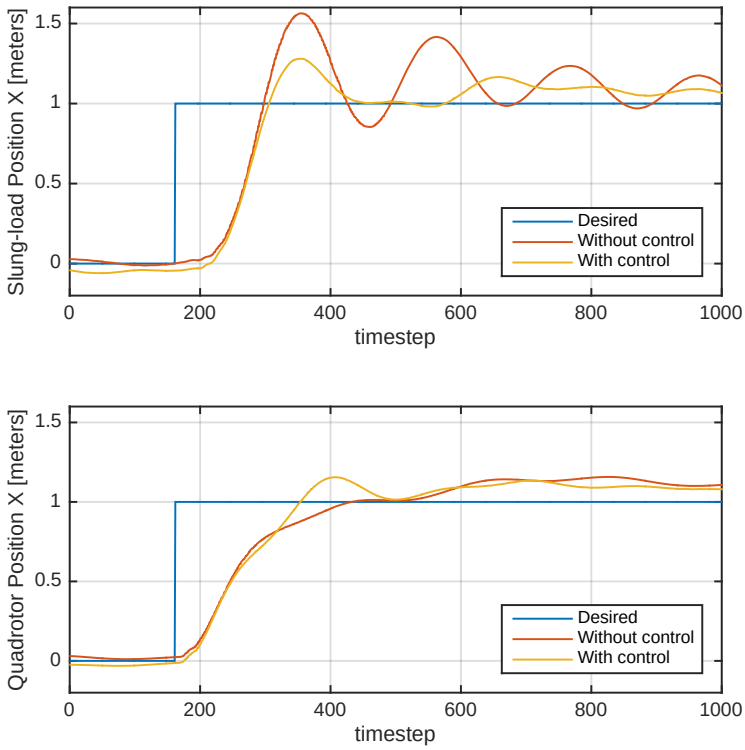


Fig. 7.28.: $1[m]$ aggressive step on X axis with and without swing-free control.

controller and a comparison is shown in Fig. 7.28, it is clear that the control scheme is capable of providing considerable damping of the slung load swing compared to flight without a dedicated slung load controller. Figure 7.29 shows a 3D plot of the same aggressive step, the data comes from two different flights, but mixed and repositioned together for easier comparison of the controller response without swing-free control and with swing-control, it is noted how the slung load trajectory oscillates much less in the latter one. As visual method of comparison Fig. 7.30 shows the transition of the same step response with-

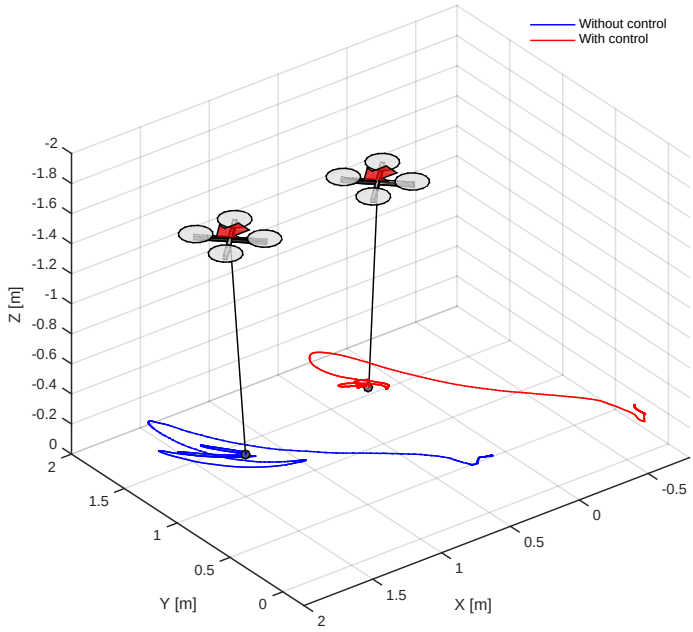


Fig. 7.29.: 3D plot of $1[m]$ aggressive step with and without swing-free control.

out and with swing-free controller. Not all of the oscillations are shown on the 6 selected frames required for the swing-free controller to stop the swinging of the load.

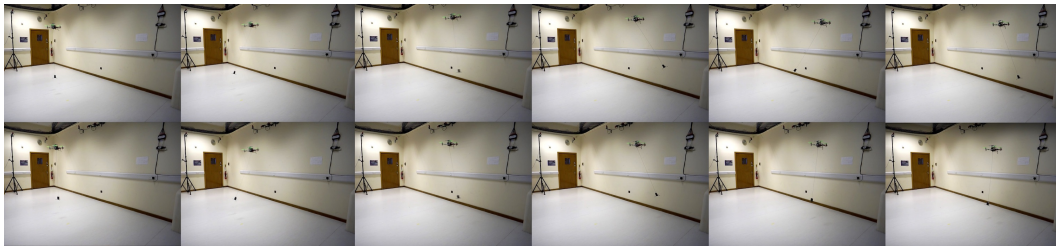


Fig. 7.30.: Transition comparison of the quadrotor/slung-load system without (top) and with (bottom) swing-free control.

7.5.2 Estimator verification

To verify the design of the estimator and test the performance of it when dampening the oscillations of the load, flight tests were carried out using an aggressive step and two different position errors for the swing-free controller. Figure 7.31 shows the controller response on two axis for the slung load position. Two tests were performed changing the source of the slung load position, the first one is using the slung load *real* position coming from the motion capture system, the second one uses the machine learning slung load position estimation. This type of machine learning estimation was described in Section 7.3.2. It

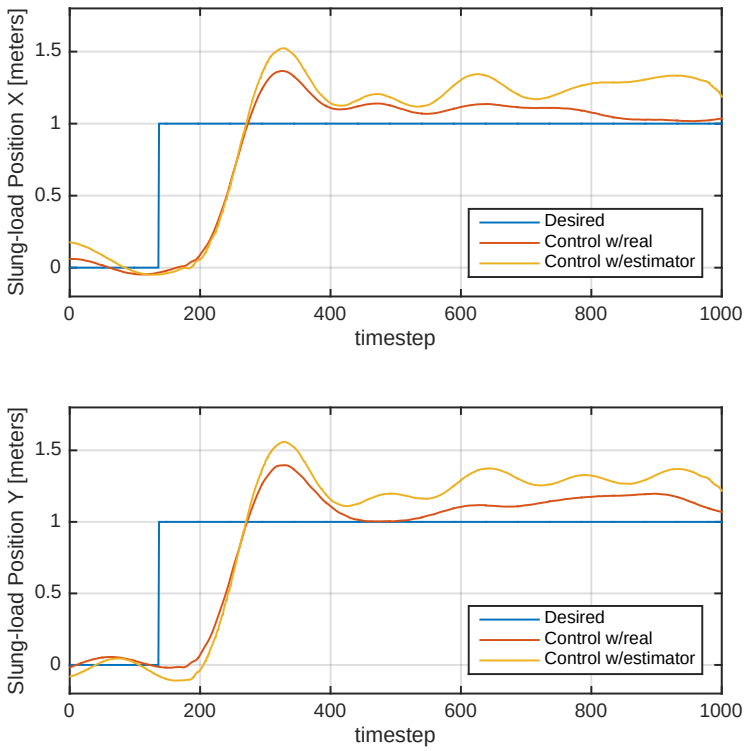


Fig. 7.31.: Controller performance comparison using different sources for the slung-load position.

can be observed that the control response with the estimator fluctuates more in comparison with the control response that used the motion capture readings, this is due to the fact that the machine learning estimator calculates the slung load position with certain band of error. Even if this estimation error is low, the controller is able to dampen the oscillations of the slung load. The controller that uses the estimated position does not have the same performance as the one that uses the real slung load position (Fig. 7.32).

7.5.3 Trajectory response

After verifying that the machine learning estimation of the slung load position works with the proposed swing-free controller, flight tests were performed using a swing-free trajectory controller with the machine learning estimation. In such tests, the prediction of the position of the slung load is fed into the swing-free trajectory controller and the real slung load position is logged from the MoCap system in order to verify the performance of the estimator and controller. Two trajectories are tested (circular and figure-of-eight).

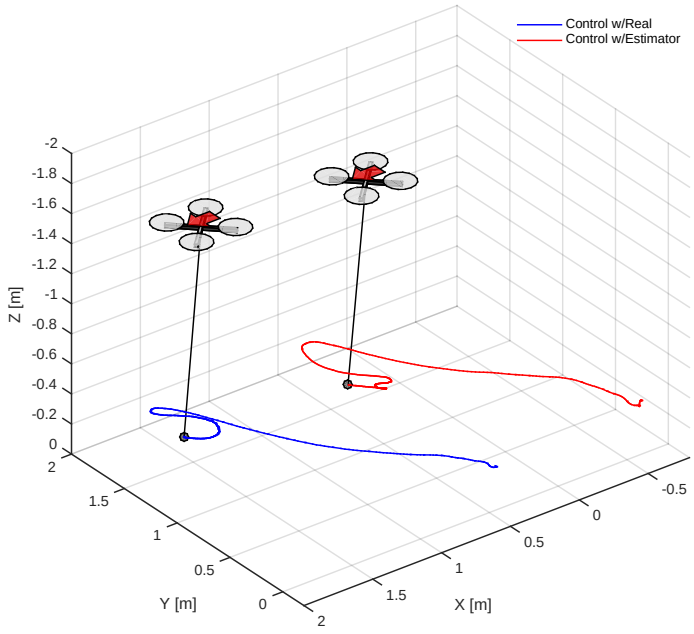


Fig. 7.32.: 3D plot of controller comparison with different sources for the slung-load position.

Circle trajectory response

Firstly, the response of the slung load to a circular trajectory without swing-free control is shown on Fig. 7.33 in order to appreciate the disturbances of the slung load when it is subjected to movements of the quadrotor tracking a trajectory. Such circular trajectory is the same as the one executed in Fig. 4.19 from Section 4.4.3 and it is set to be completed in 5 seconds. It is appreciated that the quadrotor tracks the reference trajectory but it is not as precise due to the slung load affecting its performance.

In Fig. 7.34, the same circular trajectory is repeated with the swing-free trajectory controller activated and using as input the machine learning estimation of the slung load. From the top view we can appreciate three concentric circular trajectories, the outer-most being the one of the slung load, it is now almost a circle, no big disturbances as shown before. The middle circle is the quadrotor trajectory adjusted in order to reduce the oscillations of the load, it appears more off-track than before, this is due to the effort in controlling the load.

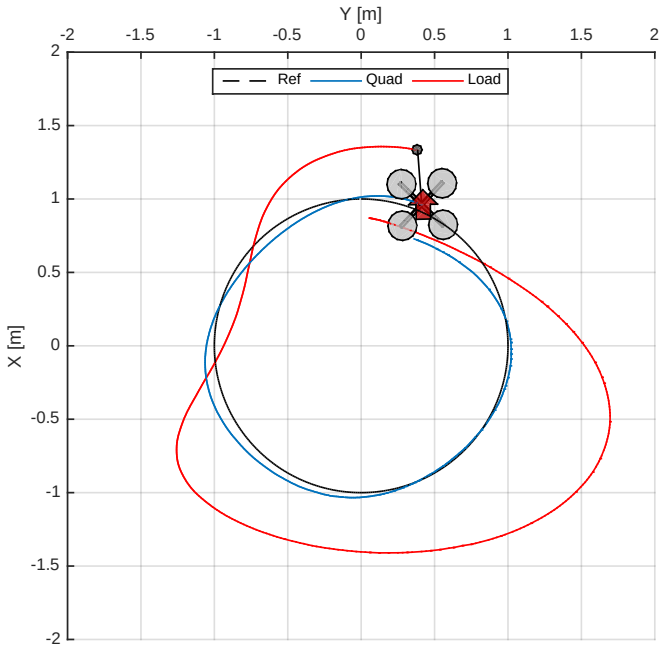


Fig. 7.33.: Slung load response to a circular trajectory - Top view.

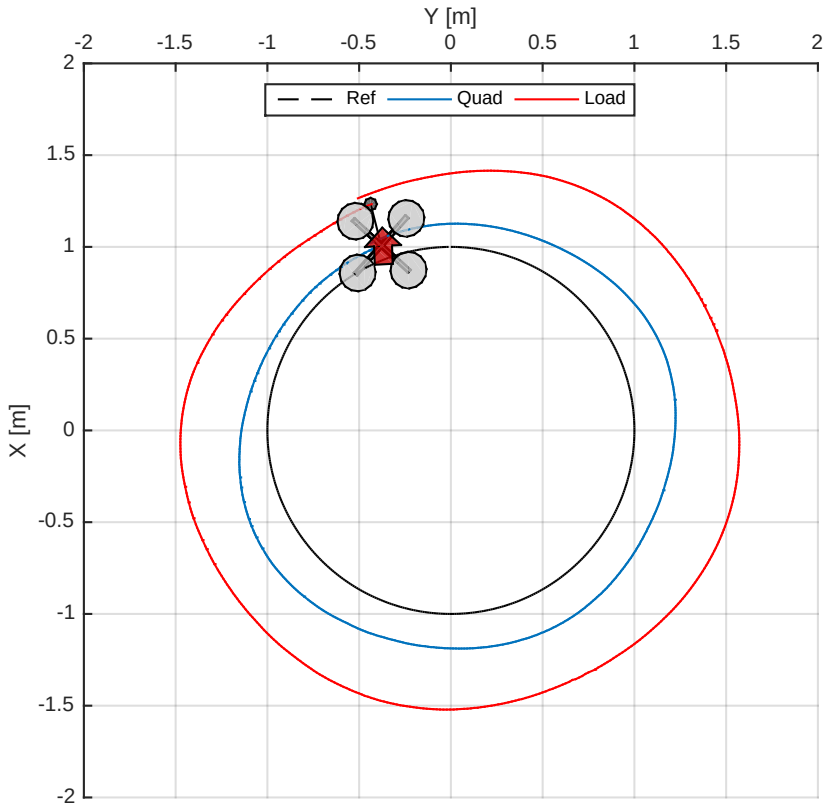


Fig. 7.34.: Slung load response to a circular trajectory with swing-free control active - Top view.

Figure-of-eight trajectory response

The disturbances generated by the slung load being coupled to the quadrotor performing a figure-of-eight trajectory are shown in Fig. 7.35. The trajectory is a *Lemniscate of Bernoulli* set to be completed in 8 seconds. In this case, it is noticed that the slung load disturbances are far larger than for the circular one, in a couple of occasions the slung load does a *loop* due to extreme changes in direction and the quadrotor is having larger performance issues attempting to track the desired trajectory due to being pulled by the slung load.

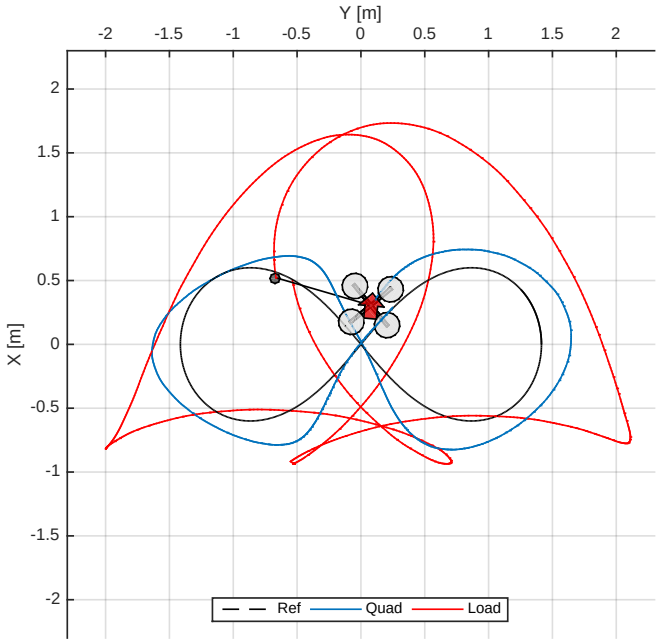


Fig. 7.35.: Slung load response to a figure-of-eight trajectory with swing-free control active - Top view.

The response of the slung load being controlled by the swing-free trajectory controller while the quadrotor is performing a desired figure-of-eight trajectory can be seen on Fig. 7.36. The load is free of disturbances and loops, it is noticed that the quadrotor trajectory diverged further from the desired trajectory, but this is due to the aggressive motions the slung load creates when moving.

As a final demonstration of the performance of the swing-free trajectory controller, a test was done in the MAST Lab with a camera doing *light painting photography* (long-exposure) which involves using a long-duration shutter speed to sharply capture the illuminated moving elements. In this case, the flight controller contains status bright LEDs (blue) and the slung load carries a red LED.

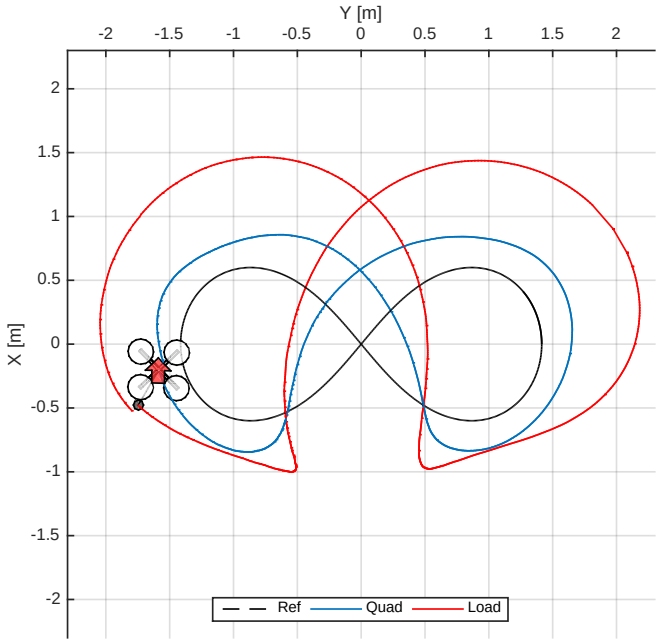


Fig. 7.36.: Slung load response to a figure-of-eight trajectory with swing-free control active - Top view.

Two light painting photographies were taken and are displayed on Fig. 7.37, both show two trajectories, in blue colour is the visual trajectory of the flight controller which is placed close to the CoG of the quadrotor and the slung load is the red colour trajectory, the slung load cable is not visible. The top image shows the quadrotor performing a figure-of-eight while the load is not taken into consideration, therefore it swings free and affects the dynamics of the final quadrotor trajectory. The bottom image shows the swing-free trajectory controller in action.

7.6 Summary

In this Chapter, the dynamics of the load coupled with a multirotor were presented. Two methods for estimating the position of the load were described and tested. It was proven that the machine learning position estimator is capable of predicting the position of the load in real-time so that it can be used on a anti-swing controller scheme. Results from the controller scheme were presented in the forms of plots and a light-paint photography that showed the oscillations of the load without anti-swing control and the result when the anti-swing control is active.

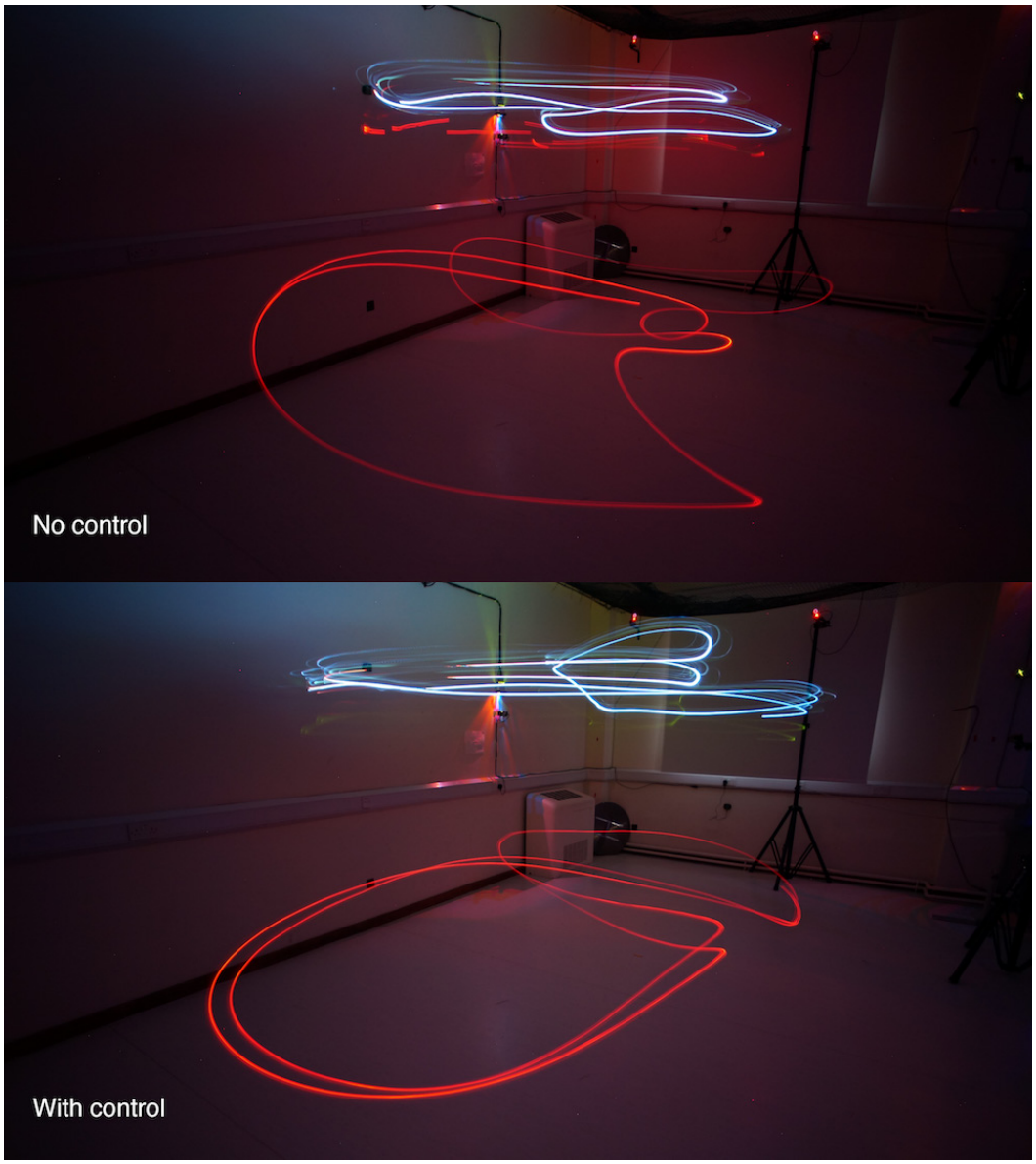


Fig. 7.37.: Light painting photographies of comparison flight tests with the swing-free controller.

8 Conclusion

One of the major contributions of this thesis is the development of estimators capable of estimating the position of the slung load relative to the vehicle. This objective was accomplished using machine learning techniques by the creation of an estimator of the slung load position relative to the MRUAV.

The machine learning estimator was designed using a recurrent neural network structure which was then trained in a supervised learning approach using real flight data of the MRUAV/SL system. This data was gathered using a motion capture facility and a software framework (DronePilot) which was created during the development of this work. After the slung load estimator was trained, it was verified in subsequent flights to ensure its adequate performance. Consequently, a control system was created and tested with the objective to remove the oscillations (swing-free) generated by the slung load during or at the end of transport. The control technique was verified and tested experimentally.

The proposed approach is an important step towards developing the next generation of unmanned autonomous multirotor vehicles. The methods presented in this thesis enables a quadrotor to perform flight manoeuvres while performing swing-free trajectory tracking.

8.1 Summary of contributions

8.1.1 On multirotor design

This thesis contributed a theoretical/experimental method aiming to predict the time of flight while hovering of a multirotor vehicle. The proposed method uses data that is commonly given by the COTS manufacturers, which sometimes can be scarce. If the latter is the case, then a rotor analysis tool presented in Appendix A.3, can be applied in order to complement the required information to estimate the time of flight.

It was demonstrated that the approach followed in this work provided flight times closer to the real life case, compared to estimations using other methods, such as on-line calculators. The main difference being that the on-line calculators contain a detailed database

compromising a large amount of COTS components. Thus, a combination of methods is the preferred tool when designing a multirotor for a specific mission when experimental data cannot be easily obtained. Furthermore, the use of 3D printing techniques was of great benefit to the MAST Lab. It proved to be an effective method for manufacturing the entire frame of a multirotor vehicle, as well as extra components for servo-gimbals, flight controller fittings, among others. However, the 3D printer's building volume capabilities limit the size of the overall multirotor frame. The design of the quadrotor vehicle *TEGOv2* presented in Sec. 2.1 Fig.2.2 improved the previous design by reducing the overall weight of the final vehicle by not using nuts and bolts in its construction. Using lighter Rotite (Burns, 2014) elements instead resulted in a time of flight increasing by approximately 22%. More importantly, as an added benefit, the arm of the quadrotor was able to *rotate* without coming loose from the frame, therefore improving crash-survivability of the *TEGOv2* frame.

The usage of the Rotite elements on the design of the quadrotor *TEGOv2* are the first aerospace application of such mechanical fastener in the world. The company Rotite and its products received widespread attention from a number of aerospace-related companies that became potential clients after they saw the vehicle flying on [1] as well as in *The 2014 Gadget Show Live* [2] at the NEC (National Exhibition Centre at Birmingham), where the author of this thesis attended to present the quadrotor frame along with Rotite executives. Undergraduate students from the ASDP (Aerospace Systems Design Project) course were benefited by being given the 3D CAD files of *TEGOv2* in order to build different multirotor vehicles. Two models were presented in their final reports, a hexarotor (left) and an octorotor (right) (Fig.8.1). These projects triggered research questions that were finally addressed and presented in Ireland et al., 2015.

Fig. 8.1.: Hexarotor (left) and Octorotor (right) based on *TEGOv2*.

Moreover, the lessons learned from the multirotor design chapter allowed the author of this thesis to select COTS components for the MAST Lab current and future test-beds of different

[1] `https://www.youtube.com/watch?v=G9jUP6Z5ENA`
[2] `https://en.wikipedia.org/wiki/The_Gadget_Show`

sizes, as well as for academic competitions such as the *IMechE UAS Challenge*, where the author has supervised and led the University of Glasgow team in two occasions (2015 and 2016).

8.1.2 On MRUAV control

The *flight stack* Vargas et al., 2016 was described and presented in Sec. 3.3 and it has become one of the biggest contributions of this thesis, due to the impact that this solution has had on the MAST Lab and in other universities and companies around the world. Figure 8.2 is composed of two light painting photographies showing the difference of a quadrotor attempting a figure-of-eight trajectory tracking using the old structure (left) and after the flight stack was implemented (right).

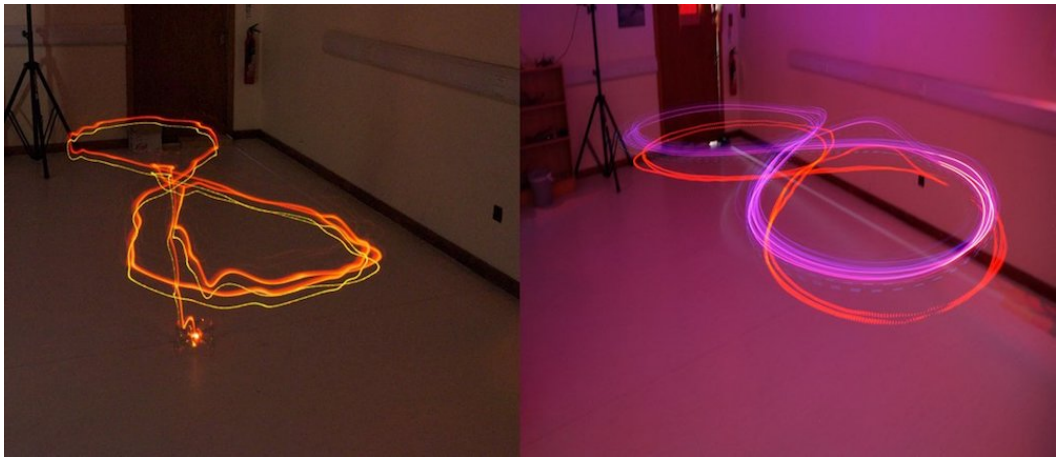


Fig. 8.2.: Comparison light painting photographies of previous structure (left) vs flight stack (right).

A key consideration in the design of the flight stack is the distribution of computation between on-board and external processing and communication between vehicles and with external systems. The flight stack can be defined as a system used for GNC (Guidance, Navigation and Control) of a UAV without constant control by a human operator being required. Since this idea was first used and released by the author of this thesis it has being accepted and replicated over a number of projects internationally (USA, UK, China, Canada, India, Pakistan, Israel, Mexico), thanks in part to the videos [3] [4] and the blog posts [5] created/distributed by the author. Moreover, along with the *DronePilot* framework, such a contribution becomes a useful tool for researchers working in GNC of UAVs.

[3] `https://www.youtube.com/watch?v=TkYeQ6orN8Y`
[4] `https://www.youtube.com/watch?v=XyyfGp-IomE`
[5] `https://altax.net/blog/flight-stack/`

The *pyMultiWii* python module (Vargas, 2013b) is another open source software contribution. The module is described in A.4 and it is in charge of sending and receiving commands to and from the *naze32* flight controller. Currently, this repository is one of the most visited and used from the author. Its purpose is not only related to UAV's, but it can be used on several applications (ground robots, projects requiring IMU and/or orientation data). The author used it on an interdisciplinary project at the University of Glasgow called *Anemoi*. The project and its results are shown in [6].

The control systems inside the *DronePilot* framework (Vargas et al., 2014) used in this thesis perform accurately and precisely, and their code is open source so that any person interested can modify them, replicate them and use them in their own research. *DronePilot* has been released with a *GNU GPLv3* license, which is a copy-left license that requires anyone who distributes the code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users. Even though the chosen control structures inside *DronePilot* are PID, the framework is designed in such a way that it allows an easy change of the control structure in order to implement other academically interesting control methodologies. Furthermore, with the aid of the extra companion computer, the system can benefit from the integration of extra peripherals. During the developing of this thesis, several peripherals were tested alongside the MRUAV test-bed, such as an ultrasonic sonar, CV camera sets, indoor positioning systems and cameras. Such devices were not documented in this thesis due to the fact that they were not used for the main goal of the research. Nevertheless, incorporating these capabilities adds functionality and flexibility to the system and contributes to future applications.

Analysing and testing two of the currently most important flight controllers (*naze32* and *Pixhawk*) for UAVs has a great relevance to researchers interested in this field. Results of this thesis may aid researchers in their assessment and selection of flight controllers that better suits their particular research needs. It has been shown that *Pixhawk* is more oriented to outdoor projects while *naze32* will be more suited for indoors applications. These conclusions further indicate that there is no general purpose flight controller, and that it has to be selected according to the particular requirements of the project. The limitation of the flight stack in this project relies in the flight controller, its control methodology and responsiveness; yet if a different flight controller is used, the control strategy would have to be modified to fit the new avionics suite control strategy. The *flight stack* and sections of the *DronePilot* framework were used by the University of Glasgow team in the *IMechE UAS Challenge* (2015 and 2016), in order to fly a MRUAV through a set of specified way-points inside a closed airfield. The vehicle was required to search (via a special visual marker)

[6] `http://www.gla.ac.uk/colleges/scienceengineering/staff/newsletterarchive/newsletterjune2015/headline_406758_en.html`

for a location where a payload ($1kg$ of flour) had to be dropped and then continue to another set of way-points, ending up landing safely; with the premise that everything must be performed autonomously.

8.1.3 On machine learning

It was demonstrated that RNNs are an excellent tool as universal approximators (predictors and estimators) for time-series sensor data, altought in some occassions they have the tendency to learn what was not expected from them to learn. The applications of RNNs algorithms (ESN, RTRL, BPTT) used in this work are an effective tool for non-linear system modeling and control (shown on simulation in Vargas et al., 2014).

The RNN ESN approach is the fastest algorithm to iterate into usable results and, when combined with optimisation, it can achieve better results in comparison with BPTT and RTRL approaches. Training times decreased 99.7% from RTRL to ESN and 99.6% from BPTT to ESN which shows one of the great advantages of using the *Reservoir Computing* approach. All the methodologies showed good consistency when compared with observed real flight data of the reviewed applications. Regarding experimental data collection, which is an important branch for machine learning algorithms, it was shown that gathering the same kind of input during training when collecting data results in a more efficient process, as it will later be required in testing or at exploitation, as long as the training data is more varied that it is expected on the exploitation phase. The author has used the *Reservoir Computing* ESN approach in Vargas et al., 2014, Vargas et al., 2015b and Vargas et al., 2015a, however it has been proved that the ESN approach became the best option when doing system identification of MRUAVs, followed by RTRL and BPTT. Alternatively, the best slung load relative position estimator was the RTRL approach, with BPTT being the second best and ESN the third one. These results render further evidence that when doing this type of research it is good practice to complete tests with different approaches in order to find the best overall fit.

The BPTT and RTRL approaches require large amounts of CPU power when training. The *Reservoir Computing* methodologies can possibly be integrated in an *on-line* manner (training or reinforcement can happen in real time) on board of the companion computer, thus, creating several new paths of applications for these types of machine learning approaches for multirotors. Increasingly powerful computers create opportunities for applications of large RNN such as ESN. Even with modern computers the basic ESN design recipe is not sufficient to create a successful ESN for a variety of important applications. Nevertheless,

improved ESN design and training procedures are needed to increase the chances of getting a successful ESN in a reasonable number of trials.

CMA-ES was proved to be a satisfactory evolutionary strategy not only for the ESN approach but for all RNNs algorithms. It was even tested against a genetic algorithm optimisation approach for a sin wave generator. In this test, the evolutionary strategy reduced the error around 95% against the GA approach. The machine learning methodologies designed and used in this thesis naturally solved sensor aliasing problems, which are common problems when working in facilities like the one used (Motion capture), as well as with noisy IMU readings from the flight controllers for UAVs.

8.1.4 On system identification of multirotors

As shown from experimental results, the black-box models generated in this work can have good generalization capabilities and can learn the dynamics of a MRUAV with good accuracy. More importantly, it was demonstrated that the learned dynamics can be used effectively on-board of the system, inside the flight stack. It was proven experimentally that the system identification architecture is potentially a solution for short indoors *GPS-denied* flights because the vehicle can predict, with a margin of error, its own position from current flight data. As an example, if the MRAUV is flying outdoors with GPS coverage and there is a sudden GPS-signal error, the proposed algorithm could predict the MRUAV's position until the GPS-signal is restored.

The methods described in Section 6.2 describe the data collection process which is paramount for generating black-box models. The flight techniques must be a combination of manual and automatic flights. This is done to encourage good results on test data operating in a working range that was used for training. Having to satisfy this rule with a non-linear system is complicated, while exciting all the rich dynamics of MRUAVs can produce large instabilities ending up in an increase chance of crashing.

The evolutionary strategy CMA-ES helped improved the parameters of the ESN, decreasing the error by 99.2%. It also highlighted that the optimal spectral radius for system identification application must be greater than the one stated at the beginning of the research. The task of identifying the dynamics of a quadrotor therefore requires a larger memory for the input when using echo state networks. The best-performance RNN algorithms in the system identification of multirotors were ESN and RTRL. The MSE between them after running optimisation routines was 0.0007, indicating consistent performance. BPTT lagged behind the other two architectures.

8.1.5 On slung load estimators

Two slung load position estimators were presented and tested. The first estimator uses a vision based system (camera) as the only sensor input. This estimator uses a stream of images from a downwards looking gimballed-camera to calculate a position vector of the load in the MRUAV fixed frame pointing from the camera to the load, therefore several computer vision colour tracking algorithms were designed. These CV algorithms became another open source software contribution of this thesis in the form of a repository [7], which is described in Appendix A.5. The repository led to a technical reviewer position for the author of this thesis and the work can be seen in Pajankar et al., 2015.

The second estimator uses real flight data to train a machine learning architecture that can predict the position vector of the load in the MRUAV fixed frame using the vehicle pose and pilot pseudo-controls as input. This estimator was tested experimentally with excellent precision and accuracy results. The data collection methodology was of great importance due to the different flight modes created in order to excite the dynamics of the slung load. This approach required testing and replacement of a number of frame parts and multirotor components. The machine learning slung load position estimator shows good performance and robustness when non-linearity is significant and varying tasks are given in the flight regime. The performance of the control scheme was evaluated through flight testing and it was found that the control scheme is capable of yielding a significant reduction in slung load swing over the equivalent flight without the controller scheme. The performance of the control scheme with and without controller can be seen in Fig. 7.37. The control scheme is able to reduce the control effort of the position control due to efficient damping of the slung load. Hence, less energy is consumed and the available flight time increases.

Regarding power management, flying a MRUAV with a load will reduce the flight times because of two main factors. The first one relates to adding extra weight to the vehicle, consequently the rotors must generate more thrust to keep the desired height of the trajectory controller, hence reducing the flight time. The second factor relates to aggressive oscillations of the load for this reason. The position controller demands faster adjustment to the attitude controller which increases accordingly the trust generated by the rotors. The proposed swing-free controller increases the time of flight of the MRUAV when carrying a load by 38% in comparison with the same flight without swing-free control. This is done by reducing the aggressive oscillations created by the load.

[7] `https://github.com/alduxvm/rpi-opencv`

8.1.6 Main contribution

The main contribution of this thesis is the development of a control system that can be integrated into an unmanned autonomous multirotor and thereby enable swing-free slung load flights. This is achieved through a two-step approach: First a slung load estimator capable of estimating the relative position of the suspension system. This system was designed using a machine learning recurrent neural network approach. The final step is the development of a feedback cascade control system that can be put on an existing unmanned autonomous multirotor and makes it capable of performing manoeuvres with a slung load without inducing residual oscillations. The overall control concept is a classical tri-cascaded scheme where the slung load controller generates a position reference based on the current vehicle position and the estimated slung load position. The outer loop controller generates references (attitude pseudo-commands) to the inner loop controller (the flight controller).

8.2 Future work

The methodologies developed in this thesis were created for indoor vehicles inside the MAST Lab. As future work, the proposed control scheme can be implemented outdoors on a larger multirotor similar to the one presented in Fig. 7.4. The implementation can be carried out using the same flight stack, but changing the flight controller from *naze32* to a *pixhawk*, which is more suited for GPS-enabled flights. The DronePilot framework is ready to interact with the *pixhawk* flight controller and changing sections of the core code will enable running tests. The machine learning slung load position estimator will have to be modified in order to run at a different rate as the one used indoors. GPS devices supply the system with $10Hz$ position updates while the Mocap system provides it 10 times faster. The computer vision estimator could be used to gather data for a new machine learning estimator, therefore allowing a MRUAV to predict the slung load position not requiring an on-board gimballed camera.

In the system identification of multirotors, a general model can be created using training data from outdoors flights including multirotors of different sizes and different components. This data can be used to train a black box system model that will be able to predict the states of a general multirotor vehicle while flying. Therefore, allowing the vehicle to be able to safely continue flying even if there is some problems with its sensors, including the position system (GPS).

A second future opportunity is to convert the DronePilot framework into a *ROS2* application. This framework could be expanded or improved if the methodology of ROS (Robot Opera-

tive System) (Quigley et al., 2009) is used. Such a methodology provides a common state-of-the-art platform to achieve communication within and between heterogeneous robots. Moreover, recently there is a new framework that helps ROS users to develop applications of robot swarms, by providing essential mechanisms, such as abstraction of swarms, swarm management, various communication tools, and a runtime environment, all within the standard ROS ecosystem. ROS2 is the new iteration of ROS and it will take advantage of the opportunity to improve our user-facing APIs. At the moment of this document being written it is under heavy development and all releases are currently "alpha"-prefixed.

ROS2 will integrate new cases such as:

- Teams of multiple robots
- Small embedded platforms
- Real-time systems
- Non-ideal networks
- Prescribed patterns for building and structuring systems

This means that a potential new application can be more easily created in the case of swing-free multi-vehicle slung load operations, allowing a fleet of MRUAVs to carry a larger load. To date most scientific research and commercial applications are limited to using single expensive multirotor UAVs. A future task is to develop inexpensive and expendable Mini-UAVs and the associated control technologies that would enable a multitude of UAVs to perform complex tasks cooperatively. These tasks may include object manipulation such as lifting and delivery of swung loads to disaster locations or data collection tasks of sensor fusion used in natural resource management.

Creating a swarm of MRUAV entails a number of challenges, among them, being able to achieve autonomy (difficult to control remotely by single pilot) and coordinating among a group of vehicles that differentiate the swarm from single vehicle operations. One of the main advantages of using a swarm is that if one of the MRUAV becomes inoperative (e.g. due to battery life-time or failure), the swarm can keep going with its current mission. Besides, applications such as surveillance or search and rescue that require coverage of large areas or imagery from multiple sensors can be addressed by coordinating multiple MRUAVs, each with different sensors.

Deep learning, which was first theorized in the early 80's (and perhaps even earlier), is one paradigm for performing machine learning. Due to a flurry of modern research, deep learning is again on the rise due to its potential as a good tool to teach computers to do what human brains can do naturally.

8.3 Extra support and projects

The author of this thesis directly supported and guided the following undergraduate and MSc theses:

- Kirill Kurbanov. *Design of a compound quadrotor*. BSc Aeronautical Engineering. University of Glasgow. 2014
- Daniel Finnigan. *A Comparison of System Identification Techniques for Rapid Prototyping of micro Unmanned Aerial Vehicles*. MEng Aerospace Engineering. University of Glasgow. 2015
- Krisjanis Kuksa. *Implementing a real-time control on a micro-controller*. MEng Aerospace Engineering. University of Glasgow. 2015
- Ulises Ramirez. *Diseño, construcción y control de una aeronave tipo dron*. BSc Mechatronics Engineering. National Autonomous University of Mexico. 2015
- Davide Restuccia. *Object detection for navigation of micro UAV nap of the earth flight*. MEng Aerospace Engineering. University of Glasgow. 2015
- Kyle Brown. *Multi-rotor System Identification and Control using Velocity Vector Commands*. MEng Aerospace Engineering. University of Glasgow. 2016
- Michael Caba. *Severe Accident Mobile Investigator (SAMI) - Quadrotor vehicle*. BSc Systems Engineering. University of North Carolina at Charlotte. 2016

8.3.1 IMechE UAS Challenge

The author of this thesis supervised and lead an interdisciplinary team of undergraduate and graduate students from the University of Glasgow to carry out the IMechE UAS Grand Challenge (IMechE, 2014). This competition (Fig.**fig:conclusion:grand**) undertake a full design and build cycle of a Unmanned Aerial System with a specific mission objective. In the first two seasons of the competition (2015 and 2016) the mission objective was to safely deliver a payload thorough a way-point circuit, finding the area where the payload must be delivered and return to the takeoff location. The main requirement was that the mission must be carry out autonomously, with no input from the users (except start and stop) or pilot unless the aircraft was out of control.

These type of projects encourage and promote UAS research within academia as well as promoting inter-university collaboration to encourage fundamental and interdisciplinary UAS research. The Glasgow team was directly benefited with MRUAV design, flight stack, human pilot and a modified version of the DronePilot software which are contributions of this thesis and author. In the 2016 challenge, the Glasgow team manage to obtain two

Fig. 8.3.: University of Glasgow quadrotor vehicle performing during competition

awards and a commendation. The awards included *Most Environmentally Friendly Team* and *Most Promise Team* while the commendation was for *Manufacturing*.

8.3.2 Media outreach

Another practical contribution was the text and video documentation of most of the parts of this research work in order for external people to benefit from it. In the text form, several work is documented in blog posts, how-to instructions and similar items in [8].

In the video documentation section, the most relevant videos from the author are shown in Tab. 8.1. This methodology became apparent to the author since he participated in the *Mathworks Simulink Challenge 2014*. Recently (Jul-2017), the author has more than 506 subscribers and 133,688 views on his videos.

Consider it DrONE [TM]. - Aldo Vargas

[8] `http://altax.net/blog/`

Title	Youtube ID	Release date	Views
Drone Pilot - Slung Load controller (raspberry pi + naze32)	94agSRWyJPc	Feb 8, 2017	6848
Computer vision using GoPro and Raspberry Pi	Z2Hq4jDWunk	Aug 17, 2016	4465
Drone Pilot - Trajectory controller (raspberry pi + naze32)	k6tswW7M_-8	May 13, 2016	4457
Computer Vision test using a Raspberry Pi 3	pu_9DGT2qO0	Mar 12, 2016	436
Drone Pilot - Position hold controller (raspberry pi + naze32)	oN2S1qJaQNU	Feb 20, 2016	3521
Color detection using QX10 and openCV	sRRwZ2hWfGU	Feb 12, 2016	381
HD low latency video transmission with Raspberry Pi	0nqXGWzH2-s	Oct 1, 2015	5427
5.038 kg HobbyKing Beer Lift 2015 - 500 mm class	orGHgrHXOYc	Sep 7, 2015	819
Flying drone from computer - raspberry pi + naze32	XyyfGp-IomE	Aug 23, 2015	5853
Flying drone from computer - raspberry pi + pixhawk	TkYeQ6orN8Y	Aug 23, 2015	40544
Odroid U3 + naze32	XpUyepii0	Jul 29, 2015	734
Raspberry Pi commands a multirotor to take off	KnjYYBKLK0s	Jun 20, 2015	1564
NoIR camera onboard drone	cfCNVA1C098	Apr 8, 2015	609
Drone color tracking	xlQw_mnJtNQ	Mar 17, 2015	1291
Quadcopter Position Controller #SimulinkChallenge2014	suD0DdpGi8k	Dec 18, 2014	1501
MultiWii + Rapsberry Pi sending UDP to Simulink	ZMc3AZBpyaE	Dec 7, 2014	19497
Python and MultiWii Serial protocol	TpcQ-TOuOA0	Dec 1, 2014	1915
TEGO indoor position control	m20qs8lJPCY	Sep 9, 2014	215

Tab. 8.1.: List of the most relevant instructional videos.

A Appendix

A.1 Makerbot Replicator 2

In September 2012, Makerbot Industries introduced the Replicator 2. This printer has a build envelope of ($285.0\,\mathrm{mm} \times 153.0\,\mathrm{mm} \times 155.0\,\mathrm{mm}$ (Width x Depth x Height)) and can print at $100\,\mu\mathrm{m}$ per layer. It can print only PLA plastic and does not include the heated build plate, extruder, or high-temperature settings for ABS plastic. Full specifications are shown in Tab. A.1.

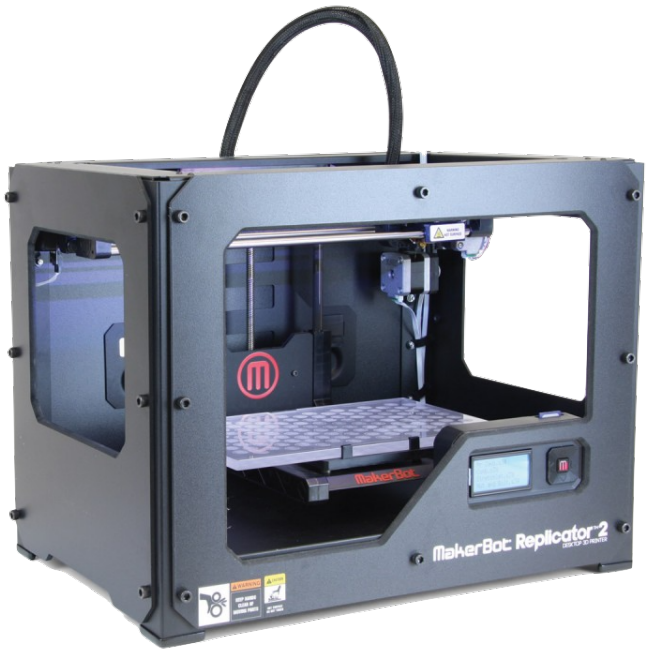


Fig. A.1.: Makerbot Replicator 2

Print Technology	Fused Filament Fabrication
Build Volume	$285.0\,\mathrm{mm} \times 153.0\,\mathrm{mm} \times 155.0\,\mathrm{mm}$
Layer Resolution	$100\,\mu\mathrm{m}$
Positioning Precision	XY: $11\,\mu\mathrm{m}$ Z: $2.5\,\mu\mathrm{m}$
Filament Diameter	$1.75\,\mathrm{mm}$
Nozzle Diameter	$0.4\,\mathrm{mm}$

Tab. A.1.: Makerbot Replicator 2 specifications

A.2 Rotite

The information showed in this appendix was given by the inventor/designer of the Rotite, Mr. Stuart Burns. Only the Rotite A will be showed on figure A.2. The design was given only for the usage of this academic effort.

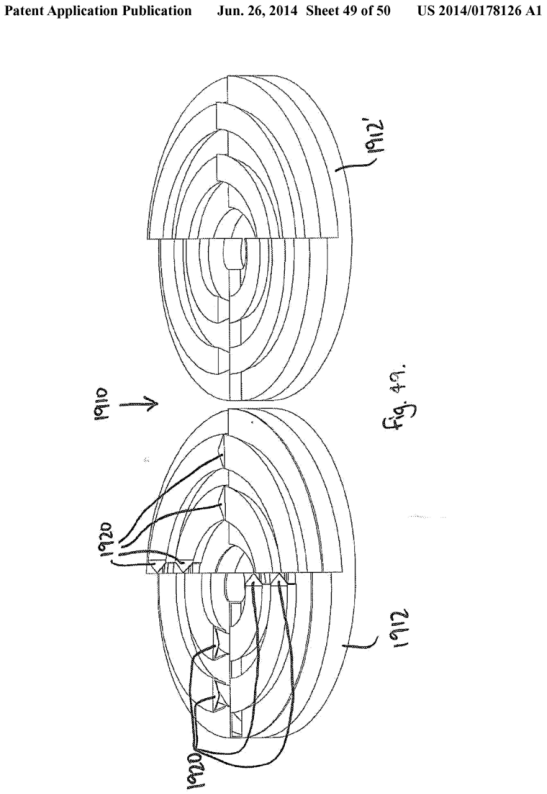


Fig. A.2.: Rotite patent figure

A.3 Rotor analysis tool

This tool designed and built by Aldo Vargas[1], it provides the data necessary to make a analysis of the behaviour of the rotor tuple. This tuple has three main components: *[electric motor, electronic speed control and propeller]*. Therefore we need the ability to measure at least four important parameters which are: voltage and current being consumed by the motor and electronic speed control, thrust produced by the propeller and the commanded PWM signal to the motor. The final tool can be seeing at figure A.3. In the electrical anal-

[1] `http://www.aldovargas.com/`

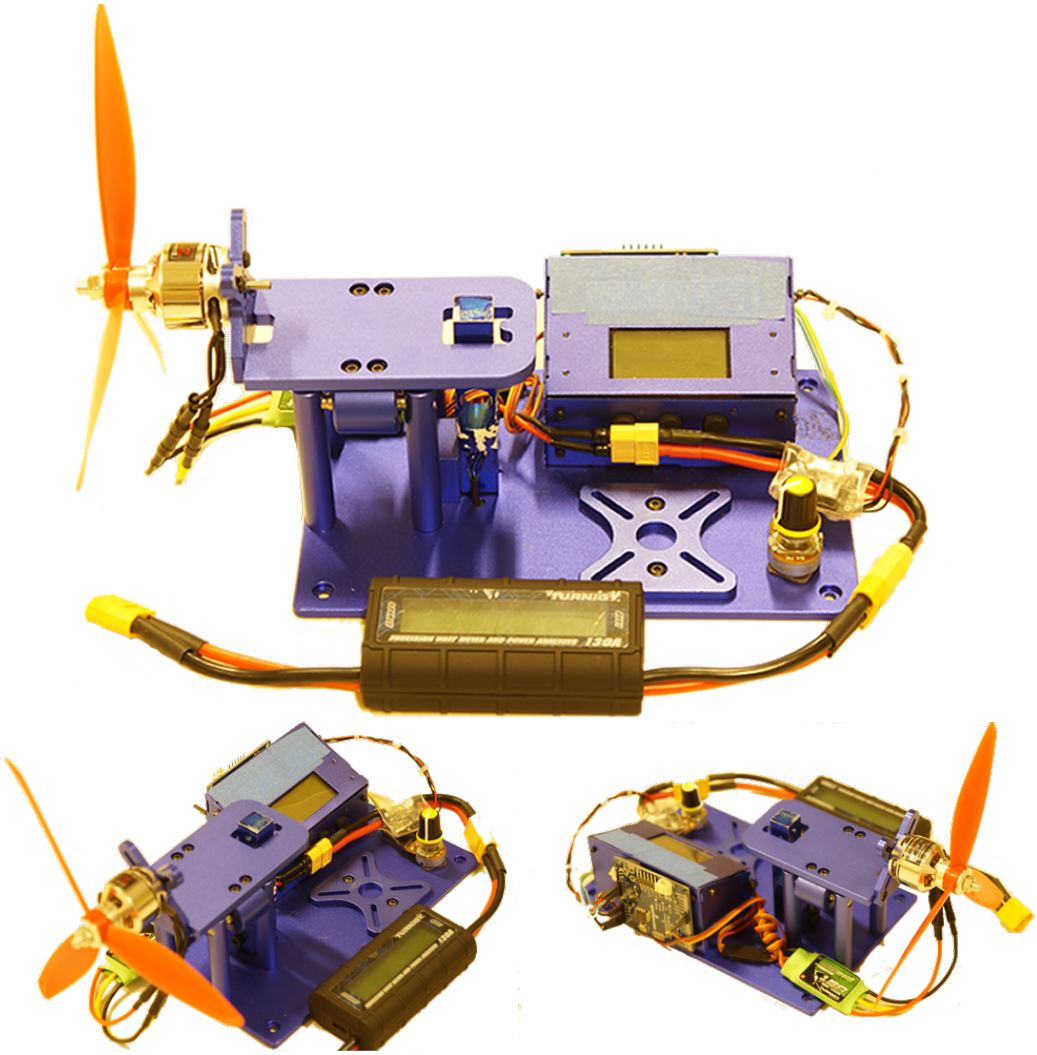


Fig. A.3.: Rotor analysis tool.

ysis, this tool compromises two different ways to obtain data. The redundancy here, helps to eliminate sensor errors by averaging the measurements from the two different sensing components. The first component is a APM power module A.4, being the second one a $180 Amps$ Power Analyser[2]. The load cell is a *Wheatstone Bridge* with a HX711 integrated circuit (A.5) that amplifies that signal and allows to measure the force being applied to a metal bar that is being pushed by the propeller thrust and the motor on top of bearings (to provide more support to the system). The input to the ESC, a PWM signal, is generated using the internal timers on a ATMEGA micro-controller. The advantage of using this methodology is that the PWM signal is identical to the one used by the MultiWii-based flight controllers, which then ensures that a specific PWM will correspond to the precise thrust generated by the rotor. This will be a added benefit to find the proper take-off PWM values that will translate into gains for the altitude controllers. The ATmega328P has three timers

[2] `http://www.hobbyking.co.uk/hobbyking/store/__75944__Turnigy_180A_Watt_Meter_and_Power_Analyzer.html`

Measurement	Component
Current	APM power module and Turnigy 180A Watt Meter and Power Analyser
Voltage	APM power module and Turnigy 180A Watt Meter and Power Analyser
Input signal	PWM generated using timers on an Arduino board
Thrust	Load cell with HX711 load cell amplifier

Tab. A.2.: Rotor analysis tool components.

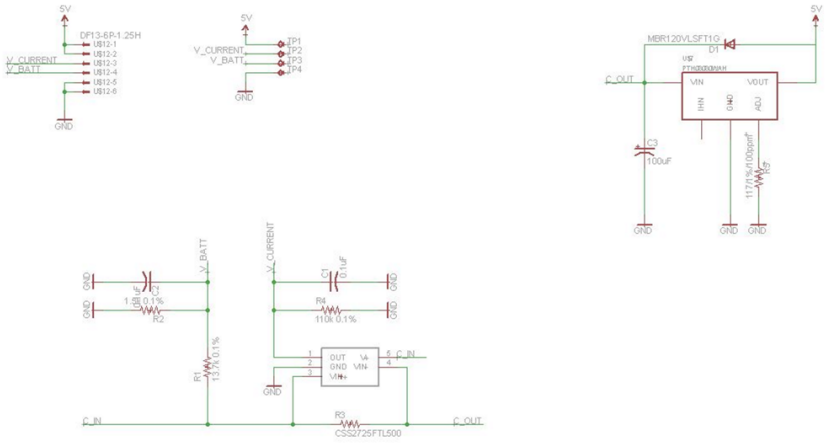


Fig. A.4.: APM power module schematics.

known as Timer 0, Timer 1, and Timer 2. Each timer has two output compare registers that control the PWM width for the timer's two outputs: when the timer reaches the compare register value, the corresponding output is toggled. The two outputs for each timer will normally have the same frequency, but can have different duty cycles (depending on the respective output compare register). Each of the timers has a prescaler that generates the timer clock by dividing the system clock by a prescale factor such as 1, 8, 64, 256, or 1024. The Arduino has a system clock of 16MHz and the timer clock frequency will be the system clock frequency divided by the prescale factor. Note that Timer 2 has a different set of prescale values from the other timers. The following code fragment will set up the fast PWM technique on pin 9:

```
1 TCCR1A |= (1<<WGM11); TCCR1A &= ~(1<<WGM10); TCCR1B |= (1<<WGM13);
2 TCCR1B &= ~(1<<CS11);
3 ICR1 |= 0x3FFF;
4 TCCR1A |= _BV(COM1A1);
```

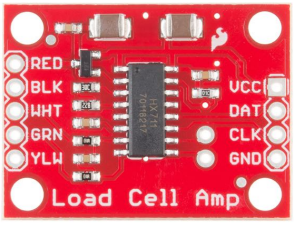


Fig. A.5.: Load cell amplifier HX711.

And the following snippet will read the voltage, current sensors and update the value of the PWM on pin 9:

```
1 void loop() {
2     Voltage = analogRead(VPin);
3     Current = analogRead(IPin);
4     val = analogRead(KnobPin);
5     val = map(val, 0, 1024, 1000, 2000);
6     pwm = map(val, 1000, 2000, 0, 255);
7     0CR1A = (val<<3);
8     delay(15);}
```

The entire code is available upon request. This is a open source project.

A.4 pyMultiWii

This Python module written by Aldo Vargas handles the MultiWii Serial Protocol (MSP) in order to send and receive data from MultiWii enabled flight controller units. This is a *text-based/console*, no Graphical User Interface (GUI), it works by sending and reading data from a computer serial port connected to a MultiWii board. This module is used for doing different requests to my flight controllers in order to control them using the *DronePilot* framework Vargas et al., 2014.

A.4.1 MultiWii Serial Protocol

MSP is a protocol designed by the MultiWii community (*MultiWii*), with the idea to be light, generic, bit wire efficient, secure. The MSP data frames are structured as showed on figure A.6. The general format of a MSP message is: $< header >, < direction >, < size >, <$

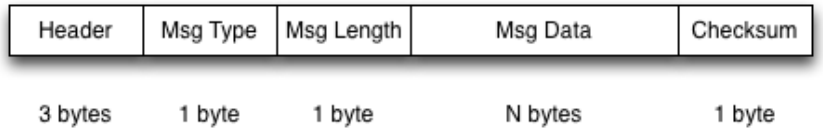


Fig. A.6.: MSP data frame.

$command >, < crc >$ Where:

- header: the ASCII characters $\$M$
- direction: the ASCII character $<$ if the message goes to the MultiWii board or $>$ if the message is coming from the board
- size: number of data bytes, binary. Can be zero as in the case of a data request to the board
- command: message_id of MSP
- data: values to be sent. UINT16 values are LSB first
- crc: (cyclic redundancy check) checksum, XOR of $< size >, < command >$ and each data byte into a zero sum

The current commands implemented on the pyMultiWii Python module are showed on Tab. A.3.

Command	**Message_id**	**Data**	**Comments**
MSP_RAW_IMU	102	accx, accy, accz, gx, gy, gz	Raw Accelerometer and Gyroscope measurements
MSP_MOTOR	104	motor*8	PWM being written to the 8 outputs of the board
MSP_RC	105	rcData	Channels from the radio control
MSP_ATTITUDE	108	angX, angY, heading	Orientation of the board
MSP_SET_RAW_RC	200	rcData	This request is used to inject RC channel via MSP

Tab. A.3.: pyMultiWii MSP implemented commands.

A.4.2 Data flow

There is basically three types of messages to interact with a MultiWii board. Those are *command*, *request* and *response*. *Command* is an incoming message without implicit outgoing

response from the board, *request* is an incoming message with implicit outgoing response while *response* is the outgoing message resulting from an incoming request. If, e.g., the orientation of the board is needed, then a message with type *request* and ID = 108 must be created and then sent to the board, after being sent, the board will reply with a *response*. The function definition of how to send a request to the board:

```
def sendCMD(self, data_length, code, data):
    checksum = 0
    total_data = ['$', 'M', '<', data_length, code] + data
    for i in struct.pack('<2B%dh' % len(data), *total_data[3:len(total_data)]):
        checksum = checksum ^ ord(i)
    total_data.append(checksum)
    try:
        b = None
        b = self.ser.write(struct.pack('<3c2B%dhB' % len(data), *total_data))
    except Exception, error:
        print "\n\nError in sendCMD."
        print "("+str(error)+")\n\n"
        pass
```

The next code snippet is just a part on how to mix a request message and get the response at the same time, its noticed how the first action is to send a request to the board, then there is an infinite loop that waits until the MSP header is found on the read serial buffer:

```
start = time.clock()
self.sendCMD(0,cmd,[])
while True:
    header = self.ser.read()
    if header == '$':
        header = header+self.ser.read(2)
        break
datalength = struct.unpack('<b', self.ser.read())[0]
code = struct.unpack('<b', self.ser.read())
data = self.ser.read(datalength)
temp = struct.unpack('<'+'h'*(datalength/2),data)
elapsed = time.clock() - start
```

```
13 self.ser.flushInput()
14 self.ser.flushOutput()
```

A.4.3 Performance

The entire implementation of this protocol does not include a *sleep* function, which means that is very fast and efficient, the rate of communication would then depend on the computer and the board capabilities. The module is also designed to be extremely simple to use, the next code will request and print (to the host computer) the orientation of the a MultiWii board connected to a USB port:

```
1 from pyMultiwii import MultiWii
2 from sys import stdout
3
4 if __name__ == "__main__":
5     board = MultiWii("/dev/ttyUSB0")
6     try:
7         while True:
8             board.getData(MultiWii.ATTITUDE)
9             print board.attitude
10    except Exception,error:
11        print "Error on Main: "+str(error)
```

This module can achieve communication back and forth of $300hz$, this was achieved using a Naze32 (32bits micro-controller) board and a Odroid U3. The next lines shows the response of the code above, for a few seconds, its noted that the elapsed time of communication is 0.016 seconds, this is around $62.5hz$, this was taken using a MultiWii AIO 2.0 (8bits micro-controller) board and a Raspberry Pi:

```
1 {'timestamp': 1417432436.878697, 'elapsed': 0.016, 'angx': -26.8, 'angy': -24.8, 'heading': -84.0}
2 {'timestamp': 1417432436.894663, 'elapsed': 0.016, 'angx': -26.8, 'angy': -24.7, 'heading': -84.0}
3 {'timestamp': 1417432436.910673, 'elapsed': 0.016, 'angx': -26.7, 'angy': -24.8, 'heading': -84.0}
4 {'timestamp': 1417432436.926812, 'elapsed': 0.016, 'angx': -26.7, 'angy': -24.7, 'heading': -84.0}
5 {'timestamp': 1417432437.134683, 'elapsed': 0.016, 'angx': -26.6, 'angy': -24.2, 'heading': -85.0}
6 {'timestamp': 1417432437.150524, 'elapsed': 0.016, 'angx': -26.6, 'angy': -24.1, 'heading': -85.0}
7 {'timestamp': 1417432437.166525, 'elapsed': 0.016, 'angx': -26.6, 'angy': -24.1, 'heading': -85.0}
```

A.4.4 Influence

This Python module has being used in several projects, not just the ones developed for this document, but every week (approximately) there is a developer e-mail communication either thanking the creator of the module and/or asking questions about it. Its important to notice that this module is also released on a GNU General Public Licence. The usage of Github [3] has being a key factor in order to make this module publicly available.

Fig. A.7.: pyMultiWii - Github project Stars

A.5 Computer Vision techniques

This repository [4] written by Aldo Vargas handles provides computer vision Python-code examples that make usage of the openCV framework Bradski, 2000 and such examples are targeted to work on credit-card-sized computers with relative low CPU power. The repository contains methods for doing the next computer vision tasks:

- Colour tracking (in several formats)
- Face detection (only for front face) (Fig. A.8)
- Motion detection
- Object detection (using region of interest)
- Time-lapse photography using openCV
- Common examples on how to open image and video files

It also contains comprehensive documentation on how to prepare a companion computer (RaspberryPi) for tasks such as computer vision. This repository triggered a technical reviewer position for the author of this thesis and the work can be seen at Pajankar et al., 2015. Such citation is a book that provides the skills needed to successfully design and implement Raspberry Pi and Python-based computer vision projects.

[3] http://www.github.com/
[4] https://github.com/alduxvm/rpi-opencv

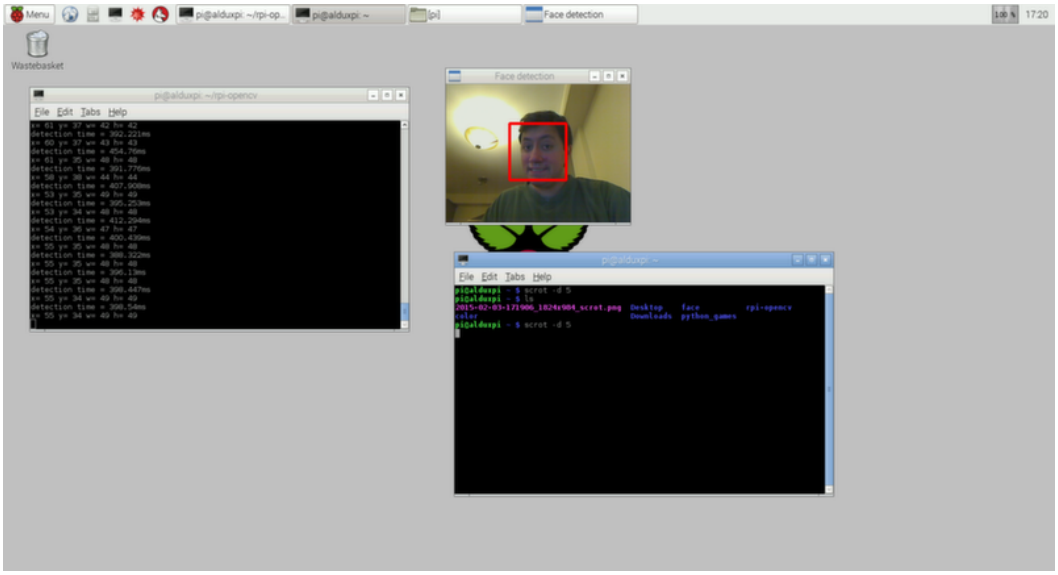


Fig. A.8.: Screen shot of the face detection algorithm working in a Raspberry Pi

A.5.1 Influence

This repository has being used in several projects, not just the ones developed for this thesis as a method for estimating the slung load position. Therefore is one of the authors most popular repositories A.9. Its important to notice that this module is also released on a GNU General Public Licence.

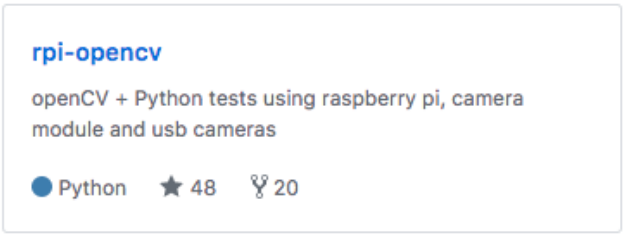


Fig. A.9.: rpi-opencv - Github project Stars

Bibliography

Adigbli, Patrick, Christophe Grand, Jean-Baptiste Mouret, and Stéphane Doncieux (2007). „Nonlinear Attitude and Position Control of a Micro Quadrotor using Sliding Mode and Backstepping Techniques“. In: *European Micro Air Vehicle Conference and Flight Competition* September, pp. 17–21 (cit. on p. 10).

Alexanderson, E. F. W., M. A. Edwards, and C. H. Willis (1938). „Electronic speed control of motors“. In: *Electrical Engineering* 57.6, pp. 343–354. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6431329 (cit. on p. 39).

Alpaydn, Ethem (2014). *Introduction to machine learning*. Vol. 1107, pp. 105–128. arXiv: 0904.3664v1 (cit. on p. 105).

Ampatis, Christos and Evangelos Papadopoulos (2014). „Parametric design and optimization of multi-rotor aerial vehicles“. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6266–6271. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6907783 (cit. on p. 8).

Anastasiou, Athanasios, Charalambos Tsirmpas, Alexandros Rompas, Kostas Giokas, and Dimitris Koutsouris (2013). „3D printing: Basic concepts mathematics and technologies“. In: *13th IEEE International Conference on BioInformatics and BioEngineering*. IEEE, pp. 1–4. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6701672 (cit. on p. 28).

Anderson, D. (2011). „Evolutionary algorithms in airborne surveillance systems: image enhancement via optimal sightline control“. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 225.10, pp. 1097–1108. URL: http://pig.sagepub.com/lookup/doi/10.1177/0954410011413014 (cit. on p. 130).

Anderson, David P., Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer (2002). „SETI @ home: an experiment in public-resource computing“. In: *Communications of the ACM* 45.11, pp. 56–61.

Anderson, James A., Edward. Rosenfeld, and Andras. Pellionisz (1988). *Neurocomputing*. MIT Press (cit. on p. 113).

Antonelli, Gianluca (2015). „Robotic Research: Are We Applying the Scientific Method?“ English. In: *Frontiers in Robotics and AI* 2. URL: http://journal.frontiersin.org/article/10.3389/frobt.2015.00013/abstract (cit. on p. 56).

Antonelo, Eric Aislan (2011). „Reservoir Computing Architectures for Modeling Robot Navigation Systems“. PhD thesis. Universiteit Gent (cit. on pp. 18, 123).

Aoustin, Yannick and Alexander Formal'sky (2003). „Simple anti-swing feedback control for a gantry crane“. In: *Robotica* 21.6, pp. 655–666 (cit. on p. 172).

Apvrille, Ludovic, Yves Roudier, and Tullio Joseph Tanzi (2015). „Autonomous drones for disasters management: Safety and security verifications“. In: *2015 1st URSI Atlantic Radio Science Conference (URSI AT-RASC)*. IEEE, pp. 1–2. URL: http://ieeexplore.ieee.org/document/7303086/ (cit. on p. 1).

Atmosphere, U S Standard (1964). „Standard Atmosphere“. In: *Nature* 201.4919, pp. 537–538 (cit. on p. 39).

Barmpounakis, Emmanouil N., Eleni I. Vlahogianni, and John C. Golias (2017). „Unmanned Aerial Systems for transportation engineering: Current practice and future challenges“. In: *International Journal of Transportation Science and Technology* (cit. on p. 2).

Beard, Randal W. (2008). „Quadrotor Dynamics and Control“. In: *Brigham Young University*, pp. 1–47. URL: http://image.ednchina.com/GROUP/uploadfile/201304/20130429210226589.pdf (cit. on p. 10).

Beaufays, Françoise and Eric a. Wan (1994). „Relating Real-Time Backpropagation and Backpropagation - Through - Time: An Application of Flow Graph Interreciprocity“. In: *Neural Computation* 6, pp. 296–306 (cit. on p. 120).

Bengio, Yoshua, Paolo Frasconi, and Patrice Simard (1993). „The problem of learning long-term dependencies in recurrent networks“. In: *IEEE International Conference on Neural Networks Conference Proceedings*. Vol. 1993-Janua, pp. 1183–1188 (cit. on p. 17).

Benjamin, Medea (2013). *Drone warfare : killing by remote control*. Verso, p. 246 (cit. on p. 1).

Bian, Xinqian and Chunhui Mou (2011). „Identification of non-linear dynamic model of UUV based on ESN neural network“. In: *Proceedings of the 30th Chinese Control Conference*, pp. 1432–1437 (cit. on p. 19).

Bisgaard, Morten (2007). „Modeling, estimation, and control of helicopter slung load system“. PhD thesis. Aalborg University.

Bisgaard, Morten, Jan Dimon Bendtsen, and Anders Cour-Harbo (2006). „Modelling of Generic Slung Load System“. In: *Proceedings of AIAA Modeling and Simulation Technologies Conference* 32.2, pp. 1–20 (cit. on p. 12).

Bisgaard, Morten, Morten Bisgaard, Anders La Cour-harbo, and Jan Dimon Bendtsen (2010). „Adaptive Control System for Autonomous Helicopter Slung Load Operations“. In: *CONTROL ENGINEERING PRACTICE*, pp. 800–811. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.303.2541 (cit. on p. 13).

Bishop, Robert H. (2007). *Mechatronic Systems, Sensors, and Actuators: Fundamentals and Modeling*. Vol. 19. CRC Press, p. 692. URL: https://books.google.com/books?id=3UGQsi6VamwC&pgis=1 (cit. on p. 9).

Blum, Adam and Adam (1992). *Neural networks in C++ : an object-oriented framework for building connectionist systems*. Wiley, p. 213 (cit. on p. 17).

Boden, Mikael (2001). „A guide to recurrent neural networks and backpropagation“. In: *Electrical Engineering* 2, pp. 1–10 (cit. on p. 15).

Bouabdallah, Samir (2007). „Design and Control of Quadrotors With Application To Autonomous Flying“. In: *École Polytechnique Fédérale De Lausanne, À La Faculté Des Sciences Et Techniques De L'Ingénieur* 3727.3727, p. 61. URL: http://biblion.epfl.ch/EPFL/theses/2007/3727/EPFL_TH3727.pdf (cit. on pp. 9, 10).

Bradski, G (2000). „The OpenCV Library“. In: *Dr Dobbs Journal of Software Tools* 25, pp. 120–125. URL: http://opencv.willowgarage.com (cit. on pp. 156, 207).

Bresciani, Tommaso (2008). „Modelling , Identification and Control of a Quadrotor Helicopter“. In: *English* 4.October, p. 213. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Modelling+,+Identification+and+Control+of+a+Quadrotor+Helicopter#0 (cit. on p. 9).

Bronz, Murat, Jean Marc Moschetta, Pascal Brisset, and Michel Gorraz (2009). „Towards a Long Endurance MAV“. en. In: *International Journal of Micro Air Vehicles* 1.4, pp. 241–254. URL: http://multi-science.atypon.com/doi/abs/10.1260/175682909790291483 (cit. on pp. 9, 46).

Brunak, S and B Lautrup (1990). *Neural Networks Computers with Intuition*. WORLD SCIENTIFIC. URL: http://www.worldscientific.com/worldscibooks/10.1142/0878 (cit. on p. 110).

Buonomano, D V and M M Merzenich (1995). „Temporal information transformed into a spatial code by a neural network with realistic properties.“ In: *Science (New York, N.Y.)* 267.5200, pp. 1028–30. URL: http://www.ncbi.nlm.nih.gov/pubmed/7863330 (cit. on p. 124).

Burghartz, J. (2013). *Guide to State-of-the-Art Electron Devices*. Ed. by Joachim N. Burghartz. Chichester, UK: John Wiley and Sons, Ltd, pp. 1–328. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6963180 (cit. on pp. 8, 47).

Campolucci, Paolo, Aurelio Uncini, and Francesco Piazza (1996). „Causal back propagation through time for locally recurrent neural networks“. In: *Proceedings IEEE International Symposium on Circuits and Systems*. Vol. 3 (cit. on p. 18).

Carbonell, J and Jaime G. (1990). *Machine learning : paradigms and methods*. MIT Press, pp. 1–9 (cit. on p. 16).

Chan, Brodie, Hong Guan, Jun Jo, and Michael Blumenstein (2015). „Towards UAV-based bridge inspection systems: a review and an application perspective“. In: *Structural Monitoring and Maintenance* 2.3, pp. 283–300. URL: http://koreascience.or.kr/journal/view.jsp?kj=E1TPK5&py=2015&vnc=v2n3&sp=283 (cit. on p. 1).

Chen, X, Y Wang, X Liu, M J F Gales, and P C Woodland (2014). „Efficient GPU-based Training of Recurrent Neural Network Language Models Using Spliced Sentence Bunch“. In: (cit. on p. 117).

Cheng, Eric (2015). *Aerial photography and videography using drones*, p. 269 (cit. on p. 1).

Chow, Tommy W S and Yong Fang (1998). „A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics“. In: *IEEE Transactions on Industrial Electronics* 45.1, pp. 151–161 (cit. on pp. 17, 122).

Cicolani, L. S. and G. Kanning (1986). „General equilibrium characteristics of a dual-lift helicopter system“. In: (cit. on p. 11).

Cicolani, Luigi S., Gerd Kanning, and Robert Synnestvedt (1995). „Simulation of the Dynamics of Helicopter Slung Load Systems“. In: *Journal of the American Helicopter Society* 40, p. 44 (cit. on pp. 3, 11).

Cios, Krzysztof J. and Mark E. Shields (1997). „The handbook of brain theory and neural networks“. In: *Neurocomputing* 16.3, pp. 259–261 (cit. on p. 122).

Cortes, Corinna and Vladimir Vapnik (1995). „Support-Vector Networks“. In: *Machine Learning* 20.3, pp. 273–297. arXiv: arXiv:1011.1669v3 (cit. on p. 15).

Coulouris, George, Jean Dollimore, and Tim Kindberg (2012). *Distributed Systems: Concepts and Design*. Vol. 4, p. 772. URL: http://www.amazon.com/dp/0321263545 (cit. on p. 10).

Crow, B.P., I. Widjaja, L.G. Kim, and P.T. Sakai (1997). „IEEE 802.11 Wireless Local Area Networks“. In: *IEEE Communications Magazine* 35.9, pp. 116–126. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=620533 (cit. on p. 65).

Csaji, Balazs Csanad (2001). „Approximation with Artificial Neural Networks Huub ten Eikelder“. In: (cit. on p. 116).

Das, Abhijit, Kamesh Subbarao, and Frank Lewis (2008). „Dynamic inversion of quadrotor with zero-dynamics stabilization“. In: *Proceedings of the IEEE International Conference on Control Applications*, pp. 1189–1194 (cit. on pp. 78, 133).

De La Torre, Gerardo, Tansel Yucelen, and Eric N. Johnson (2013a). „Neuropredictive Control and Trajectory Generation for Slung Load Systems“. In: *AIAA Infotech Aerospace (IA) Conference*. Reston, Virginia: American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/10.2514/6.2013-5044 (cit. on p. 19).

– (2013b). „Neuropredictive Control and Trajectory Generation for Slung Load Systems“. In: *AIAA Infotech@Aerospace (I@A) Conference*. Reston, Virginia: American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/10.2514/6.2013-5044 (cit. on p. 148).

Dempsey, M (2010). „US army unmanned aircraft systems roadmap 2010-2035“. In: *Federation Of American Scientists*.

Dietterich, Thomas G. (1986). „Learning at the Knowledge Level“. In: *Machine Learning* 1.3, pp. 287–315. URL: http://link.springer.com/10.1023/A:1022858530318 (cit. on p. 16).

Dijkstra, E. W. (1959). „A note on two problems in connexion with graphs“. In: *Numerische Mathematik* 1.1, pp. 269–271.

Doya, K. (1992). „Bifurcations in the learning of recurrent neural networks“. In: *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems* 6.4, pp. 1–4. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.5278&rep=rep1&type=pdf (cit. on p. 18).

Dukes, Theodor A. (1973). „Maneuvering Heavy Sling Loads Near Hover Part II: Some Elementary Maneuvers“. In: *Journal of the American Helicopter Society* 18.3, pp. 17–22. URL: http://openurl.ingenta.com/content/xref?genre=article&issn=2161-6027&volume=18&issue=3&spage=17 (cit. on p. 12).

Ede, J.D., Z.Q. Zhu, and D. Howe (2001). „Optimal split ratio for high-speed permanent magnet brushless DC motors“. English. In: *ICEMS'2001. Proceedings of the Fifth International Conference on Electrical Machines and Systems (IEEE Cat. No.01EX501)*. Vol. 2. Int. Acad. Publishers, pp. 909–912. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=971826 (cit. on pp. 8, 31).

El-Ferik, Sami, Asim H. Syed, Hanafy M. Omar, and Mohamed A. Deriche (2013). „Anti-Swing Nonlinear Path Tracking Controller for Helicopter Slung Load System“. In: *IFAC Proceedings Volumes* 46.30, pp. 134–141 (cit. on p. 148).

Elman, Jeffrey L. (1990). „Finding structure in time“. In: *Cognitive Science* 14.2, pp. 179–211 (cit. on p. 123).

Elmenreich, Wilfried (2002). „An introduction to sensor fusion“. In: *Austria: Vienna University Of Technology* February, pp. 1–28. URL: http://www.vmars.tuwien.ac.at/documents/intern/805/elmenreich_sensorfusionintro.pdf (cit. on pp. 9, 47).

Faille, D. and A.J.J. van der Weiden (1995). „Robust regulation of a flying crane“. In: *Proceedings of International Conference on Control Applications*. IEEE, pp. 494–499. URL: http://ieeexplore.ieee.org/document/555752/http://ieeexplore.ieee.org/ielx3/4243/12105/00555752.pdf?tp=&arnumber=555752&isnumber=12105%5Cnhttp://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=555752&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.j (cit. on pp. 11, 148).

Feaster, L., C. Poli, and R. Kirchhoff (1977). „Dynamics of a slung load“. In: *Journal of Aircraft* 14.2, pp. 115–121. URL: http://arc.aiaa.org/doi/10.2514/3.44578 (cit. on p. 12).

Featherstone, R. and D. Orin (2000). „Robot dynamics: equations and algorithms“. In: *Proceedings of the 2000 IEEE International Conference on Robotics and Automation* 1, pp. 826–834. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=844153 (cit. on p. 9).

Feldman, J.A., M.A. Fanty, and N.H. Goodard (1988). „Computing with structured neural networks“. In: *Computer* 21.3, pp. 91–103. URL: http://ieeexplore.ieee.org/document/34/ (cit. on p. 110).

Feng, Ying, Camille Alain Rabbath, Subhash Rakheja, and Chun-Yi Su (2015). „Adaptive controller design for generic quadrotor aircraft platform subject to slung load“. In: *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1135–1139. URL: http://ieeexplore.ieee.org/ielx7/7120003/7129089/07129434.pdf?tp=&arnumber=7129434&isnumber=7129089%5Cnhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7129434%5Cnhttp://ieeexplore.ieee.org/document/7129434/ (cit. on p. 13).

Fisher, R A (1936). „The use of multiple measurements in taxonomic problems“. In: *Annals of Eugenics* 7.2, pp. 179–188. arXiv: arXiv:1011.1669v3 (cit. on p. 105).

Friedenzohn, Daniel and Alexander Mirot (2013). „THE FEAR OF DRONES: PRIVACY AND UNMANNED AIRCRAFT“. In: 3.5 (cit. on p. 3).

Frost, Chad, mark Tischler, Mike Bielefield, and Troy L (2000). „Design and test of flight control laws for the Kaman Burro unmanned aerial vehicle“. In: *Atmospheric Flight Mechanics Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics. URL: http://arc.aiaa.org/doi/10.2514/6.2000-4205 (cit. on p. 148).

Fusato, Dario, Giorgio Guglieri, and Roberto Celi (2001). „Flight Dynamics of an Articulated Rotor Helicopter with an External Slung Load“. In: *Journal of the American Helicopter Society* 46.1, pp. 3–13. URL: http://openurl.ingenta.com/content/xref?genre=article&issn=2161-6027&volume=46&issue=1&spage=3 (cit. on p. 12).

Ghahramani, Zoubin (2008). „Bayesian Methods for Artificial Intelligence and Machine Learning“. In: *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands: IOS Press, p. 8. URL: http://dl.acm.org/citation.cfm?id=1567281.1567285 (cit. on p. 106).

Gieras, Jacek F. (2002). *Permanent Magnet Motor Technology: Design and Applications, Second Edition,* CRC Press, p. 616. URL: https://books.google.com/books?hl=en&lr=&id=u_NiSnZeLQQC&pgis=1 (cit. on p. 31).

Gieras, Jacek F (2014). „Design of Permanent Magnet Brushless Motors for High Speed Applications“. In: (cit. on p. 30).

Glad, Torkel. and Lennart. Ljung (2000). *Control theory : multivariable and nonlinear methods*. Taylor and Francis, p. 467 (cit. on p. 78).

Gonzalez-Olvera, Marcos A. and Yu Tang (2010). „Black-box identification of a class of nonlinear systems by a recurrent neurofuzzy network“. In: *IEEE Transactions on Neural Networks* 21.4, pp. 672–679 (cit. on pp. 17, 19, 71).

Gosavi, Abhijit (2003). *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Vol. 1, pp. 1–6. URL: http://books.google.com/books?hl=it&lr=&id=XqKyW9U3PWAC&pgis=1.

Guenard, Nicolas, Tarek Hamel, and Laurent Eck (2006). „Control laws for the tele operation of an unmanned aerial vehicle known as an X4-flyer“. In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 3249–3254.

Guo, Jiang (2013). „BackPropagation Through Time“. In: *Manuscript* 1, pp. 1–6 (cit. on p. 120).

Gupta, N. K. and Jr. Bryson A. E. (1976). „Near-hover control of a helicopter with a hanging load“. In: (cit. on p. 12).

Hahn, Hubert (2002). *Rigid Body Dynamics of Mechanisms: 1 Theoretical Basis*. Springer Science and Business Media, p. 336. URL: https://books.google.com/books?id=MqrN3KY7o6MC&pgis=1 (cit. on p. 9).

Hansen, N and S Kern (2004). „Evaluating the CMA Evolution Strategy on Multimodal Test Functions“. In: *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature - PPSN VIII* 3242/2004.0, pp. 282–291 (cit. on p. 130).

Hansen, Nikolaus (2016). „The CMA Evolution Strategy: A Tutorial“. In: arXiv: 1604.00772. URL: http://arxiv.org/abs/1604.00772 (cit. on p. 130).

Hansen, Nikolaus and Andreas Ostermeier (1996). „Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation“. In: *IEEE Intern. Conf. on Evolutionary Computation*, pp. 312–317. arXiv: arXiv:1011.1669v3 (cit. on p. 130).

– (2001). „Completely Derandomized Self-Adaptation in Evolution Strategies“. In: *Evolutionary Computation* 9.2, pp. 159–195. URL: http://www.mitpressjournals.org/doi/abs/10.1162/106365601750190398 (cit. on p. 130).

Hebb, D. O. (1949). „The Organization of Behaviour“. In: *Organization*, p. 62 (cit. on p. 112).

Heng, Lionel, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys (2011). „Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing“. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2472–2477 (cit. on p. 10).

Hertz, J, A Krogh, and R G Palmer (1991). *Introduction to the Theory of Neural Computation*. Vol. 1, p. 327. URL: http://books.google.com/books?id=9a_SyUG-A24C&pgis=1 (cit. on p. 114).

Hoh, Roger H., Robert K. Heffley, and David G. Mitchell (2006). „Development of Handling Qualities Criteria for Rotorcraft with Externally Slung Loads“. In: (cit. on p. 12).

Holzmann, G (2009). „Reservoir computing: a powerful black-box framework for nonlinear audio processing“. In: *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx09)*, pp. 1–8. URL: http://dafx09.como.polimi.it/proceedings/papers/paper_23.pdf (cit. on p. 17).

Honegger, Dominik, Pierre Greisen, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys (2012). „Real-time velocity estimation based on optical flow and disparity matching“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* 1, pp. 5177–5182. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6385530 (cit. on p. 10).

Huang, H. M. (2004). „Autonomy Levels for Unmanned Systems (ALFUS) Framework Volume I : Terminology Unmanned Systems Working Group Participants 1 National Institute of Standards and Technology“. In: *Framework* I.September, p. 29.

Ireland, Murray, Aldo Vargas, and David Anderson (2015). „A Comparison of Closed-Loop Performance of Multirotor Configurations Using Non-Linear Dynamic Inversion Control“. In: *Aerospace* 2.2, pp. 325–352. URL: http://www.mdpi.com/2226-4310/2/2/325/ (cit. on pp. 8, 24, 188).

Ireland, Murray L. (2014). „Investigations in multi-resolution modelling of the quadrotor micro air vehicle“. In: (cit. on p. 81).

Iwan Solihin, M. and Wahyudi (2007). „Sensorless anti-swing control strategy for automatic gantry crane system: Soft sensor approach“. In: *2007 International Conference on Intelligent and Advanced Systems, ICIAS 2007*, pp. 992–996 (cit. on p. 172).

Jaeger, H (2002a). „Short term memory in echo state networks“. In: *GMD Report 152*, p. 60. URL: papers://78a99879-71e7-4c85-9127-d29c2b4b416b/Paper/p14153%5Cnhttp://neuron-ai.tuke.sk/$\sim$bundzel/diploma_theses_students/2006/MartinSramko-EchoStateNNinPrediction/STMEchoStatesTechRep.pdf (cit. on p. 129).

Jaeger, Herbert (2001). „The "echo state" approach to analysing and training recurrent neural networks“. In: *GMD Report* 148, pp. 1–47 (cit. on pp. 17, 125, 128).

– (2002b). „Adaptive Nonlinear System Identification with Echo State Networks“. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 593–600. URL: http://books.nips.cc/nips15.html (cit. on pp. 19, 129).

– (2005a). „A tutorial on training recurrent neural networks , covering BPPT , RTRL , EKF and the " echo state network " approach“. In: *ReVision* 2002, pp. 1–46. URL: http://www.mendeley.com/catalog/tutorial-training-recurrent-neural-networks-covering-bppt-rtrl-ekf-echo-state-network-approach/ (cit. on pp. 17, 120).

– (2005b). „A tutorial on training recurrent neural networks , covering BPPT , RTRL , EKF and the " echo state network " approach“. In: *ReVision* 2002, pp. 1–46. URL: http://www.mendeley.com/catalog/tutorial-training-recurrent-neural-networks-covering-bppt-rtrl-ekf-echo-state-network-approach/ (cit. on p. 18).

– (2007). „Discovering multiscale dynamical features with hierarchical Echo State Networks“. In: 10 (cit. on p. 18).

Jaeger, Herbert and Harald Haas (2004). „Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication“. In: *Science* 304.78, pp. 78–80. arXiv: arXiv:1011.1669v3. URL: http://science.sciencemag.org/content/304/5667/78.short%5Cnhttp://www.sciencemag.org/cgi/doi/10.1126/science.1091277 (cit. on pp. 18, 123, 124).

Jain, L. C. and L. R. Medsker (2000). *Recurrent neural networks : design and applications*. CRC Press, p. 392 (cit. on p. 17).

Jang, G.H. and M.G. Kim (2004). „A bipolar-starting and unipolar-running method to drive an HDD spindle motor at high speed with large starting torque“. In: *APMRC 2004 Asia-Pacific Magnetic Recording Conference, 2004.* IEEE, pp. 36–37. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1521938 (cit. on pp. 8, 31).

Jiang, Fei, Hugues Berry, and Marc Schoenauer (2008). „Supervised and Evolutionary Learning of Echo State Networks“. In: URL: https://hal.inria.fr/inria-00337235 (cit. on pp. 129, 130).

Jordan, Michael I. (1986). „Attractor dynamics and parallelism in a connectionist sequential machine“. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 531–546. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Attractor+dynamics+and+parallelism+in+a+connectionist+sequential+machine#0 (cit. on p. 123).

Khalid, Attir, John Huey, William Singhose, Jason Lawrence, and David Frakes (2006). „Human Operator Performance Testing Using an Input-Shaped Bridge Crane“. In: *Journal of Dynamic Systems, Measurement, and Control* 128.4, p. 835. URL: http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1411554 (cit. on p. 148).

Khalil, H K (2002). *Nonlinear Systems, Third Edition*. URL: http://cdsweb.cern.ch/record/1173048 (cit. on pp. 10, 80).

Kim, Nikolai Vladimirovich and Mikhail Alekseevich Chervonenkis (2015). „Situation Control of Unmanned Aerial Vehicles for Road Traffic Monitoring“. In: *Modern Applied Science* 9.5, p. 1. URL: http://ccsenet.org/journal/index.php/mas/article/view/46350 (cit. on p. 1).

Kocabas, S., Jaime G. Carbonell, Kenneth De Jong, et al. (1991). „A review of learning“. In: *The Knowledge Engineering Review* 6.03, p. 195. URL: http://www.journals.cambridge.org/abstract_S0269888900005804 (cit. on p. 16).

Kotsiantis, Sotiris B. (2007). „Supervised machine learning: A review of classification techniques“. In: *Informatica* 31, pp. 249–268 (cit. on p. 17).

Kurose, James F. and Keith W. Ross (2013). *Computer Networking A Top-Down Approach*. 5, p. 4. arXiv: 8177588788 (cit. on p. 65).

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). „Deep learning“. In: *Nature* 521.7553, pp. 436–444. arXiv: arXiv:1312.6184v5. URL: http://dx.doi.org/10.1038/nature14539 (cit. on p. 14).

Lehmann, E L and G Casella (1998). „Theory of Point Estimation“. In: *Design* 41.3, p. 589. URL: http://www.amazon.com/dp/0387985026 (cit. on p. 126).

Levenberg, K and K Levenberg (1944). „A Method for the Solution of Certain Problems in Least Squares“. In: *Quart. Appl. Math.* Vol. 2. 2, pp. 164–168 (cit. on p. 123).

Liu, D. (2001). „Open-loop training of recurrent neural networks for nonlinear dynamical system identification“. In: *IJCNN01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. Vol. 2. IEEE, pp. 1215–1220. URL: http://ieeexplore.ieee.org/document/939534/ (cit. on p. 19).

Loshchilov, Ilya and Frank Hutter (2016). „CMA-ES for Hyperparameter Optimization of Deep Neural Networks“. In: arXiv: 1604.07269. URL: http://arxiv.org/abs/1604.07269 (cit. on p. 130).

Luigi, Cicolani, Kanning Gerd (1992). *Equations of motions of slung-load systems*. URL: http://search.library.utoronto.ca/details?4139736&uuid=4777f6b9-68ae-49cb-82e1-5ae58eac1948 (cit. on p. 12).

Lukoševičius, M (2012). „A practical guide to applying echo state networks“. In: *Neural Networks: Tricks of the Trade, Reloaded*, pp. 659–686. arXiv: 1406.6247 (cit. on p. 124).

Lukoševičius, Mantas and Herbert Jaeger (2009). „Reservoir computing approaches to recurrent neural network training“. In: *Computer Science Review* 3.3, pp. 127–149 (cit. on pp. 18, 129).

Maass, Wolfgang, Thomas Natschläger, and Henry Markram (2002). „Real-time computing without stable states: a new framework for neural computation based on perturbations.“ In: *Neural computation* 14.11, pp. 2531–2560. arXiv: arXiv:1011.1669v3 (cit. on pp. 18, 123).

Maass, Wolfgang, Thomas Natschläger, and Henry Makram (2003). „A Model for Real-Time Computation in Generic Neural Microcircuits“. In: *Advances in Neural Information Processing Systems*, pp. 213–220 (cit. on p. 124).

MacQueen, J B (1967). „Some Methods for classification and Analysis of Multivariate Observations“. In: *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*. Vol. 1, pp. 281–297. URL: http://projecteuclid.org/euclid.bsmsp/1200512992 (cit. on p. 15).

Magnussen, Øyvind, Geir Hovland, and Morten Ottestad (2014). „Multicopter UAV design optimization“. In: *MESA 2014 - 10th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, Conference Proceedings* (cit. on p. 8).

Magnussen, Øyvind, Morten Ottestad, and Geir Hovland (2015). „Multicopter design optimization and validation“. In: *Modeling, Identification and Control* 36.2, pp. 67–79 (cit. on p. 8).

Mahony, R., T. Hamel, and J.-M. Pflimlin (2005). „Complementary filter design on the special orthogonal group SO(3)“. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 1477–1484. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1582367 (cit. on pp. 9, 49, 50).

Mahony, Robert, Vijay Kumar, and Peter Corke (2012a). „Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor“. In: *IEEE Robotics and Automation Magazine* 19.3, pp. 20–32 (cit. on p. 9).

– (2012b). „Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor“. In: *IEEE Robotics and Automation Magazine* 19.3, pp. 20–32 (cit. on p. 10).

– (2012c). „Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor“. In: *IEEE Robotics and Automation Magazine* 19.3, pp. 20–32.

Marquardt, Donald W. (1963). „An Algorithm for Least-Squares Estimation of Nonlinear Parameters“. In: *Journal of the Society for Industrial and Applied Mathematics* 11.2, pp. 431–441. arXiv: arXiv:1011.1669v3 (cit. on p. 123).

McCulloch, Warren S. and Walter Pitts (1943). „A logical calculus of the ideas immanent in nervous activity“. In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. URL: http://link.springer.com/10.1007/BF02478259 (cit. on p. 110).

Meier, Lorenz, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys (2011). „PIXHAWK: A system for autonomous flight using onboard computer vision“. English. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2992–2997. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5980229 (cit. on pp. 9, 10, 46, 48, 59, 77).

Mellinger, Daniel (2012). „Trajectory generation and control for quadrotors“. PhD thesis. University of Pennsylvania, p. 136 (cit. on pp. 9, 10, 80).

Mellinger, Daniel, Nathan Michael, and Vijay Kumar (2014). „Trajectory generation and control for precise aggressive maneuvers with quadrotors“. In: *Springer Tracts in Advanced Robotics*. Vol. 79, pp. 361–373 (cit. on pp. 10, 154).

Micale, Edward C. and Corrado Poli (1973). „Dynamics of Slung Bodies Utilizing a Rotating Wheel for Stability“. In: *Journal of Aircraft* 10.12, pp. 760–763 (cit. on p. 12).

Michael, Nathan, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar (2010). „The GRASP multiple micro-UAV testbed“. In: *IEEE Robotics and Automation Magazine* 17.3, pp. 56–65 (cit. on pp. 8, 10).

Miller, Derek Scott (2011). „Open loop system identification of a micro quadrotor helicopter from closed loop data“. PhD thesis, p. 161. URL: http://ezproxy.net.ucf.edu/login?url=http://search.proquest.com/docview/923785954?accountid=10003%5Cnhttp://sfx.fcla.edu/ucf?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&genre=dissertations+&+theses&sid=ProQ:ProQuest+Dissertations+&+The (cit. on pp. 10, 79).

Mistler, V., A. Benallegue, and N.K. M'Sirdi (2001). „Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback“. In: *Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication. ROMAN 2001 (Cat. No.01TH8591)*. IEEE, pp. 586–593. URL: http://ieeexplore.ieee.org/document/981968/ (cit. on p. 78).

Mitchell, G. (2012). „The Raspberry Pi single-board computer will revolutionise computer science teaching“. In: *Engineering and Technology* 7, p. 26 (cit. on p. 60).

Mitchell, Tom M. (1982). „Generalization as search“. In: *Artificial Intelligence* 18.2, pp. 203–226 (cit. on p. 16).

Mitchell, Tom M (1997). *Machine Learning*. 1, pp. 417–433. arXiv: 0-387-31073-8. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0070428077 (cit. on pp. 15, 101).

Myers, Ware (1986). „INTRODUCTION TO EXPERT SYSTEMS.“ In: *IEEE Expert* 1.1, pp. 100–109 (cit. on p. 112).

Nawi, Nazri Mohd, Meghana R. Ransing, and Rajesh S. Ransing (2006). „An improved learning algorithm based on the Broyden-Fletcher-GoldfarbShanno (BFGS) method for back propagation neural networks“. In: *Proceedings - ISDA 2006: Sixth International Conference on Intelligent Systems Design and Applications*. Vol. 1, pp. 152–157 (cit. on p. 121).

Nelson, Robert C. (1998). *Flight stability and automatic control*. WCB/McGraw Hill, p. 441 (cit. on pp. 10, 79).

Neocleous, Costas and Christos Schizas (2002). „Artificial Neural Network Learning: A Comparative Review“. In: Springer Berlin Heidelberg, pp. 300–313. URL: http://link.springer.com/10.1007/3-540-46014-4_27 (cit. on p. 17).

Ni, Shaobo, Lei Liu, Zhishen Wang, and Zirui Wang (2011). „Predictive Control of Vehicle Based on Echo State Network“. In: *2011 International Conference on Intelligence Science and Information Engineering*, pp. 37–40. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5997371 (cit. on p. 19).

Norouzi, Mohammad, Mostafa Yaghobi, Mohammad Rezai Siboni, and Mahdi Jadaliha (2008). „Recursive line extraction algorithm from 2D laser scanner applied to navigation a mobile robot“. In: *2008 IEEE International Conference on Robotics and Biomimetics, ROBIO 2008*, pp. 2127–2132.

Ok, Kyel, Sameer Ansari, Billy Gallagher, et al. (2013). „Path planning with uncertainty: Voronoi Uncertainty Fields“. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4596–4601.

Oliveira, P., I. Kaminer, and A. Pascoal (2000). „Navigation system design using time-varying complementary filters“. In: *IEEE Transactions on Aerospace and Electronic Systems* 36.4, pp. 1099–1114. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=892661 (cit. on pp. 10, 50).

Omar, Hanafy M. (2009). „New fuzzy-based anti-swing controller for helicopter slung-load system near hover“. In: *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*. IEEE, pp. 474–479. URL: http://ieeexplore.ieee.org/document/5423159/ (cit. on p. 148).

Omatu, Sigeru, Marzuki Khalid, and Rubiyah Yusof (1996). „Neuro-Control Applications“. In: Springer London, pp. 171–243. URL: http://link.springer.com/10.1007/978-1-4471-3058-1_5 (cit. on p. 108).

Pajankar, Ashwin and Aldo Vargas (2015). *Raspberry Pi computer vision programming : design and implement your own computer vision applications with the Raspberry Pi*. Packt Publishing (cit. on pp. 193, 207).

Palunko, Ivana, Rafael Fierro, and Patricio Cruz (2012). „Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach“. In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2691–2697. URL: http://ieeexplore.ieee.org/document/6225213/ (cit. on pp. 13, 154).

Palunko, Ivana, Aleksandra Faust, Patricio Cruz, Lydia Tapia, and Rafael Fierro (2013). „A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots“. In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4896–4901 (cit. on p. 13).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). „On the difficulty of training recurrent neural networks“. In: *Proceedings of The 30th International Conference on Machine Learning* 2, pp. 1310–1318. arXiv: arXiv:1211.5063v2. URL: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf (cit. on p. 17).

Plagianakos, V. P., G. D. Magoulas, and M. N. Vrahatis (2001). „Learning Rate Adaptation in Stochastic Gradient Descent“. In: Springer US, pp. 433–444. URL: http://link.springer.com/10.1007/978-1-4613-0279-7_27 (cit. on p. 121).

Ploger, Paul G, Adriana Arghir, Tobias Giinther, et al. (2004). „Echo State Networks for Mobile Robot Modeling and Control“. In: *RoboCup 2003 Robot Soccer World Cup VII* Robocup 20, pp. 157–168. URL: http://www.springerlink.com/content/88yex45c3jcx81x6 (cit. on p. 19).

Poli, C (1973). „Dynamics of Slung Bodies Using a Single-Point Suspension System“. In: *Journal of Aircraft* 10.2, pp. 80–86. URL: http://arc.aiaa.org/doi/abs/10.2514/3.60200 (cit. on p. 12).

Postel, J. „User Datagram Protocol“. In: URL: https://tools.ietf.org/html/rfc768 (cit. on p. 65).

Prabhakar, A. (1977). „Stability of a helicopter carrying an underslung load“. In: (cit. on p. 12).

Prasad Sampath (1980). *Dynamics of a Helicopter-slung Load System*. URL: https://books.google.co.uk/books/about/Dynamics_of_a_Helicopter_slung_Load_Syst.html?id=-xJcnQEACAAJ&redir_esc=y (cit. on p. 12).

Puskorius, Gintaras V. and Lee A. Feldkamp (1994). „Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks“. In: *IEEE Transactions on Neural Networks* 5.2, pp. 279–297 (cit. on p. 18).

Quigley, Morgan, Ken Conley, Brian Gerkey, et al. (2009). „ROS: an open-source Robot Operating System“. In: *ICRA* 3, p. 5. arXiv: 1106.4561. URL: http://pub1.willowgarage.com/$\sim$konolige/cs225B/docs/quigley-icra2009-ros.pdf (cit. on p. 195).

Richards, Arthur, John Bellingham, and Tillerson (2002). „Coordination and control of multiple UAVs“. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 1–11. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Coordination+and+Control+of+Multiple+UAVs#1%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.1820&rep=rep1&type=pdf (cit. on p. 7).

Romero Ugalde, Hector M., Jean-Claude Carmona, Victor M. Alvarado, and Juan Reyes-Reyes (2013). „Neural network design and model reduction approach for black box nonlinear system identification with reduced number of parameters“. In: *Neurocomputing* 101, pp. 170–180. URL: http://linkinghub.elsevier.com/retrieve/pii/S0925231212006522 (cit. on p. 17).

Rossum, Guido Van, Python Software Foundation, Unladen Swallow, et al. (2011). „Python (programming language)“. In: *Flying*, pp. 1–14 (cit. on p. 62).

Rothman, Peter L. and Richard V. Denton (1991). „Fusion or confusion: knowledge or nonsense?“ In: ed. by Vibeke Libby. International Society for Optics and Photonics, pp. 2–12. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=963284 (cit. on pp. 9, 47).

Rumeihart, D E and J L Mcclelland (1968). „in Parallel Distributed Processing. Explorations in the Microstructure of Cognition“. In: *Clin. Neurophysiol.J. Physiol. J. Neurophysiol. J. Physiol. Shinoda, J. I. Yokota, T. Fukami, Neurosci. Lett.. RedmanJ. Comp. Neurol* 24.25, pp. 423–434. arXiv: arXiv:1011.1669v3. URL: http://www.jstor.org/stable/1702143%5Cnhttp://www.jstor.org/page/info/about/policies/terms.jsp%5Cnhttp://www.jstor.org (cit. on p. 109).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986a). „Learning representations by back-propagating errors“. In: *Nature* 323.6088, pp. 533–536. arXiv: arXiv:1011.1669v3 (cit. on p. 17).

Rumelhart, David E., James L. McClelland, and San Diego. PDP Research Group. University of California (1986b). *Parallel distributed processing : explorations in the microstructure of cognition*. MIT Press (cit. on p. 17).

Sadr, S., S. Ali A. Moosavian, and P. Zarafshan (2014). „Dynamics Modeling and Control of a Quadrotor with Swing Load“. In: *Journal of Robotics* 2014, pp. 1–12. URL: http://www.hindawi.com/journals/jr/2014/265897/ (cit. on p. 13).

Sai Dinesh, P, J Ananthapadmanabha, V Aravind, et al. (2010). „Low cost and real time electronic speed controller of position sensorless brushless DC motor“. In: *2010 Fifth International Conference on Information and Automation for Sustainability*. IEEE, pp. 329–334. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5715682 (cit. on pp. 8, 39).

Salameh, Z. M. and B. G. Kim (2009). „Advanced lithium polymer batteries“. In: *2009 IEEE Power and Energy Society General Meeting*. IEEE, pp. 1–5. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5275404 (cit. on p. 41).

Salamí, Esther, Cristina Barrado, and Enric Pastor (2014). „UAV Flight Experiments Applied to the Remote Sensing of Vegetated Areas“. In: *Remote Sensing* 6.11, pp. 11051–11081. URL: http://www.mdpi.com/2072-4292/6/11/11051/ (cit. on p. 1).

Schiavoni, R. Scattolini P. Bolzern and N. (2015). „Fondamenti di Controlli Automatici“. In: *McGraw-Hill Education* (cit. on pp. 10, 76).

Sethi, I.K. (1990). „Entropy nets: from decision trees to neural networks“. In: *Proceedings of the IEEE* 78.10, pp. 1605–1613. URL: http://ieeexplore.ieee.org/document/58346/ (cit. on p. 17).

Shamsudin, Syariful Syafiq, Xiaoqi Chen, Wenhui Wang, Christopher E. Hann, and Geoffrey Chase (2010). „Neural Networks based System Identification for an Unmanned Helicopter System“. In: *Proceedings of the 4th Asia International Symposium on Mechatronics*. Singapore: Research Publishing Services, pp. 12–19 (cit. on p. 19).

Shavlik, Jude W. and Thomas Glen. Dietterich (1990). *Readings in machine learning*. Morgan Kaufmann Publishers (cit. on p. 15).

Shim, David H., H. Jin Kim, and Shankar Sastry (2002). „A flight control system for aerial robots: Algorithms and experiments“. In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 15. 1, pp. 241–246 (cit. on p. 7).

Siegelmann, H T (1995). „Computation beyond the turing limit.“ In: *Science (New York, N.Y.)* 268.5210, pp. 545–8. URL: http://www.sciencemag.org/content/268/5210/545 (cit. on p. 114).

Simon, Dan (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley and Sons, p. 552. URL: https://books.google.com/books?hl=en&lr=&id=UiMVoP_7TZkC&pgis=1 (cit. on pp. 9, 48).

Singer, N., W. Singhose, and E. Kriikku (1997). *An input shaping controller enabling cranes to move without sway*. Tech. rep. Aiken, SC: Savannah River Site (SRS). URL: http://www.osti.gov/servlets/purl/491559-NFLqr9/webviewable/ (cit. on p. 148).

Smith, J. H., G. M. Allen, and D. Vensel (1973). „Design, Fabrication, and Flight Test of the Active Arm External Load Stabilization System for Cargo Handling Helicopters“. In: (cit. on p. 148).

Smith, Julius O. (Julius Orion) (2010). *Physical audio signal processing : for virtual musical instruments and audio effects*. W3K Publishing, p. 803 (cit. on p. 118).

Smith, Warren D. and Peter W. Shor (1992). „Steiner tree problems“. In: *Algorithmica* 7.1-6, pp. 329–332.

Sreenath, Koushil, Taeyoung Lee, and Vijay Kumar (2013). „Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load“. In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 2269–2274 (cit. on pp. 147, 150, 154).

Sreerama, K. Murthy (1998). „Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey“. In: *Data Mining and Knowledge Discovery* 2.4, pp. 345–389. URL: https://www.cs.nyu.edu/$\sim$roweis/csc2515-2006/readings/murthy_dt.pdf.

Starr, Gregory, John Wood, and Ron Lumia (2005). „Rapid Transport of Suspended Payloads“. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* April, pp. 394–399. URL: http://ieeexplore.ieee.org/document/1570310/ (cit. on p. 148).

Steil, Jochen J. (2004). „Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity“. In: *IEEE International Conference on Neural Networks - Conference Proceedings*. Vol. 2, pp. 843–848. arXiv: arXiv:1506.08836v1 (cit. on pp. 18, 123).

Stevens, L.B. and L.F. Lewis (2003). *Aircraft control and simulation*, pp. 103–106. URL: http://www.ulb.tu-darmstadt.de/tocs/114368600.pdf (cit. on pp. 10, 79).

Suzuki, Satoshi and KeiichiA Be (1985). „Topological structural analysis of digitized binary images by border following“. In: *Computer Vision, Graphics and Image Processing* 30.1, pp. 32–46 (cit. on p. 156).

Tabassum, Mujahid and Kuruvilla Mathew (2014). „Software evolution analysis of linux (Ubuntu) OS“. In: *2014 International Conference on Computational Science and Technology, ICCST 2014* (cit. on p. 61).

Takahashi, Osamu and R. J. Schilling (1989). „Motion Planning in a Plane Using Generalized Voronoi Diagrams“. In: *IEEE Transactions on Robotics and Automation* 5.2, pp. 143–150.

Tanaka, Tomohiro, Takahiro Shinozaki, Shinji Watanabe, and Takaaki Hori (2016). „Evolution Strategy Based Neural Network Optimization and LSTM Language Model for Robust Speech Recognition“. In: (cit. on p. 130).

Tang, Sarah and Vijay Kumar (2015). „Mixed Integer Quadratic Program Trajectory Generation for a Quadrotor with a Cable-Suspended Payload“. In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2216–2222 (cit. on pp. 13, 154).

Tarascon, J M and M Armand (2001). „Issues and challenges facing rechargeable lithium batteries.“ In: *Nature* 414.6861, pp. 359–67. URL: http://dx.doi.org/10.1038/35104644 (cit. on pp. 8, 41).

Thiels, Cornelius A, Johnathon M Aho, Scott P Zietlow, and Donald H Jenkins (2015). „Use of unmanned aerial vehicles for medical product transport.“ In: *Air medical journal* 34.2, pp. 104–8. URL: http://www.ncbi.nlm.nih.gov/pubmed/25733117 (cit. on p. 2).

Urnkranz, Johannes (1999). „Separate-and-Conquer Rule Learning“. In: *Artificial Intelligence Review* 13, pp. 3–54 (cit. on p. 17).

Valenti, Mario, Brett Bethke, Gaston Fiore, and Jonathan P How (2006). „Indoor Multi-Vehicle Flight Testbed for Fault Detection , Isolation , and Recovery“. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit* 8, pp. 1–18 (cit. on p. 7).

Vargas, Aldo, Murray Ireland, and David Anderson (2014). „Swing free manoeuvre controller for RUAS slung-load system using ESN“. In: *Proceedings of the 1st World Congress on Unmanned Systems Engineering* (cit. on pp. 21, 22, 68, 77, 118, 124, 125, 129, 133, 147, 190, 191, 203).

– (2015a). „Swing-Free Manoeuvre Controller for Rotorcraft Unmanned Aerial Vehicle Slung-Load System Using Echo State Networks“. In: *International Journal of Unmanned Systems Engineering* 3.1, pp. 26–37. URL: http://www.ijuseng.com/#/ijuseng-3-1-26-37-2015/4587568279 (cit. on pp. 22, 147, 191).

– (2015b). „System Identification of multi-rotor UAVs using echo state networks“. In: *AUVSI's Unmanned Systems* (cit. on pp. 21, 118, 130, 133, 191).

Vargas, Aldo, Murray Ireland, Kyle Brown, and David Anderson (2016). „The MAST Lab flight stack for GNC of micro UAV ' s“. In: (cit. on pp. 11, 21, 189).

Vargas, Aldo and David Anderson (2017). „Computer vision technique to estimate the slung load dynamics when coupled to a Multirotor Unmanned Aerial Vehicle“. In: *Revista Internacional de Investigacion e Innovación Tecnologica (RIIIT)* ISSN 2007-.Latindex Folio: 23614 (cit. on p. 147).

Verstraeten, D., B. Schrauwen, M. D'Haene, and D. Stroobandt (2007). „An experimental unification of reservoir computing methods“. In: *Neural Networks* 20.3, pp. 391–403 (cit. on pp. 18, 123).

Verstraeten, David, Benjamin Schrauwen, and Dirk Stroobandt (2006). „Reservoir-based techniques for speech recognition“. In: *Neural Networks*, pp. 1050–1053 (cit. on p. 124).

Wagstaff, Kiri (2012). „Machine Learning that Matters“. In: *Proceedings of the 29th International Conference on Machine Learning*, pp. 529–536. arXiv: 1206.4656. URL: http://arxiv.org/abs/1206.4656%5Cnhttp://icml.cc/discuss/2012/298.html (cit. on p. 15).

Werbos, Paul J. (1990). „Backpropagation Through Time: What It Does and How to Do It“. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560 (cit. on pp. 18, 120, 122).

Williams, Ronald J. and David Zipser (1989). „A Learning Algorithm for Continually Running Fully Recurrent Neural Networks“. In: *Neural Computation* 1.2, pp. 270–280. arXiv: arXiv:1011.1669v3 (cit. on pp. 17, 18, 122, 123).

Wong, K.C., J.A. Guerrero, D. Lara, and R. Lozano (2007). „Attitude stabilization in hover flight of a mini tail-sitter UAV with variable pitch propeller“. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2642–2647. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4399278 (cit. on p. 36).

Wysocki, Rafal and Wojciech Zabierowski (2011). „Twisted framework on game server example“. In: *2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, pp. 361–363.

Yamazaki, Tadashi and Shigeru Tanaka (2007). „The cerebellum as a liquid state machine“. In: *Neural Networks* 20.3, pp. 290–297 (cit. on pp. 18, 124).

Zameroski, Daniel, Gregory Starr, John Wood, and Ron Lumia (2008). „Rapid Swing-Free Transport of Nonlinear Payloads Using Dynamic Programming“. In: *Journal of Dynamic Systems, Measurement, and Control* 130.4, p. 041001. URL: http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1475743 (cit. on p. 148).

Websites

Abhijit, Singh (2016). *Unmanned and Autonomous Vehicles and Future Maritime Operations in Littoral Asia | ORF*. URL: http://www.orfonline.org/research/unmanned-and-autonomous-vehicles-and-future-maritime-operations-in-littoral-asia/ (visited on Feb. 22, 2017) (cit. on p. 1).

Alec Momont (2014). *TU Delft - Ambulance Drone*. URL: https://www.youtube.com/watch?v=y-rEI4bezWc (visited on Dec. 17, 2016) (cit. on p. 14).

Amazon (2013). *Amazon Prime Air*. URL: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011 (visited on Dec. 17, 2016) (cit. on pp. 3, 14).

Andreas Raptopoulos (2013). *Andreas Raptopoulos: No roads? There's a drone for that | TED Talk | TED.com*. URL: http://www.ted.com/talks/andreas_raptopoulos_no_roads_there_s_a_drone_for_that (visited on Dec. 17, 2016) (cit. on p. 14).

Brumfield, Eric (2014). *Armed Drones for Law Enforcement: Why it Might Be Time to Re-Examine the Current Use of Force Standard* (cit. on p. 1).

Burns, Stuart (2014). *Rotite Fastener*. URL: http://www.rotite.com/ (cit. on pp. 20, 29, 188).

DHL (2013). *Deutsche Post DHL Group | DHL Parcelcopter 3.0*. URL: http://www.dpdhl.com/en/media_relations/specials/parcelcopter.html (visited on Dec. 17, 2016) (cit. on p. 14).

DIYDrones, 3DRobotics. *Arducopter*. URL: http://copter.ardupilot.com/ (cit. on pp. 9, 46).

Ehang Inc (2016). *EHANG 184 autonomous aerial vehicle*. URL: http://www.ehang.com/ehang184 (visited on Feb. 20, 2017) (cit. on p. 2).

French, Robert M. (1999). *Catastrophic forgetting in connectionist networks* (cit. on p. 124).

GmbH, Ascending Technologies. *Ascending Technologies*. URL: http://www.asctec.de/ (cit. on p. 68).

Google (2014). *Google Project Wing X*. URL: https://x.company/projects/wing/ (visited on Dec. 17, 2016) (cit. on pp. 3, 14).

Hardkernel Co., Ltd. (2014). *Odroid U3* (cit. on p. 60).

Harrington, Aaron Michael (2011). *Optimal Propulsion System Design for a Micro Quad Rotor*. URL: http://drum.lib.umd.edu/handle/1903/12029 (cit. on p. 36).

HiSystems (2006). *MikroKopter*. URL: http://www.mikrokopter.de/en/home (cit. on pp. 9, 46).

IEEE (2013). *IEEE 802.3 Standard for Ethernet' Marks 30 Years of Innovation and Global Market Growth*. URL: http://vlib.excelsior.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bwh&AN=bizwire.c49319159&site=eds-live&scope=site (cit. on p. 65).

IMechE (2014). *The Unmanned Aircraft Systems Challenge*. URL: https://www.imeche.org/get-involved/young-members-network/auasc (visited on Jan. 5, 2017) (cit. on p. 196).

Kendoul, Farid (2012). *Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems*. arXiv: 10.1.1.91.5767 (cit. on p. 7).

Korsak, Kazimierz, Kenneth R. Meenen, Donald N. Meyers, and Frank N. Piasecki (1972). *Multi-Helicopter Heavy Lift System Feasibility Study* (cit. on p. 11).

Lefkowitz, Glyph (2002). *Twisted*. URL: http://twistedmatrix.com/ (cit. on p. 65).

Liyan Yi (2014). *Swinging Crane Project*. URL: https://www.eleceng.adelaide.edu.au/students/wiki/projects/index.php/Projects:2014s2-80_Swinging_Crane_Project (visited on Jan. 4, 2017).

Meier, Lorenz (2009). *MAVlink*. URL: https://github.com/mavlink/mavlink/commit/a087528b8146ddad17e9f39c1dd0c1353e5991d5 (cit. on p. 66).

Müller, Markus (2004). *eCalc*. URL: http://www.ecalc.ch/.

MultiWii. *MultiWii*. URL: http://www.multiwii.com/ (cit. on p. 203).

– (2010). *MultiWii*. URL: http://www.multiwii.com/ (cit. on pp. 9, 10, 46, 77).

NN3 (2007). *Forecasting Competition for Neural Networks and Computational Intelligence*. URL: http://www.neural-forecasting-competition.com/NN3/index.htm (cit. on p. 124).

OpenPilot (2011). *OpenPilot*. URL: http://www.openpilot.org/ (cit. on pp. 9, 46).

Paparazzi (2003). *Paparazzi Project*. URL: http://paparazziuav.org (cit. on pp. 9, 46).

ROSENBLATT, FRANK (1961). *PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS* (cit. on p. 17).

ST (2009). *Three-phase BLDC motor control software library*. URL: http://www.st.com/web/en/resource/technical/document/user_manual/CD00236524.pdf (cit. on p. 40).

Starlino (2011). *DCM Tutorial*. URL: http://www.starlino.com/dcm_tutorial.html (cit. on p. 51).

Tridgell, Andrew (2013). *MAVProxy*. URL: http://tridge.github.io/MAVProxy/ (cit. on p. 66).

Vargas, Aldo (2013a). *3D printed Quadrotor using Rotite elements*. URL: http://altax.net/hangar/tego-v3/ (cit. on p. 29).

– (2013b). *pyMultiwii - python module that handles the MultiWii Serial Protocol*. URL: https://github.com/alduxvm/pyMultiWii (cit. on pp. 66, 190).

– (2014). *DronePilot - Open Source project to control flight controllers*. URL: https://github.com/alduxvm/DronePilot (cit. on p. 55).

Voos, H (2009). *Nonlinear control of a quadrotor micro-UA[1] H. Voos, Nonlinear control of a quadrotor micro-UAV using feedback-linearization, no. April. Ieee, 2009, pp. 16.V using feedback-linearization*. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4957154 (cit. on p. 78).

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style, modified by the author of this document.