# Enhancing 3D Autonomous Navigation Through Obstacle Fields 

## Homogeneous Localisation and Mapping, with

 Obstacle-Aware Trajectory Optimisation
## By

Benjamin J. Morrell

A thesis submitted in fulfilment of the requirements of the degree of DOCTOR OF PHILOSOPHY.

Faculty of Engineering and IT<br>School of Aerospace, Mechanical \& Mechatronic Engineering<br>The University of Sydney

For my parents and grandparents

## AUTHOR'S DECLARATION

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.
I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

SIGNED: DATE:

Benjamin J. Morrell

## Acknowledgements

This thesis was completed thanks to the support of the Australian Government, through the Australian Postgraduate Award, and the University of Sydney who granted the award, in addition to the Vice Chancellors Research Scholarship and a Faculty of Engineering top-up scholarship. Pursuing a PhD thesis can become all-consuming, and having financial matters all comfortably accounted for is of great help to enable a focus on the research. I am constantly aware of how fortunate we are in Australia to have such strong support.

Thanks also go to the University of Sydney and their staff, for their support throughout the pursuit of the PhD. Part of the research presented in this thesis was carried out during a year at Texas A\&M University, and thanks go to the staff and students there for being incredibly welcoming and adopting an Aussie Aggie. The time at Texas A\&M was supported by the American Australian Association, Northrop Grumman, Australian to US Fellowship. I am humbled by their support and am grateful for the opportunity they enabled.

Part of this work was also carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by The University of Sydney and the National Aeronautics and Space Administration. I thank JPL for creating opportunities to do research there and to learn from those at the organisation.

I have had some genuinely remarkable opportunities to be involved in exciting, engaging and unique projects throughout my PhD, and this is thanks to my supervisor, Prof. Gregory Chamitoff. My life has truly been enriched with being able to spend time with you, learn from you, and work with you in robotic space engineering and beyond. The pathways you have opened for me are staggering, and it is often hard to believe what I have been able to be a part of, with your guidance.

Thanks also to Prof. Peter Gibbens for providing a great amount of insight, advice and guidance throughout my research at the University of Sydney.

To Robert Reid, I owe great thanks, for providing the unique opportunity to pursue research at JPL, and for your guidance, mentorship and support throughout my time there. Thanks also to Prof. Stefan Williams for helping to open the potential to do research at JPL.

To Prof. KC Wong, when I walked into your office at the end of my undergraduate degree, wondering what I should do next, the opportunity you offered to work with Greg has opened up a world of possibility of which I could not have dreamt. For this, and more, you have my enduring gratitude.

Much of the work that is presented in this thesis was the product of collaborative projects. I thoroughly enjoyed these collaborations both from the academic preservative and for the camaraderie. To Derek Keuther, I gained a lot from our time working closely together at Texas A\&M. To Mauricio Coen, I very much enjoyed working with you over an extensive range of projects and am extremely happy that we were able to work together over the last six months of my PhD . It is empowering to see how much we can achieve, and I look forward to working together in the future. To Marc Rigter, I feel fortunate to have been able to work with you both in Sydney and at JPL. Our discussions and collaborations have significantly helped to advance my research. Thanks for your support through the late nights of flights testing and long days of debugging. To Anne Bettens, it has been invaluable to have a great team to work with through the long coding sessions to push towards creating something valuable. Thanks for your support and understanding through the final rush.

Throughout my time at The University of Sydney, I have been fortunate to have the support of fellow students, to push through the challenges of a PhD. To Andrew Gong, it has been a joy to be on the journey with you right from the first year of undergrad. To Asiful 'Ovi' Islam, it was a pleasure to share an office with you and valuable to learn from your impressive approach to research and life. To Anastasiia Volkova, I feel privileged to have been able to share so much time together (whenever we are in the same location). You are an inspiration to strive for more and push through the challenges of pursuing great things. Thanks for your steadfast support in the final stages of the PhD. To Abhijeet Kumar, as partners in crime for outreach work, it is incredible to see the impact we have been able to have. It has also been invaluable to have your support through these endeavours, and striving for the balance of research and outreach. There has also been a healthy community of graduate students that I would like to thank: Matt Anderson, Steven Piper, David Williams, Darren Lamburn, Daniel Linton and Zihao Wang. A particular thanks to Matt Anderson for his valuable review of this thesis, and regular moral support. Thanks to numerous other friends and mentors who have been through a PhD and have provided guidance on navigating the journey.

I am extremely grateful to my friends and family for their patience, understanding and support, especially through times when I am travelling around the world, or when I fall off the radar in the depths of research work. A particular thanks to my uncle Richard Harding for providing his house as a writing vacation destination. To my uncle Tony Harding, I am forever grateful for your incredible support, particularly through the final stages of my PhD, by providing a place for me to live, being flexible to my work hours, and reviewing my thesis. Thanks to Randii and Alice Wesson for creating a Northern Hemisphere home during my time at JPL, and providing the support to allow me to focus on my research there.

I reserve the final thanks for my parents. You are the foundation from which I can pursue this PhD , the launch complex from which I can reach for the stars. You are always there with unfailing support, and knowing this is incredibly powerful. It gives me the confidence to pursue exciting and engaging opportunities. Your understanding and assistance throughout the PhD have been dearly valued. I hope to make the most of what you have given me to truly make a positive impact on the world.

## AUTHORSHIP ATTRIBUTION STATEMENT

This thesis contains material published in [35, 109, 153-156, 192]. The location of this material and my contribution in the publications is elaborated below and explained throughout the thesis.
[155]Morrell, B. J.; Chamitoff, G. \& Gibbens, P.
Autonomous Operation of Multiple Free-Flying Robots on the International Space Station 25th AAS/AIAA Spaceflight Mechanics Conference, 2015.

## Contribution:

I conducted the work and wrote the paper.

## Location of material in thesis:

Sections 2.3.3, 4.3.5, 4.4.3 and 4.4.7
[109]Kuether, D. J.; Morrell, B. J.; Chamitoff, G. E.; Bishop, M.; Mortari, D.; Gibbens, P. W. \& Coen, M. Cohesive Autonomous Navigation System
AIAA Guidance Navigation and Control Conference, AIAA SciTech, 2016.

## Contribution:

An equal collaboration with the primary author. I worked on the 3D modelling and SLAM components.

## Location of material in thesis:

Section 3.1.1
[156]Morrell, B. J.; Chamitoff, G. E.; Kuether, D. J.; Coen, M. \& Gibbens, P.
Integration of 3D SLAM, Rigid Body Landmarks and 3D Path Planning
AIAA SPACE 2016, 2016, 5411.

## Contribution:

Extension from work in collaboration with D. Kuether. I worked on the 3D modelling and SLAM components.

## Location of material in thesis:

Sections 2.2.2, 3.1.1 and 3.1.2
[35]Chamitoff, G. E.; Saenz-Otero, A.; Katz, J. G.; Ulrich, S.; Morrell, B. J. \& Gibbens, P. W.
Real-time maneuver optimization of space-based robots in a dynamic environment: Theory and on-orbit experiments
Acta Astronautica, Elsevier, 2018, 142, 170-183.

## Contribution:

I worked on analysis of on-orbit test-data, simulated test-cases and the literature review. My work in this publication provides the basis for the trajectory optimisation contributions in this thesis.

## Location of material in thesis:

Sections 2.3.2, 4.3, 4.4.1 and 4.5
[153]Morrell, B.; Rigter, M.; Merewether, G.; Reid, R.; Thakker, R.; Tzanetos, T.; Rajur, V. \& Chamitoff, G.

Differential Flatness Transformations for Aggressive Quadrotor Flight
Robotics and Automation (ICRA), 2018 IEEE International Conference on Robotics and Automation, 2018.

## Contribution:

I developed the theory, conducted tests, analysed results and wrote the paper.

## Location of material in thesis:

Sections 5.1 and 6.2
[154]Morrell, B.; Thakker, R.; Merewether, G.; Reid, R.; Rigter, M.; Tzanetos, T. \& Chamitoff, G.
Comparison of Trajectory Optimization Algorithms for High-Speed Quadrotor Flight in Close Proximity to Obstacles
Robotics and Automation Letters, 2018. Currently in submission process.

## Contribution:

I developed the theory, conducted tests, analysed results and wrote the paper.
Location of material in thesis:
Sections 2.3.2, 2.3.4, 5.3 and 6.3
[192]Reid, R.; Merewether, G.; Tzanetos, T.; Morrell, B.; Rigter, M. \& Matthies, L.
A High-Speed Autonomous Quadrotor System for Vision-based Teach \& Repeat
Journal of Field Robotics, 2018. Currently in submission process.

## Contribution:

I contributed to development of controllers, trajectory planners and the ground control station, as well performing test and analysis. I wrote the parts of the paper that appear in this thesis.

## Location of material in thesis:

Sections 2.3.4 and 6.1

All work from these publications that is contained in this thesis is my own work unless otherwise stated and cited. In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

## SIGNED:

DATE:
Benjamin J. Morrell

As the supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

SIGNED:
DATE:
Gregory E. Chamitoff

Abstract

The capability of small flying robots, such as quadrotors and free-floating satellites, are making them useful tools for a wide range of applications. Quadrotors can be used for search and rescue, facility inspection, infrastructure surveying and parcel delivery. Free-flying satellites can be assistants inside space stations and monitor the outside of space stations and satellites. A critical capability to enable these applications is autonomous navigation near obstacles. Autonomous navigation is a challenge for small flying robots as they have limited payload capacity, hence require low-powered, low-weight sensors, and efficient computation. The robots also need to localise, map, and plan trajectories in 3D, a significantly enhanced challenge over 2D applications. Being flying vehicles, the dynamic-feasibility of planned trajectories and the control algorithms to track the trajectories are also essential considerations.

The state-of-the-art for autonomous navigation systems is heterogeneous, with a combination of many different algorithms. What is proposed here is a more homogeneous system, with the aim for enhanced efficiency.

To determine the location of a robot, visual Simultaneous Localisation and Mapping (SLAM) algorithms using stereo or depth cameras are the leading approach for small flying robots. While SLAM algorithms produce a map, it is purely for localisation, so a separate 3D mapping algorithm is required: producing occupancy grids or signed distance fields for trajectory planning. No algorithms can combine SLAM and 3D mapping into one algorithm without the use of lidar. This work proposes the use of 3D objects, modelled with Non-Uniform Rational B-Spline (NURBS) surfaces, to serve both as features for SLAM and as obstacles for trajectory planning. Modelling as objects, rather than complete environments, manages the computational requirements, and using NURBS surfaces allows the resolution to be varied for different tasks. The proposed approach is demonstrated on sets of simulated data, demonstrating tracking errors of under $2 \%$ of the total path length, mapping errors as low as 2 cm and an appropriate collision-cost profile for obstacle representation in trajectory planning.

Leading trajectory planning approaches are also heterogeneous, with the combination of a global path planner, local trajectory optimiser and reactive obstacle avoidance. This split of algorithms can provide sub-optimal trajectories, though, when being used for flight close to obstacles. Presented here is the Admissible Subspace Trajectory Optimiser (ASTRO), an algorithm that provides a middle ground, optimising dynamics over a large horizon with consideration for complex obstacle fields. ASTRO performs polynomial optimisation with the inclusion of constraints. The constraint formulation is flexible to include a wide range of obstacles, including dynamic obstacles with motion models and uncertainty growth. ASTRO is shown to provide comparable computation time and success rate to the state-of-the-art, through batches of simulations. Flight tests on quadrotors show that the algorithm can produce trajectories that are more dynamically-feasible (easier to track) than the state-of-the-art, by including obstacles directly in the optimisation.

Trajectory tracking control for quadrotors utilises the differential flatness transformation to link position and attitude controllers. There are singularities in the transformation though, and existing methods to handle the singularities can fail in different scenarios. These methods are analysed in detail to highlight where failures occur, and a new, robust method is proposed. The new method is successfully demonstrated in aggressive flights.

The proposed algorithms for SLAM and trajectory planning are brought together into a complete system to demonstrate the homogeneous concept. This system is compared to the state-of-the-art in a novel simulation framework. The results successfully prove the concept that a single 3D representation can be used for localisation, mapping and planning with lightweight sensors. The current implementation of NURBSLAM is shown to be less efficient and less accurate than the state-of-the-art; however, it is more robust in scenarios with sparse visual features, successfully operating in cases where other visual SLAM algorithms fail, and demonstrating better recovery from errors.

The work presented in this thesis can be built upon to evolve the SLAM algorithm further, to be more efficient and accurate. Tests can be performed in more environments and with real camera data, to aid development and to further characterise where NURBSLAM provides benefits over the state-of-the-art.

## TABLE OF CONTENTS

Page
List of Tables ..... xix
List of Figures ..... xxi
1 Introduction ..... 1
1.1 Background ..... 2
1.2 Challenges and Current State-of-the-Art ..... 4
1.3 Gaps ..... 6
1.4 Focus of Thesis ..... 8
1.5 Contributions ..... 8
1.5.1 Localisation and Mapping ..... 9
1.5.2 Trajectory Optimisation ..... 10
1.5.3 Analysis of the Differential Flatness Transformation for Quadrotors ..... 11
1.5.4 Analysis of Dynamic Feasibility of Trajectories for Quadrotors and the Impact of How Obstacles are Considered ..... 11
1.6 Outline of Thesis ..... 14
2 Background and Related Work ..... 15
2.1 Simultaneous Localisation And Mapping (SLAM) ..... 16
2.1.1 Key Concepts and Terminology ..... 16
2.1.2 Point Cloud SLAM ..... 20
2.1.3 Visual SLAM - Preliminaries ..... 21
2.1.4 Indirect Visual SLAM ..... 24
2.1.5 Direct Visual SLAM ..... 26
2.1.6 Semi-Direct Visual SLAM ..... 28
2.1.7 Appearance-Based Visual SLAM ..... 29
2.1.8 Geometric-Feature-Based Visual SLAM ..... 30
2.1.9 Summary and Assessment ..... 30
2.2 3D Mapping ..... 35
2.2.1 3D Mapping Algorithms ..... 35
2.2.2 3D Modelling ..... 43
2.3 Trajectory Optimisation ..... 49
2.3.1 Optimisation Approaches ..... 49
2.3.2 Planning with Obstacles ..... 50
2.3.3 Planning with Dynamic Obstacles ..... 53
2.3.4 Trajectory Planning for Quadrotor UAVs ..... 57
2.4 Complete Systems ..... 62
2.4.1 Software ..... 62
2.4.2 System Examples ..... 62
2.4.3 Current State-of-The-Art ..... 64
2.5 Summary and Identification of Gaps ..... 64
3 Localisation and Mapping with 3D Object Representations ..... 67
3.1 Review of Candidate 3D Object Representations ..... 68
3.1.1 Ellipsoids - Full Application to SLAM and Trajectory Planning ..... 68
3.1.2 Gaussian Process Implicit Surfaces - Assessment of Potential for SLAM and Trajectory Planning ..... 76
3.1.3 Non-Uniform Rational B-Splines - Assessment of Potential for SLAM and Trajec- tory Planning ..... 79
3.1.4 Selected 3D Object Representation ..... 86
3.2 NURBSLAM: Using NURBS Surfaces for Localisation, Mapping and Trajectory Planning ..... 87
3.2.1 Data Association ..... 87
3.2.2 Mapping - Object Generation ..... 87
3.2.3 Mapping - Object Update ..... 90
3.2.4 Localisation ..... 97
3.2.5 SLAM ..... 103
3.2.6 Trajectory Optimisation ..... 104
3.3 NURBSLAM Demonstration, Testing and Analysis ..... 106
3.3.1 Mapping ..... 106
3.3.2 Localisation ..... 110
3.3.3 SLAM ..... 112
3.3.4 Trajectory Optimisation ..... 118
3.4 Conclusion ..... 119
4 Trajectory Optimisation ..... 121
4.1 Contributions ..... 122
4.2 Preliminaries ..... 124
4.3 Algorithm Description ..... 124
4.3.1 Convexity of the Cost Function ..... 127
4.3.2 Boundary Conditions ..... 128
4.3.3 Obstacles and Performance Constraints ..... 131
4.3.4 Example Constraint Cost Functions ..... 134
4.3.5 Dynamic Obstacles ..... 140
4.3.6 Optimisation Techniques ..... 142
4.3.7 Replanning and Multiple Robots ..... 146
4.3.8 Multi-Segment Optimisation ..... 148
4.3.9 Summary of ASTRO ..... 151
4.4 Simulation Results ..... 152
4.4.1 Static Demonstrations ..... 152
4.4.2 Randomised Seeding and Perturbations ..... 157
4.4.3 Dynamic Obstacles and Multi-Robot Planning ..... 158
4.4.4 Analysis of Optimisation Techniques ..... 164
4.4.5 Constraint Type Comparison ..... 167
4.4.6 Convex, Quadratic Steps and Line Search ..... 169
4.4.7 Computation Time Analysis ..... 173
4.4.8 Summary of Simulation Tests ..... 174
4.5 Trajectory Optimisation for Space-Based Robotics ..... 175
4.5.1 SPHERES ..... 175
4.5.2 On-Orbit Testing ..... 176
4.5.3 Results From On-Orbit Testing ..... 177
4.5.4 Lessons Learned ..... 180
4.6 Conclusion ..... 181
5 Trajectory Optimisation for Quadrotor UAVs ..... 183
5.1 The Differential Flatness Transformation for Quadrotors ..... 184
5.1.1 Description of the Transformation ..... 185
5.1.2 Singularities ..... 188
5.1.3 Existing Methods to Address the Singularity ..... 188
5.1.4 Analysis of Differential Flatness Transformations ..... 190
5.1.5 New Approaches to Address the Singularity ..... 192
5.1.6 Summary of Analysis ..... 196
5.2 ASTRO for Quadrotors ..... 197
5.2.1 Modifications to ASTRO for Application to Quadrotors ..... 198
5.2.2 Comparison with Existing Planners ..... 198
5.3 Quadrotor Trajectory Optimisation - Simulation Comparisons ..... 202
5.3.1 Algorithm Implementation ..... 202
5.3.2 Test Case Generation ..... 203
5.3.3 Results ..... 203
5.3.4 Summary and Assessment of Simulation Comparisons ..... 205
5.4 Conclusion ..... 206
6 UAV Flight Demonstrations ..... 207
6.1 Description of Hardware System ..... 208
6.1.1 High-Level Architecture ..... 208
6.1.2 Airframe ..... 210
6.1.3 On-Board Computing ..... 210
6.1.4 Actuation ..... 211
6.1.5 Control ..... 211
6.1.6 Localisation ..... 212
6.1.7 Mapping ..... 213
6.1.8 Planning ..... 214
6.1.9 Ground Control Station ..... 215
6.2 Differential Flatness Testing - Aggressive Flights ..... 217
6.2.1 Software-in-the-Loop Tests ..... 218
6.2.2 Flight Tests ..... 218
6.2.3 Conclusions - Differential Flatness ..... 221
6.3 Comparison of Planners ..... 222
6.3.1 Obstacle-Aware Flight Tests ..... 223
6.3.2 Conclusions - Comparison of Planners ..... 225
6.4 Conclusions ..... 227
7 Integrated System ..... 229
7.1 SpaceCRAFT Robot Simulation Framework ..... 230
7.1.1 Framework Design ..... 231
7.2 SLAM Demonstration ..... 233
7.2.1 Test Case ..... 233
7.2.2 Results ..... 233
7.2.3 Comments ..... 236
7.3 Full System Demonstration ..... 236
7.3.1 Test Case ..... 236
7.3.2 Results - NURBSLAM ..... 237
7.3.3 Results - Performance Comparison ..... 241
7.4 Conclusions and Discussion ..... 242
8 Conclusion ..... 245
9 Future Work ..... 249
A SVD for Determining Transformations ..... 251
B Subspace Projection ..... 253
B. 1 Gradient ..... 253
B. 2 Coefficients ..... 254
C Quaternion Maths ..... 255
C. 1 Quaternion Definition ..... 255
C. 2 Quaternion Multiplication ..... 257
C. 3 Quaternion Multiplication properties ..... 258
C. 4 Transformations vs. Rotations ..... 259
C. 5 Quaternions for Attitude Transformations ..... 259
C.5.1 Quaternions to Rotation Matrices ..... 260
C. 6 Quaterion Rates ..... 261
C. 7 Quaternion Logarithm and Exponential ..... 261
C. 8 Quaternion Interpolation - SLERP ..... 262
C. 9 Quaternion Finite Differencing ..... 263
C. 10 Quaternion Integration ..... 264
D Quaternion Derivation of Differential Flatness Transform ..... 265
D. 1 Compute the Thrust Vector ..... 265
D. 2 Coupling Thrust with Attitude ..... 265
D. 3 Singularities ..... 267
D. 4 Angular Rates ..... 267
D. 5 Angular Acceleration ..... 268
List of Acronyms and Abbreviations ..... 270
List of Symbols ..... 274
Bibliography ..... 279

## List of TABLES

TABLE Page
1.1 Summary of gaps and contributions ..... 13
2.1 Comparison of key factors for indirect and direct visual SLAM ..... 29
2.2 Datasets for SLAM testing ..... 32
2.3 Summary of SLAM algorithms ..... 33
2.4 Demonstrations and tests of SLAM algorithms ..... 34
2.5 Assessment of algorithms for 3D modelling ..... 48
3.1 Tracking and mapping errors for simulated Ellipsoid SLAM examples ..... 71
3.2 Summary of strengths and limitations of 3D modelling algorithms ..... 86
3.3 RSME errors for mapping tests ..... 106
3.4 Tracking and mapping errors for SLAM test case ..... 114
3.5 Scaled sensitivity to NURBSLAM parameters ..... 115
3.6 Percentage of computation time for a single scan for steps of the NURBSLAM process ..... 117
4.1 Comparison of optimisation configurations: ..... 165
4.2 Trajectory planning results for a single trajectory in a large warehouse environment ..... 168
4.3 Comparison of constraint types for batch of 100 test cases in a large warehouse environment 169
4.4 Trajectory planning computation times and improvement ..... 174
4.5 Trajectory planning computation times ..... 174
5.1 Performance summary of differential flatness methods ..... 197
5.2 Results from Simulation Batch Test ..... 205
6.1 Tracking errors for flights at 35s ..... 224
7.1 Tracking errors for orbit test case ..... 235
7.2 Tracking errors for full system demonstration ..... 241

## List of Figures

Figure Page
1.1 The autonomous navigation stack ..... 2
1.2 The different layers of planning algorithms. ..... 3
2.1 SLAM depiction ..... 17
2.2 Demonstration of loop closure with ORB-SLAM2 ..... 20
2.3 Depiction of the different variations of direct SLAM methods ..... 22
2.4 Examples of image features being extracted ..... 24
2.5 Example maps produced by indirect and semi-direct SLAM algorithms ..... 26
2.6 Example maps produced by semi-dense and sparse visual SLAM algorithms ..... 27
2.7 Example map from RatSLAM ..... 30
2.8 Example maps from geometric-feature-based SLAM ..... 31
2.9 Dense 3D reconstruction from ORB-SLAM2 ..... 36
2.10 Example points cloud dense 3D reconstructions ..... 37
2.11 Example combination of global and local occupancy maps ..... 38
2.12 Example map representations ..... 40
2.13 Rapid replanning examples ..... 54
2.14 Demonstrated applications of quadrotor trajectory planning ..... 58
2.15 Examples of aggressive manoeuvres for quadrotors ..... 60
2.16 Hierarchical controller diagram ..... 61
2.17 Example quadrotor systems ..... 63
3.1 Ellipsiod model described by centroid ..... 69
3.2 Example ellipsoid extraction ..... 69
3.3 Ellipsoid SLAM simulated examples. ..... 71
3.4 Large simulated Ellipsoid-SLAM test case ..... 72
3.5 Top-down view of the Large simulated Ellipsoid-SLAM test case ..... 72
3.6 Image processing pipline ..... 73
3.7 Ellipsoid-SLAM trajectory and map with real data ..... 74
3.8 Tracking errors from the Ellipsoid-SLAM test on real data ..... 74
3.9 Illustration of issues with Ellipsoid-SLAM ..... 75
3.10 GPIS surface generation for a sphere ..... 77
3.11 Parametric mesh required for NURBS ..... 79
3.12 NURBS curve fitting examples ..... 83
3.13 Mesh requirements for NURBS surface fitting ..... 84
3.14 NURBS surface fitting example ..... 85
3.15 Mesh generation example ..... 90
3.16 Steps to use new data to extend a NURBS curve ..... 91
3.17 Depiction of surface splitting steps ..... 96
3.18 Example Surface extension. ..... 97
3.19 Demonstration of NURBS alignment ..... 99
3.20 Flow diagram for the SLAM algorithm ..... 104
3.21 Mapping sequence for the distorted spheroid object ..... 107
3.22 Final mapping result from a single orbit of the distorted spheroid object ..... 108
3.23 Mapping results for different objects ..... 109
3.24 Mapping result for the distorted spheroid object when using multiple NURBS surfaces ..... 109
3.25 3D track of localisation testing ..... 110
3.26 Error plots for localisation test ..... 111
3.27 Localisation odometry error ..... 111
3.28 SLAM Tracking and Mapping ..... 112
3.29 Error plots for NURBSLAM tracking ..... 113
3.30 SLAM odometry error analysis ..... 113
3.31 Mapped object from SLAM test case ..... 114
3.32 Accuracy/computation time trade-off analysis ..... 116
3.33 Trajectory planning with NURBS objects ..... 118
4.1 ASTRO algorithm depiction ..... 124
4.2 Ellipsoid depiction ..... 135
4.3 Cylinder constraint diagram ..... 136
4.4 Rectangular prism constraint diagram ..... 138
4.5 Example of a slice of an ESDF ..... 139
4.6 Diagram of dynamic obstacle model ..... 141
4.7 Timeline of delays and fixed replanning computation time ..... 147
4.8 ASTRO planning sequence for static obstacles ..... 153
4.9 ASTRO planning a single segment with cubic prism obstacle constraints ..... 154
4.10 Multi-segment trajectory optimisation with ASTRO using an ESDF ..... 155
4.11 Example of trajectory optimisation with keep-in corridor constraints ..... 156
4.12 Demonstration of randomised initial seeding ..... 157
4.13 Effect of random perturbations to escape from infeasible local minima ..... 158
4.14 Dynamic obstacle test case snapshots ..... 159
4.15 Adversarial dynamic obstacle with keep-in corridor constraints ..... 159
4.16 Two robot dynamic replanning example ..... 160
4.17 Cooperative planning impasse example ..... 161
4.18 Six robot trajectory planning example, without cooperation ..... 162
4.19 Animation sequence for six robots navigating through a junction with cooperation ..... 163
4.20 Comparison of different methods of trajectory optimisation through obstacles ..... 168
4.21 Cost step results at different stages of optimisation for the first iteration ..... 170
4.22 Cost step results at different stages of optimisation for the third iteration ..... 171
4.23 Cost step results showing benefit of quadratic line-search ..... 172
4.24 Convex trajectory optimisation between seven waypoints with cylindrical keep-in constraints17 ..... 173
4.25 The Synchonized Position Hold, Engage, Reorient Experimental Satellites (SPHERES) ..... 176
4.26 Implementation limitations ..... 177
4.27 SPHERES planned and true trajectories with a dynamic obstacle ..... 178
4.28 SPHERES planned and true trajectories with dynamic and static obstacles ..... 179
5.1 Block diagram of the hierarchical tracking controller ..... 185
5.2 Quadrotor axes convention ..... 185
5.3 Axes evolution for pitching through the singularity ..... 191
5.4 Axes evolution for picthing forward near the singularity ..... 192
5.5 Two examples where the differential flatness transformations fail ..... 193
5.6 Axes evolution for pitching while at $90^{\circ}$ roll ..... 195
5.7 Closest axes selection and error between each axes-set and the current orientation ..... 196
5.8 Environments used for trajectory planning ..... 204
6.1 Concept of operations ..... 209
6.2 High-Level Architecture ..... 209
6.3 Quadrotor used for flight tests ..... 210
6.4 Example of RDP reducing the number of waypoints ..... 215
6.5 Ground Control Station with operator controls ..... 216
6.6 Planned aggressive trajectory between three waypoints ..... 217
6.7 Software in the loop simulation results for pitching trajectory ..... 218
6.8 Aggressive trajectory flight results for three differential flatness transformations ..... 219
6.9 Planned aggressive trajectory acceleration and corresponding attitude set points as quaternions220
6.10 Flight results from highly aggressive trajectory ..... 221
6.11 Medium lab environment with numerous obstacles ..... 222
6.12 Planned and executed trajectories for each algorithm at a trajectory time of 35 s ..... 224
6.13 Tracking Root-Mean-Square (RMS) errors across a range of flights of increasing speed for each algorithm ..... 225
6.14 Planned and executed trajectories for UNCO and ASTRO at a trajectory time of 25 s ..... 226
7.1 System diagram for the ROS-Unreal robot simulation framework ..... 231
7.2 Example graphics generated in an Unreal simulation ..... 232
7.3 Orbit test case trajectories, comparing NURBSLAM and ORB-SLAM2 ..... 234
7.4 Position errors for orbit test case ..... 234
7.5 Angular errors for orbit test case ..... 235
7.6 Odometry-error for orbit test case ..... 235
7.7 Mapping examples from NURBSLAM in the orbit test case ..... 236
7.8 Planned, tracked and true trajectories from NURBSLAM in the full system demonstration ..... 237
7.9 Trajectory from NURBSLAM in the full system demonstration with top-down view ..... 238
7.10 Tracking errors for NURBSLAM in the full system demonstration ..... 238
7.11 Odometry errors for NURBSLAM in the full system demonstration ..... 239
7.12 Mapping examples from NURBSLAM in the full system demonstration ..... 240
7.13 Full system results for ORB-SLAM2 with Voxblox and ASTRO ..... 241


## INTRODUCTION

Improvements in light, powerful processors and compact sensors have led to increasing capabilities for small flying robots, which in turn has led to a growing number of areas to apply such robots. Unmanned Aerial Vehicles (UAVs), also known as drones, are one type of flying robot that has rapidly become more prevalent. In particular, quadrotor UAVs that use multiple propellers to provide lift and control, have grown in popularity due to their ability to take-off and land vertically, to hover, fly at slow speeds, and manoeuvre around obstacles. These capabilities have opened up applications for search and rescue, surveillance, inspection of infrastructure, inspection of industrial plants, warehouse inventory checking, and package delivery. Additionally, Small flying robots are often cheaper than larger robotic systems, opening up more use-cases in high-risk environments, or where there is a restricted budget. In many of these applications there is also growing interest in high-speed and high-acceleration flight, so a robot can complete a task more quickly, and can reactively avoid collisions.

Another field for small flying robots is in space. In this domain, there has been growing interest to provide an autonomous, free-floating satellite as an assistant onboard space-stations. These systems also have applications to inspection and repair outside a space-station or on satellites. Additionally, there is demand in future space exploration missions for small flying robots to explore more rapidly than rovers and to access caves and cliff faces.

A significant strength of small flying robots is their 3D manoeuvrability and size, allowing them to operate in and around buildings and objects, to move into areas where other systems can not. Because of this strength, the most beneficial applications for small flying robots require flight near to obstacles. The applications for these robots also requires autonomy: the robot needs to determine where it is, what is around it, and how to get to its goal, all with its own sensors and computation. Achieving this autonomous navigation capability for operation near obstacles is one of the critical challenges for enabling the many applications of small flying robots.

### 1.1 Background

For small flying robots to achieve autonomy, they need to implement what will be referred to as the autonomous navigation stack, as depicted in Fig. 1.1. The stack includes several layers of capability starting from sensors to perceive the environment through to the control to move through it. While there is not always a strict division between layers and robotic systems may not require every layer, the stack is a convenient structure to discuss the different components required for autonomous navigation. Each layer in the stack will be described in general, and then the challenges relevant for small flying robots will be highlighted.


Figure 1.1. The autonomous navigation stack. A depiction of the different layers required for autonomous navigation

The sensing layer relates to the devices that are used to perceive the environment. This sensing helps the robot determine where it is, and what obstacles are around it. Possible sensors include monocular cameras, stereo cameras, depth cameras (such as RGBD cameras: Red, Green, Blue and Depth), lidar, radar, proximity sensors, GPS antennae, and inertial measurement units (IMUs). The particular selection of sensors depends on the desired tasks for a robot.

The image processing layer takes information from the sensors, such as the cameras or lidar, and extracts useful information for other layers of the stack. This information could be to help localisation, by detecting and tracking landmarks, to help the mapping of obstacles by computing 3D locations of observations, or to classify observations to guide higher-level task-planning.

Odometry is tracking the movement of a robot from a starting position. For ground-based robots, this can be done with wheel encoders, but for flying robots, Visual Odometry (VO) algorithms tend to be used. These algorithms use successive camera images of the environment to track movement. Distinct features are extracted from images and are matched in subsequent observations. The movement of the features from frame to frame gives information to update the estimate of the robot's motion.

Localisation is similar to odometry, in that the goal is to track the position and orientation (the pose) of the robot; however, localisation looks to give a global position with respect to a fixed reference frame. GPS can provide localisation information, but for applications flying in and around buildings, GPS is not reliable. Instead, localisation can be done with sensor observations of distinct landmarks to provide location information. The landmarks are similar in concept to how humans use distinct buildings,
structures and natural features as landmarks for navigation. For robots, these landmarks could be beacons or visual icons specifically designed for the task. Alternatively, lidar scans could be matched to a known 3D map of the objects in an environment. Another approach is to use visual features, similarly to VO. The map of landmarks or features that are used for localisation could be pre-mapped or could be generated online ${ }^{1}$, for operation in an unknown environment. This process of online localisation and map generation is known as Simultaneous Localisation And Mapping (SLAM). If using visual features, the algorithms are referred to as Visual SLAM (VSLAM). VSLAM differs from VO in that the features detected and tracked are stored in a global map, enabling the global pose to be detected when revisiting explored areas. The overlap of VSLAM with VO means that the one algorithm often fills both roles.

While SLAM does include a mapping component, often the map generated is purely for localisation and does not provide a useful representation for other tasks, such as representing obstacles or producing a visually detailed map. Therefore, other algorithms generate 3D maps for these purposes, using the sensor information and a known global robot pose from localisation. Often mapping can be hierarchical, with a large scale global map that is infrequently updated, and a small obstacle map around the robot that is updated rapidly.

The trajectory planning layer uses a map of the obstacles in the environment and a known robot pose to determine the path the robot should fly to reach a goal location. The planner layer itself has several layers, as depicted in Fig. 1.2. A particular robotic system may only have some subset of these layers. The highest layer is the decision making, to determine where the robot should go, given some activity-goal, such as exploration. Next, a global planner produces a long distance path to get from where the robot is to the goal, through a global obstacle map. This global plan can then used to generate waypoints, between which a trajectory is optimised to minimise traverse time, minimise control effort, or maximise smoothness (dynamic-optimality). Here we make a distinction between a path: which is a sequence of positions, and a trajectory: which is a time-dependent sequence of positions, and as such encodes velocity and acceleration. Flying robots need trajectories, as they incorporate dynamic considerations to ensure the systems can fly where planned. In addition to the optimised trajectory, there can also be local replanners that rapidly adjust a short-term trajectory to avoid obstacles.


Figure 1.2. The different layers of planning algorithms.

The near-term trajectories are tracked using the final layer of the stack: control. The control layer needs to take into account the dynamic restrictions of the robot to track the trajectory and reject

[^0]disturbances. Reactive collision avoidance can also be incorporated into the controller, to quickly avoid new obstacles.

Each layer of the autonomous navigation stack has specific challenges, which are in addition to the challenges of combining the different layers to give the desired capability for a given robotic system. For small flying robots, these challenges are enhanced due to limitations of size, weight and power. Being small in size, and needing to fly, these robots cannot carry large payloads, hence cannot carry large sensors or large processors. Additionally, small flying robots cannot carry large batteries; hence the power for sensors and computation is limited. Another challenge is that the robots are operating in 3D, a more complex problem than for 2D, ground-based robots. Finally, the requirement to be flying means that careful consideration needs to be taken of the dynamics.

The specific challenges in each of the layers, for small flying robots, will be discussed below, in addition to highlighting the current state-of-the-art for addressing those challenges.

### 1.2 Challenges and Current State-of-the-Art

For sensors, there have been strong demonstrations using lidar [52, 103], but the restrictions of small flying robots mean that these sensors, which are large and power-hungry, tend to be on larger systems. RGBD cameras are lighter, with lower power requirements and have frequently been used on flying robots [4, 63, 99, 163, 177]. Stereo cameras are emerging as a standard sensor modality [147, 172, 177], being small, lightweight, low-power sensors that can also provide depth information (through stereo image processing). Some systems operate with only a single camera and extract depth from the motion of the robot [61]. These cameras are complemented with IMUs and occasionally ultrasonic proximity sensors.

With stereo or RGBD cameras as the primary sensor, image processing requirements are to a) produce 3D positions of observations in each pixel (point clouds) from stereo or depth images, and b) extract visual features from the images for use in odometry and localisation. Visual features are distinct points in the environment that can be consistently detected and described from a wide range of viewpoints. The limited computational power onboard small flying robots means that these algorithms need to run efficiently, and often the 3D location of observations may only be computed for features, rather than the whole image.

VO is the leading approach for odometry on small flying robots, as there is limited information available other than the camera images (monocular, stereo or RGBD) and IMU readings. Some systems integrate with IMU output to perform Visual Inertial Odometry (VIO). There are numerous capable algorithms, including MSCKF [213], ORB-SLAM2 [163], DSO [55], SVO [68], ROVIO [19] and VinsMono [183]. These algorithms differ in the visual features used, and the method of performing the estimation.

Localisation and odometry tend to use the same algorithms for small flying robots by performing VSLAM. Leading VSLAM algorithms include ORB-SLAM2 [163], MSCKF [213] and VinsMono [183]. Additionally, VO algorithms could be combined with a back-end SLAM algorithm for longer-term navigation. These SLAM algorithms store a 3D map of the visual features used for localisation. The features in new observations are matched to the features in the map to give information on the robot
pose. More features are added to the map as the robot explores new areas. These VSLAM algorithms do not require a pre-mapped environment and can operate in real-time onboard small flying robots. Localisation can also be done using a physical, 3D map of the environment, but these algorithms require lidar, which is not ideal for small flying robots. Localisation can also be performed in pre-mapped environments, with stored visual features, such as the result from a VSLAM algorithm or from prior imagery of an environment [42]. Some systems do not do localisation at all, and instead, only use odometry to track position within a small area of operation.

The SLAM algorithms used for localisation produce a map of 3D features. While these features have 3D locations, the map is not a physical representation of the environment that can be used for planning. Therefore, it is currently common practice to use another algorithm that takes point cloud data from processed stereo or depth images, along with the estimate of the robot pose, to build a 3D map. Efficiently mapping in 3D brings a more significant challenge than mapping in 2D, with a significantly increased amount of information to consider. For the limited computational power on small flying robots, these maps tend to be discretised into 3D grids. Cells in the grid could contain a probability of occupancy in what is an occupancy grid, such as in OctoMap [91]. Alternatively, the cells could contain a signed distance to the nearest surface, called a Euclidean Signed Distance Field (ESDF), as in Voxblox [171]. These maps can be generated and updated online and provide the needed volumetric representation of the environment for planning. Often systems use a global occupancy map, that may be pre-mapped, and a regularly updated local occupancy map for collision avoidance [52, 63, 147, 177].

The occupancy maps tend to be a representation of the whole environment that has been observed. If instead, individual objects in the environment were mapped and modelled, then the resulting representation could have uses beyond just obstacle representation and localisation. For instance, 3D object representations play an important role in interaction with the environment, such as grasping [22, 144]. Object representations are also useful for object classification [17, 44], a capability that is important for feeding higher level logic, such as finding tools, and deciding how best to interact with them. An object representation that can be used for localisation, obstacles, grasping models and classification, would be highly valuable, however, such capabilities are currently difficult to achieve. Another benefit to representing objects is that dynamic obstacles can naturally be captured, by assigning a velocity to the object that is observed to move: something that can not be done with occupancy grids.

The challenge for planning trajectories with small flying robots is in efficiently generating collisionfree trajectories through obstacle-rich environments in 3D, with 6 degrees of freedom. The current leading approach is first to generate a set of waypoints with a global planner and then to optimise a trajectory through these waypoints to maximise smoothness. Obstacle avoidance relies on the collisionfree global plan, with the trajectory optimisation step having no consideration of obstacles [26, 32]. Other systems use rapid local replanning over a short planning horizon to avoid collisions and dynamic obstacles. To enable operation with limited computational power, the global planners run at a slow rate, and the local planners more frequently, but only with short trajectories.

For planning with quadrotors, a property of quadrotor dynamics called differential flatness is used to ensure dynamic-feasibility. This transformation allows trajectories for position and yaw to be planned, which can then be transformed into the full quadrotor state, including the desired rotor revolutions. Planning continuous trajectories in position, yaw and their derivatives leads to continuous controls,
helping to ensure that the trajectory can be flown.
The primary goal of control is to track a planned trajectory, with consideration of the robot dynamics and actuation restrictions. Such considerations are critical for flying robots. For quadrotors, the differential flatness property is again used to have a split between a position controller, and an attitude controller. The position controller runs a feedback loop to track the position and velocity of the trajectory and produce a desired acceleration vector. The differential flatness transformation then uses the acceleration and desired yaw to give the desired orientation for the attitude controller to track, giving thrust and moments. The controller by Lee et al [118] is the most commonly used and employs this hierarchical control architecture.

When looking at a complete flying robot system, they tend to take components of the layers discussed above. For example, an early example developing the algorithms for a complete system is presented in [115]. More recently, there are many field-tested examples, such as Mohta et al. [147] who achieve high-speed autonomous flight indoors and outdoors by implementing: VIO with stereo cameras and no localisation; mapping with a 3D laser scanner to have a 2D global occupancy map and 3D local occupancy map; and a global planner to produce collision-free paths. Another example is Perez-Grau et al. [177], who demonstrate indoor autonomous collision avoidance using stereo for VIO and an RGBD camera for localisation and mapping. They use a pre-mapped global occupancy grid and a local occupancy grid for collision avoidance. For planning trajectories, they use a long-term global planner, combined with a rapidly replanning local planner.

### 1.3 Gaps

For small flying robots, there are currently many very capable algorithms for each layer of the autonomous navigation stack and systems that have successfully demonstrated autonomous flight. However, gaps are still present between the current state-of-the-art and the desired capability. In general, the current leading algorithms are heterogeneous: multiple algorithms are needed to satisfy each layer of the stack [4, 63, 147, 170, 177]. SLAM is separated from 3D mapping, and trajectory planning is divided between global and local planners. For computationally limited systems, as with flying robots, there are potential benefits in having more homogeneous algorithms. There are also a number of gaps within the different layers of the autonomous navigation stack. These gaps will be elaborated below and are explored in detail in Chapter 2.

## 1. No map produced by SLAM with lightweight sensors is also useful for trajectory planning.

The leading SLAM algorithms use visual features as landmarks for localisation and mapping; hence the resulting map is a sparse set of points that are not useful for trajectory planning [19, 55, 163, 183]. A 3D, volumetric representation is required for trajectory planning algorithms, to adequately model the obstructions in the environment; therefore, a separate 3D mapping algorithm needs to be employed. This requirement could be alleviated by having a SLAM algorithm that immediately produces a map of obstacles. The result is potential gains in efficiency. Such a capability can be achieved by algorithms using lidar [16, 52], but not yet for RGBD cameras or stereo cameras. A central part of this concept is the method of representing the 3 D environment. No 3 D modelling method currently produces a
representation that can effectively be used for mapping, localisation and as an obstacle, with lightweight sensors. Further, no methods of obstacle mapping represent 3D objects individually, an area where there are potential benefits for feeding object classification and interaction as well as representing dynamic obstacles.

## 2. The leading approaches for trajectory planning near obstacles are hierarchical, sacrificing optimality.

For the current state-of-the-art, a global planner generates collision-free paths, without considering dynamics to produce waypoints for a trajectory optimiser [26]. The optimiser does consider dynamics but does not include obstacles and instead relies on the global plan to produce collision-free trajectories. The overall result can be less optimal, in terms of trajectory smoothness, than a long-term trajectory optimiser that considers obstacles. A sacrifice in smoothness leads to a reduction in trajectory tracking performance, especially for high-speed flight.

## 3. Methods to avoid dynamic obstacles for flying robots are only local or are very conserva-

 tive.Reactionary planners and control are used to avoid dynamic obstacles for flying robots, and do so very effectively; however, they operate over a short horizon, with the primary focus to avoid collisions $[5,13,176]$. Therefore the resulting actions could push the robot in a direction that is inferior for the longer-term trajectory. Dynamic obstacles could be included in a trajectory optimisation algorithm to consider a large planning horizon, but existing approaches to do so are very conservative [174, 227].

## 4. Existing methods for the quadrotor differential flatness transformation have sensitivities near a singularity.

There are several methods to address the singularity in the differential flatness transformation, but there are orientations near the singularities where these methods fail [118, 121, 139, 216]. These issues occur during aggressive flight, such as can be required for obstacle avoidance and high-speed flight amongst obstacles.

## 5. There has not been consideration of how the method of including obstacles in trajectory planning impacts the dynamic-feasibility of trajectories.

Different methods of considering obstacles in planning (e.g. [26, 32]) can affect how dynamically-optimal the resulting trajectory is. The dynamic-optimality affects how easily the trajectory can be tracked: the dynamic-feasibility. When flying near obstacles, this factor is critical, yet there has not been any analysis to characterise which methods of considering obstacles provide the most dynamically-feasible trajectories.

### 1.4 Focus of Thesis

The goal of the thesis is to contribute to enhancing the capability of flying robots to navigate autonomously near to obstacles. This goal is realised through a focus on four main layers of the autonomous navigation stack: localisation, mapping, planning and control. These layers are developed to address the gaps identified above. In general, an approach is taken to strive for greater homogeneity in algorithms, to have one algorithm filling many roles, rather than multiple separate algorithms. In particular, the focus areas of the thesis are:

1. Combined localisation and 3D mapping in one algorithm with a common 3D representation objects in the environment.
2. A trajectory optimisation algorithm that includes static and dynamic obstacles directly in the optimisation.
3. Handling singularities in the quadrotor dynamic model for control.
4. Analysing the dynamic-feasibility of trajectories for flying near obstacles.

Focus 1 aims to combine the localisation and mapping layers to have one SLAM algorithm that also produces a 3D map of obstacles that can be used for trajectory planning. This algorithm uses Non-Uniform Rational B-Splines (NURBS) as the single 3D representation for localisation, mapping and obstacle representation, and is referred to as NURBS Localisation And Mapping (NURBSLAM).

Focus 2 address the planning layer, by enhancing a trajectory optimisation algorithm to provide a middle ground between a global planner and a local optimiser. The algorithm is called the Admissible Subspace TRajectory Optimiser (ASTRO). ASTRO optimises the coefficients of a polynomial to minimise a cost function that combines a trajectory cost (smoothness) and obstacle costs. The algorithm is developed to generate solutions with many obstacles, including dynamic obstacles, all of which are included directly in the optimisation.

Focus 3 investigates the control layer by considering quadrotors and performing an analysis of the differential flatness transformation. In particular, the work focuses on issues near singularities in the transformation.

Focus 4 analyses a combination of the planning layer and the control layer by testing how variations in trajectory planning algorithms affect how well a quadrotor can track a trajectory in flight.

Focus 1 and 2 are also brought together to test the concept of homogeneous localisation and mapping with a combined system demonstration of NURBSLAM and ASTRO in a novel robotics simulation tool.

### 1.5 Contributions

The main contributions of this thesis are summarised below. The relation of these contributions to the gaps identified, and the relevant section in this thesis are summarised in Table. 1.1.

### 1.5.1 Localisation and Mapping

One of the main contributions in this thesis is NURBSLAM, which uses one 3D representation for mapping, localisation and obstacle representation by modelling objects. NURBSLAM is designed to operate with RGBD data, and is enabled by the following contributions:

## A. Application of NURBS for Mapping.

An algorithm is presented that uses NURBS for online mapping of objects from point cloud data. The algorithm includes scan processing to produce a structured mesh, initial surface fitting, and extending surfaces with new observations. The result is a representation of objects, which has potential future applications for grasping and manipulation. The algorithm is shown to be able to produce accurate representations of objects, with Root-Mean-Square Errors of under 5mm when observations are made from a known pose.

## B. Application of NURBS for Localisation.

The shape of a NURBS surface is exploited to provide detailed information for localisation. This information is used by aligning new point cloud observations of an object to the NURBS surface. Tools from the Point Cloud Library [200] are used to find point correspondences and alignments. The resulting transformation is included in an Extended Kalman Filter (EKF) as an observation to update the state estimate, with a Multiplicative EKF used to estimate attitude. The approach can achieve mean tracking errors of under $1 \%$ of the total trajectory length when localising to pre-mapped objects.

## C. Application of NURBS as obstacle representations.

Surface points and surface normals from a NURBS surface are used to compute signed distances to query points on a trajectory. The algorithm also computes an approximate distance gradient with a vector from a query point to the nearest location on the surface. The signed distance and gradient information provide what is required for trajectory optimisation algorithms. The method of using NURBS as an object is successfully demonstrated, and the evolution of the signed distance is shown to have the desired traits for an obstacle representation. Being a representation of obstacles, the NURBS has the potential to be extended to represent dynamic obstacles.

## D. Application of NURBS for SLAM.

Localisation and mapping with NURBS are combined to perform SLAM, online, in an unknown environment. This capability is demonstrated in simulated test cases, showing tracking errors at $2 \%$ of the total path length, and mapping errors as low as 2 cm . The algorithm is susceptible to bad alignments, though, when there is shape ambiguity but recovers quickly for good odometry tracking.
E. System demonstration of NURBSLAM with trajectory planning.

The complete system of NURBSLAM with trajectory planning is successfully demonstrated, showing that a single representation can be used for mapping, localisation and obstacle representation, without the need for lidar. NURBSLAM is also compared to the current state-of-the-art in heterogeneous approaches: ORB-SLAM2 [163] with Voxblox [171]. NURBSLAM is shown have a higher computational load, and be less accurate, but is more robust and has with the ability to quickly recover from errors. Additionally, NURBSLAM can successfully operate in cases with sparse visual features and changing light conditions, where ORB-SLAM2 fails.

## F. Development of a novel robotic simulation framework.

As part of a collaborative effort, a connection between the game development engine, Unreal and the Robot Operating System (ROS) was used to create a framework for testing autonomous navigation algorithms in environments with high visual fidelity. The capability of this tool for assessing autonomous navigation algorithms is demonstrated by testing NURBSLAM and the current state-of-the-art.

### 1.5.2 Trajectory Optimisation

Another central contribution of this thesis is the development and extension of ASTRO. The algorithm has been enhanced from an earlier version to provide the following contributions ${ }^{2}$.

## A. Long-term trajectory optimisation with obstacles

ASTRO provides the middle ground between a global planner that generates collision-free paths without dynamic considerations and a local optimiser that produces dynamically-optimal trajectories without consideration of obstacles. By including obstacles directly in the optimisation, ASTRO can produce dynamically-optimal and collision-free trajectories. These trajectories can be generated without the need for a global planner for moderately complex environments. A large set of simulated examples demonstrate these capabilities of ASTRO. The following two contributions enable ASTRO to efficiently generate solutions to these complex planning problems with many constraints.

## B. Enhanced constraint formulations to enable the application of ASTRO to a wide

 range of problems.ASTRO has been made flexible to include a variety of constraints that can represent numerous obstacle shapes, free-space restrictions or performance limitations. This flexibility of representation allows the appropriate constraints to be defined for a given scenario, resulting in a planning problem that is well suited to ASTRO. An analysis of the constraint formulations provides insight into the best way to handle constraints in the optimisation for efficient solution generation. Simulations demonstrate how the flexible constraint formulation in ASTRO can allow the algorithm to be applied to a wide range of scenarios.

## C. Development of techniques for optimisation to enable efficient solutions to problems with many constraints.

A suite of techniques has been implemented in ASTRO to improve the optimisation performance and enable solutions to be generated when there are many constraints. These techniques include a quadratic line-search, iterative sub-problems, adaptive weights for constraint costs and randomised perturbations. The optimisation performance is analysed in detail to demonstrate the benefit of these techniques

## D. Formulation to include dynamic obstacles in the optimisation

Dynamic obstacles are included in ASTRO with a dynamic model for their motion, allowing the obstruction to be time-dependent. This approach makes the trajectories less conservative, and allows more optimal trajectories, by accounting for the free-space that becomes available when an obstacle moves. The size of dynamic obstacles is increased as a function of time to account for the increase in the uncertainty of the obstacle position and to ensure safe trajectories. Tests cases with numerous

[^1]static and dynamic constraints demonstrate the challenging classes of problems that can be solved by including dynamic obstacles in the optimisation.
E. Analysis of on-orbit tests from the International Space Station

An earlier version of ASTRO was tested on free-floating robots on the International Space Station. An analysis of the results from the tests, in particular with dynamic obstacles, identified several lessonslearned for trajectory optimisation of free-floating satellites, which has informed further development of ASTRO.

### 1.5.3 Analysis of the Differential Flatness Transformation for Quadrotors

The singularity in the differential flatness transformation for quadrotors has been analysed in detail in this thesis to address the issues with existing methods. The main contributions from this work are:

## A. Analysis of the limitations with existing methods

A thorough analysis is performed to identify where existing methods of handling the singularity in the differential flatness transformation fail, showing where there are issues in orientations near the singularity. The analysis provides valuable information for those using the existing methods to understand where issues may arise.

## B. Development of a new Combined Method

A combination of existing transformation methods is proposed, where multiple transformations are computed, and the result that is closest to the current state of the quadrotor is selected. The combined method is shown to provide robust performance throughout all scenarios tested.

## C. Flight tests to demonstrate challenges and solutions

A set of aggressive flights tests showed that the issues with the differential flatness transformation can be experienced in flight and that the new Combined Method successfully addresses those issues. The tests also highlighted remaining challenges in tracking yaw through orientations where it is ill-defined.

### 1.5.4 Analysis of Dynamic Feasibility of Trajectories for Quadrotors and the Impact of How Obstacles are Considered

Three trajectory optimisation algorithms: ASTRO, the UNConstrained Optimiser (UNCO) [26] and the Tube and Cube constrained Optimiser (TACO) [32], were compared to assess the impact that the method of considering obstacles has on dynamic-feasibility. The algorithms were compared in batches of tests cases, and with trajectories tracked in flight. The contributions in this area are:
A. Implementation of ASTRO for quadrotors and comparison with the state-of-the-art

ASTRO is extended to optimise for quadrotor dynamics and perform multi-segment optimisation through a set of waypoints. ASTRO provides an alternative method of considering obstacles by including them directly in the optimisation. This approach allows fewer waypoints to be used, and smoother trajectories to be generated. ASTRO is also able to represent free-space restrictions in the optimisation, providing the capability to produce more conservative trajectories.

## B. Analysis of tracking performance in flight

Trajectories planned by ASTRO, UNCO and TACO in an obstacle-rich indoor environment were flown with a quadrotor to assess the tracking performance. The tests highlighted that different methods of considering obstacles do impact the dynamic-feasibility of the resulting trajectory. Additionally, the tests showed that ASTRO provides dynamically superior trajectories, with better tracking performance when flying at high speed, near obstacles.

TABLE 1.1. Summary of gaps and contributions

| \# | Gap | Contribution to address gap | Chapter/Section |
| :---: | :---: | :---: | :---: |
| 1 | No map produced by SLAM with lightweight sensors is also useful for trajectory planning. | Development of algorithms to use Non-Uniform Rational B-Splines (NURBS) for online mapping, localisation and obstacle representation with RGBD data: NURBSLAM. | Chapters 3, 7 |
| 2 | The leading approaches for trajectory planning near obstacles are hierarchical, sacrificing dynamic-optimality. | Extension of the Admissible Subspace TRajectory Optimiser (ASTRO) to provide a middle-ground between local optimising and long term planing by including obstacles directly in the optimisation. This contribution includes enhancing constraint formulations and optimisation techniques in ASTRO to enable the efficient generation of solutions to problems with numerous constraints. | Chapter 4 |
|  | Methods to avoid dynamics obstacles for flying robots are only local or are very conservative. | Development of a class of dynamic obstacles in ASTRO that model the dynamics of obstacles inside the optimisation to enable less conservative trajectories. The size of the obstacles are grown to account for the uncertainty in their position. | Chapter 4 |
| 4 | Existing methods for the quadrotor differential flatness transformation have sensitivities near a singularity. | 1) Analysis of existing methods to address the singularity and identification of their limitations. <br> 2) Formulation of a combined method for the transformation that is robust to sensitivities near the singularity. <br> 3) Flight tests to demonstrate issues can be experienced in flight, and that the combined method addresses those issues. | Chapters 5, 6 <br> Sections 5.1, 6.2 |
|  | There has not been consideration of how the method of including obstacles in trajectory planning impacts the dynamicfeasibility of trajectories. | 1) Analysis of three different obstacle avoidance strategies for trajectory optimisation of quadrotors including an extension of ASTRO to apply to quadrotors. <br> 2) Testing of the algorithms in flight to highlight differences in dynamicfeasibility. <br> 3) Demonstration that ASTRO for quadrotors can give the best dynamicfeasibility when flying at high speeds near obstacles. | Chapters 5, 6 Sections 5.2, 5.3, 6.3 |

### 1.6 Outline of Thesis

The remainder of the thesis is structured as follows:

| Chapter | What is presented |
| :---: | :--- |
| 2 | A thorough review of the literature and current state-of-the-art for autonomous <br> navigation of small flying robots. This review covers localisation, mapping, <br> trajectory planning, control and complete systems. The chapter provides the <br> context for the work in this thesis. |
| 3 | A presentation of NURBSLAM, the combination of localisation, mapping and <br> obstacle representation. This chapter includes an analysis of 3D modelling <br> algorithms, a description of NURBSLAM and demonstration of the algorithm in <br> simulated test cases. |
| 4 | A detailed description of the trajectory optimisation algorithm, ASTRO, includ- <br> ing a thorough presentation of theory, analysis of the algorithm and tests on a <br> variety of planning scenarios. |
| 5 | Quadrotor applications for planning and control. First, a description and analysis <br> of the differential flatness transformation is presented, followed by tests in <br> simulation. The chapter also describes how ASTRO is applied to quadrotors and <br> compares the performance with the state-of-the-art in a batch of test cases. |
| 6 | Hardware flight tests with quadrotors to analyse both the differential flatness <br> transformation and how the dynamic-feasibility of trajectories is affected by the <br> method of considering obstacles in trajectory planning. |
| 7 | Demonstration of the combined NUBRSLAM and ASTRO system, with a com- <br> parison to the state-of-the-art in heterogeneous algorithms. Tests are performed <br> in a new simulation tool with high-fidelity visuals. |

The thesis ends with a conclusions chapter before summarising avenues for future work.

## Background and Related Work

This thesis studies methods to enhance the autonomous navigation capability of small flying robots such as quadrotor unmanned aerial vehicles (UAVs) and free-flying space-based robots. The desired capability is for a robot to be able to use its own sensors and processors to: determine its location, build a map of the environment around it, and plan a trajectory to move through that environment. These capabilities touch on all aspects of the navigation stack described in Chapter 1: sensors, image processing, odometry, localisation, mapping, planning and control.

This chapter provides a background on the key components of the navigation stack before analysing the state-of-the-art at the time of publication. In particular, there is a focus on algorithms that are suitable for small flying robots. Localisation will first be reviewed, where the leading algorithms perform Simultaneous Localisation And Mapping (SLAM). SLAM incorporates the top half of the stack: sensors, image processing, odometry and localisation. While SLAM does include a mapping component, it will be shown that current leading SLAM algorithms require a separate mapping algorithm. Therefore a review of 3D mapping algorithms is presented, including a review of methods to model 3D objects. Trajectory optimisation algorithms are then reviewed, with a particular focus on the consideration of static and dynamic obstacles. An important component of trajectories is their dynamic feasibility: how well a robotic system can fly the trajectory. The methods to ensure a dynamically-feasible trajectory are reviewed for one class of hardware systems, quadrotors, which also includes an analysis of the bottom of the navigation stack: control. The complete navigation stack is then considered by comparing existing systems for autonomous navigation of quadrotors.

In addition to a background on autonomous navigation for small flying robots, this chapter highlights the gaps in the literature. These gaps are summarised at the end of this chapter, along with a summary of how the work in this thesis addresses those gaps.

### 2.1 Simultaneous Localisation And Mapping (SLAM)

One of the key requirements for autonomous navigation is localisation, to know where the robot is positioned in the world. Localisation can be done with global positioning systems, such as GPS, but this approach is not robust for operations indoors, in dense cities, underwater or for space exploration. GPS has limited precision as well, which can make the technology insufficient for navigation near obstacles. Hence, localisation for robots is generally done by making observations of the world, with sensors such as a camera, lidar, proximity sensor or a combination of such sensors. These observations detect features in the environment and track the position of these features with respect to the robot, to estimate the movement of the robot. These features can be thought of as landmarks, similarly to how humans may use distinct buildings or natural features and landmarks for determining where they are. For robotics, these landmarks can take a variety of forms, including 3D structures, detailed visual tags, distinct visual points or visual appearance. If a map of features is already known and is provided to the robot, then the robot can match the features it sees to this map to determine its location. If such a prior map is not known, then the robot can match the features it sees to features it has previously seen. The storage of observed features is the process of mapping. Building a feature map while using it to localise is known as Simultaneous Localisation And Mapping (SLAM) and is a coupled estimation of the robot state and the state of the features in the map. SLAM has broad applications to all areas of mobile robotics, wherever a robot needs to navigate in an unknown, or partially-known environment. Equally, there is a broad range of SLAM algorithms, each with different strengths. The state-of-the-art for these algorithms will be discussed here.

In particular, there are three key components of SLAM that will be discussed:

1. Features: what features are used for localisation and how are the features linked with the sensors used?
2. Map Representation: how are the features stored and grouped to make the map?
3. Estimation Algorithm: given observations of features and the map, how is the joint estimation of state and map performed?

Through the range of variations of the components above, there are a set of categories within which many of the leading algorithms in the literature can be placed. The first category of algorithms directly use 3D data as features, with point clouds from a lidar, or a depth camera (for example, a Red, Green, Blue and Depth camera: RGBD). Visual cameras have seen much application in recent times, and algorithms using visual features are split into categories of direct, indirect, semi-direct, appearance-based and geometric-feature-based. Each of these categories will be discussed in relation to the three components listed above.

First, a high-level overview of the key concepts and terminology of SLAM is given, to give the context for a discussion of the different categories of SLAM algorithms.

### 2.1.1 Key Concepts and Terminology

The basic premise of SLAM is to estimate both the robot state (position and orientation) and the state of a map. In the simplest sense, the map could be thought of as a set of features, also known
as landmarks. Fig 2.1.a depicts the SLAM problem for the case where the landmarks are assumed to be point landmarks. The robot makes predictions on its movement to update its state estimate (robot position and orientation). This prediction uses either IMU information, odometry information, or a simple motion model, such as constant acceleration or constant velocity. Observations of landmarks are matched to predicted landmark observations from the map. The error between the predicted landmark and the observed landmark, sometimes referred to as the innovation, can be used to update the estimate of both the robot state and the landmark positions. It is the observation of multiple landmarks that helps to update the estimates accurately: this can be thought of as using multiple observations to triangulate position. In contrast to using GPS or fixed beacons, though, the position of the landmarks is not known; hence SLAM is a joint estimation of the robot state and the states of the landmarks. As a robot moves around an environment and makes more observations, the correlations between landmarks and the robot can become stronger to reduce the uncertainty in the map and robot state, as depicted in Fig. 2.1.b.


Figure 2.1. The joint estimation of the robot state and landmarks states that characterises SLAM. (a) Robot states, $x$, are tracked over time, with dynamics, $u$ connecting states and observations, $z$, of landmarks, $m$, which adds information to the joint estimation. (b) Depiction of the correlation between landmarks and the robot. As the correlations strengthen (thicker lines), the esimates become more certain. Images from DurrantWhyte et al. [92]

The features used for landmarks are not always point-landmarks, as will be discussed in later sections, and the map representation can vary. Nonetheless, the same conceptual understanding applies: observations are matched to features from a map, and the error between the predicted and the actual observation is used to update estimates of both the robot pose and the map.

### 2.1.1.1 Estimation Approaches

The estimation of robot states and the map of landmarks can be performed in a number of ways. The problem is commonly formulated in a probabilistic sense with a joint posterior of the robot and landmark states given the past states, movements and observations. This framework naturally lends itself to a filtering-based approach, using a variant of a Kalman filter, where there is a dynamic prediction-
correction sequence. The prediction step uses the motion model, IMU or odometry to estimate the new state of the robot, which increases the uncertainty. The correction step then uses the observations of the landmarks to update both the robot state and the map, reducing the uncertainty. Kalman Filter approaches have seen frequent use, being a convenient way to handle the probabilistic representation of the SLAM problem. The type of filter can vary, from being an Extended Kalman Filter (EKF) [19, 46, 92, 137, 225], a particle filter [31, 63, 92, 148, 177], a combination of particle filters and EKF filters [23], or more modern variants to the Kalman filter [213]. A challenge for filtering-based methods is that there can be large computational loads when the environment grows larger and the number of states (robot and landmark states) to track increases.

More recently, graph-based approaches have shown strong performance, where the map is represented by a connected set of robot poses (the graph nodes) and the landmarks observed at those poses. These graphs are called pose-graphs. Any two nodes can be connected by the robot motion between them as well as the common landmark observations (called a co-visibility graph). The graph-based representations enable the use of graph-optimisation techniques to update the estimate of the robot and map in a way that is more efficient when there are many landmarks in the environment. The improved efficiency with scale is because the pose-graph approach estimates a set of pose-states, with landmarks projected from the poses, whereas filter approaches estimate the states of the map feature, as well as the current state. Another contrast between pose-graphs and filters is that the graph maintains information on the history of the trajectory, rather than having that history represented in a covariance. The resulting optimisation is more computationally expensive than filters but can produce more accurate results. This optimisation can be done over the whole graph, or a recent history of the graph to improve computation time. This general field of estimation is called Pose-Graph Optimisation (PGO) and has benefited from recent mathematical advances that have allowed the problems to be solved efficiently, such as in the open-source tool GTSAM [47], making PGO amenable to SLAM. Graph-based representations have been successfully used in a wide range of state-of-the-art algorithms [55, 57, 119, 143, 163, 183].

### 2.1.1.2 Data Association

A component of SLAM that is critical to strong performance is the accurate matching the observed features to the map features: a process called data association. When selecting the features to extract and the map representation, the method of data association is of prime consideration. Ideally, features can be represented with a detailed descriptor that includes information beyond just a position. The descriptor aids robust data association, to ensure the observed feature is matched to the correct map feature. False data association, when features are incorrectly matched, can be fatal for a SLAM algorithm, leading to an incorrect update to the state and map estimates from which the algorithm may not be able to recover.

The nature of the feature descriptors depends on the SLAM algorithm, ranging from image features, structure in the environment and visual textures. The discussion of different SLAM techniques will talk about the features used and how data association is performed.

Another approach to robust data association is to take advantage of a large set of features to determine the robot pose that satisfies the close alignment of the most features. Such an approach requires rejection of outliers to not corrupt the solution with false associations. Algorithms like Random

Sample Consensus (RANSAC) [65] are commonly used for outlier rejection. A simple example of such data association is Iterative Closest Point (ICP) [228], for cases where features are 3D points. The criteria for outlier rejection is the distance between an observed 3D point and the closest map feature. The inlier sets (those points not rejected) are used to estimate an update in the relative pose of the robot to best align the points. This alignment is applied to all points, and the process repeated: finding the closest points, then outlier rejection, then pose transformation until a convergence criterion is met. The result is that the observation has been transformed to best align all points with the map. Each inlier observation feature is then associated with the closest feature point on the map. ICP can be used for matching point cloud observations (large sets of 3D points) to a physical 3D map as a data association step. More commonly, though, the pose transformation computed by ICP to align observed points and mapped points is directly used to inform the estimate of the motion of the robot [141, 168]. When matching point cloud observations to a point cloud map, the process is often referred to as scan registration.

### 2.1.1.3 Loop Closure

For long term navigation in large environments, it can be important to recognise that a location has been revisited and to utilise that information to correct the drift in the robot and map estimates. This recognition and correction process is known as loop closure. The first step in loop closure is recognising that a location has been revisited: loop detection. Data association of the features themselves can be used for loop detection, by matching observations at the end of a loop to features stored at the start of the loop. In large environments though, the drift in the estimate of the map and robot states often means that a complete loop is not predicted with the map even when a physical loop has been travelled. Therefore the features at the start of the loop and the current observations are too far away to be matched (see Fig. 2.2.a). To address this challenge, algorithms employ place recognition as a separate algorithm to detect loop closures based on the appearance of a scene. Techniques such as visual bag of words can be used to describe the appearance of a given camera view to then be used to detect loops [162].

Once a loop has been detected, then the new information is used to perform a loop correction to do a global adjustment to the map and robot trajectory. The algorithm for the loop correction depends on the map representation used. For example, an effective approach for graph-based methods is to run a pose-graph optimisation with extra edges in the graph from the loop closure. The result of loop correction is shown in Fig. 2.2.b:

Loop closure algorithms can also be used for relocalisation if a robot becomes lost or starts in an unknown location of a mapped environment.

### 2.1.1.4 Summary

What has been described here is a conceptual overview of SLAM and some of the key terms and algorithmic components. Not all SLAM algorithms use all of the techniques described, and the combination of techniques that are used can be a point of comparison between SLAM algorithms. The following


Figure 2.2. Demonstration of loop closure with ORB-SLAM2 [162]. (a) A loop is completed, but the error of the estimate of the robot pose means that a loop is not predicted. The visual bag of words approach detects the loop though, as indicated by the blue line. (b) With a loop detected and corrected, the trajectory and map now aligns to complete the loop. Images are from [162].
sections review particular types of SLAM that vary based on the sensors required, the features used, the map representation and the method of estimation.

### 2.1.2 Point Cloud SLAM

Point Cloud SLAM algorithms directly use 3D point observations of the environment (point clouds) as the features to match against a 3D model of that environment. Each 3D point is matched to the closest 3D point in the model, and the difference between them is used to inform the update of the state and map estimates. The map is a 3D model such as a point cloud or an occupancy grid (3D cells that are either occupied or free ${ }^{1}$ ) and data association is achieved by minimising the distance between the set of observed 3D points and the map 3D points. ICP, or a similar variant, is commonly used in these algorithms to compute the transformation to update the position of the robot.

Point cloud algorithms require an accurate point cloud from the sensors and hence tend to be limited to the use of lidars, or RGBD (colour and depth) cameras. Lidar can either operate in 2D or 3D with the use of a gimbal or a lidar with multiple scan lines. Lidar with point cloud SLAM is used with great success for ground-based robots, with many readily available algorithms, such as in the Robot Operating System, as reviewed by Santos et al. [203]. Ground-based robots can use 2D representations of the environment, but the algorithms are more difficult to implement for 3D navigation.

There are a few notable examples in 3D, with varying map representations and optimisation approaches. Fang et al. [63] matches point cloud observations from an RGBD camera to an occupancy grid, which is produced offline. The estimation is performed with a particle filter but only to update the state estimate: the process is purely to provide global localisation of the robot and relies on VIO (described in Section 2.1.3) to track position; therefore the approach is not a full SLAM solution.

[^2]A similar approach is taken by Perez-Grau et al. [177], who use an RGBD camera to create a point cloud and use that point cloud to match to a 3D probability grid, where a field of occupancy probability is sampled at grid points. Similarly to Fang et al. [63], their approach relies on visual inertial odometry and a map that has been previously built. They use a Monte Carlo Localisation approach to estimate the pose of the robot, given the point cloud observations. Both [63] and [177] use a pre-mapped environment, hence are performing localisation, rather than SLAM.

One algorithm that does do full SLAM with point cloud observations is presented by Droeschel et al. [52]. The map representation is an occupancy grid where each grid stores a surfel: a mean and covariance of the points within that cell. Observations are grouped into surfels and matched to the map of surfels, using a Gaussian mixture observation model in an optimisation-based approach to determine data association and pose transformation. The map is generated online, in a local area around the robot. For larger scale SLAM, local surfel maps are stored in keyframes and a graph optimisation approach employed.

Full 3D SLAM with matching of point clouds to a point cloud map has also been demonstrated by Kaul et al. [103], with operation onboard a multi-rotor UAV. They use a novel algorithm for scan registration (matching an observed point cloud to the map point cloud) to incrementally localise the robot as well as build a global map.

The algorithms of both Droeschel et al. [52] and Kaul et al. [103] rely on observations from a lidar that is either actively or passively actuated. A downside to these techniques is that they required heavy sensors with large power consumption as well as a large amount of processing power to run the algorithms ${ }^{2}$. RGBD sensors require less power, but no existing technique can perform online localisation and mapping with them. Therefore, there is a general trend toward using lighter, passive sensors such as visual cameras. These techniques will be discussed below.

### 2.1.3 Visual SLAM - Preliminaries

Visual SLAM algorithms are becoming more and more popular due to the low cost, size, weight and power of visual cameras. This trend has been supported by the enhanced capability of digital cameras with suitable properties for computer vision applications. The use of visual cameras is of particular value to small flying robots, where there are tight constraints on weight and power.

Visual SLAM algorithms take on a variety of forms, which will be discussed and compared in the sections below. First, this section will overview terminology, key points of comparison and common components of visual SLAM algorithms. For a detailed overview of the key algorithmic concepts in visual SLAM, refer to the tutorial presented by Scaramuzza and Fraundorfer [70, 205].

### 2.1.3.1 Determining Depth

The sensors used in visual SLAM can either be monocular cameras, stereo cameras or multiple-camera arrays. Stereo cameras enable a depth estimate of a pixel in the image to be determined from a single observation using the disparity in pixel locations from left to right camera. This stereo depth perception

[^3]is similar to the way that human eyes perceive depth. Multi-camera systems can extend this concept to more than two cameras by making multiple stereo pairs. In contrast, monocular cameras require motion to determine the 3D structure of observations, using approaches called Structure From Motion (SfM) [106]). SfM can be thought of as constructing a virtual stereo pair of images from two monocular images at different locations. Without a physical baseline (such as the distance between two stereo cameras), the depth estimates from monocular techniques have a scale ambiguity, though: i.e. the relative size and location of observations can be determined, but an absolute size can not. The scale ambiguity can be resolved if an IMU is used as this will give information on the effective baseline between two monocular images. Stereo and monocular cameras are the most popular, with stereo benefiting from direct 3D measurements but at the expense of additional computation for processing two images. Stereo cameras also require an additional camera and a rigid mounting to have a consistent baseline between the two cameras.

### 2.1.3.2 Visual SLAM Classes

There are two main classes of visual SLAM algorithms: indirect and direct. Indirect algorithms extract visual features from the environment and use the error between the 3D location of the features to update the estimate of both the robot pose and the map. The estimation for indirect methods aims to minimise the geometric error between features. Direct methods, in contrast, directly use pixels in an image as the features and look to minimise photometric error: the differences in pixel intensities. A differentiating component of indirect SLAM algorithms is the type of visual features used. For direct SLAM algorithms, a differentiating factor is the choice of which pixels are used in an image, either with all pixels (dense), a subset of pixels (semi-dense), or patches of pixels (sparse), as depicted in Fig. 2.3.


Figure 2.3. Depiction of the different variations of direct SLAM methods. Different amounts of pixels are used ranging from (a) dense, (b) semi-dense and (c) sparse. Images from [68].

Another axis of classification is the method of estimation. Along this axis, one end is filter-based approaches, estimating the current state and covariance along with the map-feature-states. On the other end, there are pose-graph-based approaches, building a graph and optimising over that graph. To limit the size of this review, the comparison of this axis of classification is included throughout the discussions on indirect and direct visual SLAM. Refer back to Section 2.1.1.1 for a summary of the key differences between filter approaches and pose-graph approaches.

### 2.1.3.3 Common Algorithms

A common algorithmic feature for visual SLAM is a process called Bundle Adjustment (BA) [218], where the bundle of observed features at a given camera pose can have their positions in space adjusted by modifying the pose of the camera. With many poses stored and their corresponding bundles of features, BA will adjust the poses to best align all features.

Another common algorithm employed is Perspective from $n$ Points ( PnP ), which is used in estimating the position of the robot from matched observations (an example algorithm is [150]). PnP operates similarly to ICP by matching observations of a set of features to a stored map of features to determine the motion of the robot. Where ICP matches 3D points to a 3D map, though, PnP matches 3D points to 2 D image points. The stored map features in 3D space are projected onto image space and compared against the observed features in the image. The algorithm computes the transformation of the robot pose to minimise the reprojection error: the error in the 2 D space of the image. This transformation is used to update the estimate of the robot pose.

### 2.1.3.4 Visual SLAM Terminology

The literature of visual SLAM algorithms differentiates between Visual Odometry (VO), Visual Inertial Odometry (VIO) and visual SLAM (VSLAM). There are many common elements between each of the algorithms, and the differences can be subtle. The difference between VO and VIO is that VIO uses an IMU to aid in position estimation. VSLAM could be with or without an IMU. Algorithms without integration of an IMU simply use each observation to update the robot pose estimate.

Regardless of IMU integration, the primary goal for VIO and VO algorithms is to track the position of the robot, i.e. odometry. Nonetheless, in tracking the position of the robot, these algorithms are running a SLAM algorithm: detecting features and updating a map of those features. The differentiation to VSLAM is that VSLAM performs loop closures, allowing the creation, maintenance and use of a large scale map. To confuse the matter further, often VO or VIO algorithms can be combined with a back end SLAM algorithm to achieve VSLAM. For the discussion here, the important components to consider are the features, the map representation and the estimation methods, which can be equivalently compared between VO, VIO and VSLAM. The capability for loop closure or the use of an IMU are just two factors that will be discussed below.

### 2.1.3.5 Performance Comparisons

Many of the references describing the algorithms include a performance comparison with other, state-of-the-art algorithms. For a complete comparison, Delmerico et al [48] present a benchmark analysis for monocular VIO algorithms. Their tests are on hardware suitable for small flying robots, with datasets recorded by quadrotors. Li et al. [120] also provide a broad assessment, with tests of 8 algorithms that do not require an IMU, on indoor, outdoor and underwater datasets. They also compare each algorithm on the datasets provided in the original publications of each algorithm.

Presented below is a discussion on the leading algorithms for VSLAM, describing the features used, the map representation, the method of estimation and the capability demonstrated.

### 2.1.4 Indirect Visual SLAM

Many of the most successful VSLAM algorithms use an indirect approach: extracting image features from the environment and tracking those image features as the robot moves through the environment [104, 119, 163, 183, 213]. There are two components to using image features: features detectors and feature descriptors. Detectors extract the feature points from an image by running a sequence of image processing steps and extracting distinct points of interest. Descriptors provide a detailed description of the feature. The description is represented by a vector of parameters that are computed based on the surrounding pixels. Two examples of image features being extracted are shown in Fig. 2.4. The ideal detector is to able to extract the same features from a wide range of view angles, rotations, changes in scale, blurring and illumination changes. The ideal descriptor enables robust data association between these features: to accurately match a feature when observed again from a different view angle, scale, illumination, etc. Both descriptor and detector are ideally very fast to compute so that features can be extracted from a stream of images, as is required for VSLAM.


FIGURE 2.4. Examples of image features being extracted. (a) SURF features, in red, on an example image, depicting the use of surrounding pixels (image is from OpenCV Python tutotials [149]). (b) ORB features, in green, highlighted on an image (image is from ORB-SLAM2 [161]).

There are many types of detectors and descriptors, and in general, an algorithm can perform both tasks. These algorithms range from corners detectors, such as Harris Corner Detectors or FAST corners, to techniques that calculate metrics from surrounding pixels, such as SURF, SIFT, BRISK, BRIEF, FREAK and ORB [159]. Mukherjee et al. [159] present a thorough experimental review of image detectors and descriptors, including summaries of each of the different algorithms and comparative assessment of performance. They conclude that SIFT feature descriptors with a FAST detector is a strong combination, as is the use of ORB as a detector and descriptor. The most appropriate combination, though, can depend on the desired application, which affects what the visual nature of the environment is, what the likely distortions are, what sensors are being used and the goal for using the features.

Whichever detector/descriptor combination is used, indirect VSLAM consists of extracting a set of features from each image and over multiple observations the 3D location of those features are determined. The 3D location can be determined with stereo cameras, RGBD cameras or with the movement of a monocular camera. As the camera moves through an environment, the observed features
and their 3D location are stored and updated to build the map representation. For localisation, the features that are observed are matched to the set of features in the map to inform the estimate of the robot pose. These observations also give information on the estimate of the feature locations. Herein lies a challenge of SLAM: the joint estimation of the robot state and the map state (the positions of the features in the map). The method to perform the joint estimation is a key factor of different algorithms and influences the map representation used. One approach is to use a Kalman filter. The resulting map is the set of features in 3D space. This state vector can become very large as the area of operation grows, making the Kalman filter less feasible.

A more efficient map representation has emerged with recent work, to represent the map with a pose-graph $[19,55,119,163,183]$. This map representation is used in the estimation by performing a variant of pose-graph optimisation: adjusting the poses in the keyframes to minimise the geometric error of features observed in different keyframes. This allows the history of observations and estimations to be considered in an update, rather than just the current state and covariance, as done in a filterbased algorithm. Considering the whole history of observations allows for loop closure operations to be performed in a global update. The estimation can use the complete graph [104, 163], or a non-linear optimisation could be performed on a sliding window of keyframes: considering only a reduced subset of recent keyframes [119, 183].

Pose-graph-based SLAM algorithms tend to have separation of tracking the robot pose and updating the map. In Parallel Tracking and Mapping (PTAM) [104] and ORB-SLAM2 [161, 163], for instance, the robot pose is tracked frame to frame by matching features observed to existing features in the map. Only when a new keyframe is inserted does the map get updated and only on loop closure is the full graph optimised.

Both the feature map and pose-graph map are very effective for localisation, but they do not provide a 3 D representation of the environment that is useful for visualisation or representation of obstacles. An example of a map produced by indirect algorithms is shown in Fig. 2.5.a.

Indirect VSLAM algorithms show consistently strong performance, with stereo cameras, RGBD cameras and with monocular cameras. PTAM [104] was the state-of-the-art until recently. The algorithm uses FAST features in a pose-graph. ORB-SLAM2 could be considered as the current state-of-the-art (at the time of publishing) and has been frequently utilised, including for aerial robots [4]. ORB-SLAM2 uses ORB feature detectors and descriptors in a pose-graph with loop closure capability (see Fig. 2.2 for a loop closure example). ORB-SLAM2 can operate through large environments using either monocular, stereo or RGBD cameras and does not require an IMU. OKVIS [119] and VINSMono [183] do integrate an IMU into their algorithm, use monocular cameras and employ a pose-graph that is optimised with a sliding window approach (fixed-lag smoothing). A recently developed algorithm that uses close integration with an IMU is the Stereo Multi-State Constraint Kalman Filter (S-MSCKF) [213], that fuses observations of FAST features into the measurement model of a Kalman Filter. Their approach has shown impressive performance running onboard a quadrotor, with high-speed flight and transition from indoor to outdoor lighting conditions. S-MSCKF can be regarded as one of the leading VIO algorithms to date. Other than ORB-SLAM2, all of these algorithms can be regarded as VO algorithms: they do not perform loop closure.


Figure 2.5. Example maps produced by indirect and semi-direct SLAM algorithms. (a) Map produced from the indirect algorithm, ORB-SLAM2 [163]. The robot pose is in blue, and the features are the black and red points. (b) Map produced by Semi-Direct Visual Odometry (SVO) [68], with ground truth in red, tracked trajectory in blue and features as green points. Both maps consist of the robot poses and a large set of point features in 3 D space: a representation that is not useful for trajectory planning. Images are from the respective references.

Indirect approaches tend to perform well on a wide range of scenarios, with strong data association between features and robustness to photometric errors, but the algorithms can perform poorly in lowdetail environments, or environments with high-frequency texture. In these cases, it becomes difficult to extract and match distinct features when much of the image looks similar. Direct methods, using more of the pixels in an image, can perform better in these challenging environments.

### 2.1.5 Direct Visual SLAM

Direct VSLAM algorithms use image pixels and their intensities as the features to match frame to frame. When using pixels as features, it is the photometric error that is used in the estimation: the goal is to minimise the differences in pixel intensities between an image taken by the robot and the predicted image based on a model of the environment. This model is created by projecting the observed pixels into 3D space to produce a point cloud, where each point has an associated intensity. When predicting the image the camera will see, a simulated image of the model is generated, as would be seen from estimated camera pose. The 3D point cloud model is the map: and is often referred to as a dense 3D reconstruction. To start the SLAM process, there needs to be an initial model, from which to produce a predicted image. The initial model can simply be a random depth field [57], a fixed depth with large uncertainties [19] or the initial pose could be tracked by an indirect method to then build the starting direct map [55].

Direct SLAM is a less mature field than indirect SLAM [56], but it is a field that has recently grown in capability due to enhanced computational speeds and the greater availability of cameras suited for computer vision applications. Direct methods are particularly sensitive to photometric errors, such as image distortions and rolling shutters; hence they require modern cameras that are designed to
minimise such errors.
The benefit of direct algorithms, compared to indirect algorithms, is that more of the information in an image is used, with can lead to more robust data association and the ability to localise in environments with low or high-frequency texture. By using the image intensities directly, the computation time in extracting features is removed, but that computational load is replaced by the need to generate predicted images from a 3D model. Additionally, there are more pixels to consider. As a result, direct algorithms are generally slower than indirect algorithms. Direct algorithms can readily be parallelised though, enabling quicker computation if a GPU can be used [56].

One of the large computational burdens for direct SLAM algorithms is in updating and maintaining a dense reconstruction of the environment. To address this issue, the map is often represented in a pose-graph, as with indirect methods, where a set of keyframes are stored, along with their point cloud observations. To further improve the computational efficiency, approaches are taken to reduce the number of pixels considered to only the most useful: the pixels with high gradients. These algorithms are called semi-dense direct SLAM. The map produced by semi-dense algorithms is also semi-dense, providing a more compact representation. Sparse maps can also be used, by using patches of pixels around image features (extracted with similar methods to indirect SLAM) and running an algorithm to minimise the photometric error between these patches.

The semi-dense map representation can be visually impressive (see Fig. 2.6) and can enable strong performance for localisation $[19,41,55,56,166]$, but the map is not suitable for trajectory planning, being a highly detailed representation that can be slow to query and can have many gaps between points. The representation is as a large set of points, rather than volumetric entities such as voxels, which is the type of representation needed for trajectory planning to adequately identify collisions.


Figure 2.6. Example maps produced by semi-dense and sparse visual SLAM algorithms. (a) Map from a semi-dense algorithm, LSD-SLAM [57], with camera pose in blue, connected by green lines. (b) Map from a sparse algorithm, DSO [55] with poses in black and trajectory in red. Images are from the respective publications.

Direct SLAM estimation methods are similar to that for indirect SLAM, utilising the data structure of a pose-graph, with full pose-graph optimisation [57], or sliding window optimisation [55]. Another estimation approach particular to direct algorithms is an energy minimisation formulation [41, 166]. All
these methods minimise the photometric errors between observations and the predicted observations from the map.

The leading direct algorithms, to date, operate with a monocular camera and generate the depth from the motion of the camera. LSD-SLAM [56] is one such leading algorithm that performs full graph optimisation with a semi-dense representation and is capable of performing loop closures. LSD-SLAM can also operate with stereo cameras [57]. An example map produced by LSD-SLAM is shown in Fig. 2.6.a. DTAM [166] is an earlier dense method that uses energy minimisation for the estimation. DPP-TAM [41] is a semi-dense method that also uses energy minimisation and exploits planar structures in the environment to provide a more complete map. Both DTAM and DPP-TAM are designed with a goal for 3D dense reconstruction, rather than robotic navigation. This is in contrast to Direct Sparse Odometry (DSO) [55], a sparse method using sliding window optimisation, that is tailored towards providing VO information for robotics. An example map from DSO is shown in Fig. 2.6.b. The strong performance of each of the algorithms in a variety of environments (see Table 2.4) shows the effectiveness of direct methods.

All of the direct algorithms described above operate without an IMU, which highlights the difficulty in fusing an IMU into direct VSLAM. The challenge is due to the complex mapping from the space of physical camera movement to the image space, which makes it difficult to formulate in an estimation algorithm. Nonetheless, Robust Visual Inertial Odometry (ROVIO) [19], is a sparse direct SLAM algorithm that uses BRISK and FREAK detectors to extract image features and minimises the photometric error of patches around these features in the update step of an EKF. The EKF enables the integration with the IMU, which allows ROVIO to operate with a constant depth initial model, making it a useful algorithm for mobile robots.

### 2.1.6 Semi-Direct Visual SLAM

Semi-direct VSLAM algorithms aim to take the strengths from both classes of SLAM by using the indirect elements of: sparse features, low computation time, minimisation of geometric errors in a joint estimation; combined with the direct SLAM characteristic of robust data association using pixel intensities. A summary of the differences between indirect and direct VSLAM algorithms is presented in Table 2.1. Semi-direct approaches first extract visual features, using feature detectors similarly to indirect algorithms but then use a patch of pixels around the feature for data association, with the aim to minimise photometric error as with direct algorithms. Once matched, though, the location of the features are used in a joint estimation of the map and robot state to minimise geometric error, by using similar methods to indirect VSLAM. The photometric error is minimised in matching features, and geometric error is minimised in the estimation of robot and map state.

The map representation for semi-direct approaches is similar to both direct and indirect methods: pose-graphs with observed features but with the features being image patches and their centres. As with direct and indirect methods, the map enables effective localisation but is not a representation of the physical obstructions in the environment (see Fig. 2.5.b). Estimation methods likewise take similar forms, with Kalman filter approaches and pose-graph optimisations. Semi-direct methods have shown strong performance and are becoming more and more popular as a VSLAM algorithm of choice, particularly for robotic navigation in 3D.

TABLE 2.1. Comparison of key factors for indirect and direct visual SLAM.

| Comparison Factor | Indirect | Direct |
| :--- | :--- | :--- |
| Information used | Less: only features | More: sets of pixels |
| Initialisation | Not needed | Needs good initialisation |
| Computation time | Faster. | Slower, but can be parallelised |
| Data association | Critical component of features | Inherent in matching pixels |
| Performance in high <br> frequency or sparse texture | Sensitive | Robust |
| Performance with photometric <br> noise | Robust | Sensitive |
| Ability to fuse IMU <br> information | Simple | Difficult |

Semi-direct Visual Odometry (SVO) [67, 68] is a semi-direct algorithm that has seen wide, successful use, in particular for flying robots, with the use of monocular cameras [61, 68, 147]. SVO uses FAST corners and image edgelets as the feature detectors and performs a Bayesian depth estimation to resolve the 3D locations of features from motion. As a visual odometry algorithm, SVO operates without an IMU and is best for tracking movement in a relatively small environment. SVO can be combined with a back end EKF or pose-graph SLAM framework to enable operation in large environments, fusing IMU information and performing loop closures [48].

Work by Ait-Jellal at al. [4] has recently extended ORB-SLAM2 to be a semi-direct method by extracting image patches around ORB features and using the pixels in those patches to match features. These modifications are shown to be very effective for SLAM with quadrotors.

### 2.1.7 Appearance-Based Visual SLAM

A separate class of VSLAM algorithms utilises appearance descriptors to give identification of locations throughout an environment. The goal is for large scale, persistent navigation, to emulate how animals, such as rats, explore environments. In RatSLAM [143], a set of experiences are stored that consist of a robot pose and a view of the environment. A set of experiences are linked together with odometry information on the motion between them to form a semi-metric, topological, experience map. The features in RatSLAM are the appearance descriptors for a given view of the environment. When an appearance is observed, it can be matched back to an experience stored in the map to give information on the location of the robot. RatSLAM has been demonstrated with online, real-time SLAM for long term navigation of ground-based robots. The algorithm can operate in dynamic and unstructured environments and is robust to being moved from a known to an unknown location. The algorithm is not designed, though, to track fast dynamics or provide visual odometry information that could be used by a flying robot. The map is purely for long term navigation and gives no information on the physical obstructions in the environment (see Fig. 2.7).


Figure 2.7. Example map from RatSLAM: an experience map. Nodes indicate particular experiences, with robot pose and view appearance information. Nodes are connected by odometry information. (a) Map over a large environment. (b) Close-up of map section highlighted in the grey box in (a). Image from [143].

### 2.1.8 Geometric-Feature-Based Visual SLAM

The concept of using features of an image for SLAM, as in direct methods, can be extended further to utilise the structure of the environment and extract extended geometric features. Sola et al. [209], for example, presents an analysis of EKF-SLAM with line features, that could be computed to fit distinct edges in the environment. Alternatively, features can take the form of splines, giving the advantage of fitting to non-straight edges. For aerial navigation purposes, when looking down on the environment, the splines can be fit to features such as roads, the edges of bodies of water or a combination of such features, as in Terrain Aided SLAM (TASLAM) [42, 221, 225]. On the ground, the spline could be fitted to the edges of paths, or roads, such as in Curve-SLAM [137, 190]. These algorithms involve fitting a spline to observed features and matching the shape of the spline to a previously observed spline. The shape of the spline provides a more richly detailed feature than 3D points, which can provide more information to inform the estimate of the robot pose.

The map that is produced is a set of spline features that are primarily for navigation. The map can also provide information on the structure of the environment, such as the boundaries of a path, that could be used for path planning. For example, maps using spline features, see Fig. 2.8.

CurveSLAM has only been demonstrated for 2D navigation of ground-based robots, and the aerial algorithms are best suited to Terrain Aided Navigation (TAN), where a map of the spline features is available before flight. Nonetheless, the concepts used in these algorithms may apply to 3D contexts.

### 2.1.9 Summary and Assessment

The VSLAM algorithms discussed here are summarised in Table 2.3, where the features, the map representation, the sensors and the estimation approach are highlighted. Table 2.4 presents the environments in which the algorithms have been successfully demonstrated. The focus of this thesis is the application of SLAM algorithms to small flying aerial or space-based robots, with power and weight constraints as well as limited computational capacity. Out of those reviewed, there are a number of leading algorithms that are suitable for small flying robots. ORB-SLAM2 [163] stands out as state-of-the-art for a SLAM algorithm without an IMU, with S-MSCKF [213] being the leading approach with IMU integration. Out of the direct VSLAM methods, DSO [55] shows strong performance but only as


Figure 2.8. Example maps from geometric-feature-based SLAM, overlayed on Google Earth imagery. (a) Map from CurveSLAM [137], with spline features in blue with the endpoints of the splines in red. The splines follow the structure of the path. (b) Map from TASLAM [225], with spline features in red matching the edge of a body of water. The SLAM trajectory is in blue and the GPS tracked trajectory is in black. Images are from the respective publications.
an odometry algorithm. Recent developments with semi-direct algorithms appear to give competitive performance, with both SVO (with a SLAM back-end) [48] and a modified ORB-SLAM2 [4] providing robust VSLAM that can readily be applied to small flying robots.

While there are a range of suitable algorithms for tracking the location of a flying robot, what can be seen in Table 2.3 and Figs. 2.5, 2.6, 2.7 and 2.8 is that none of these algorithms generate a map that is a physical representation of the environment: a representation that can be used for trajectory planning. Some of the dense, direct methods do produce a dense point cloud, but as noted above, the point cloud is not a volumetric representation of the obstructions; hence another algorithm is needed to produce such a representation from the point cloud. The geometric-feature-based SLAM algorithms can provide a representation of some physical restrictions, but these are not a representation of obstacles that need to be avoided. The point cloud SLAM methods do utilise a map that represents physical obstructions, but these algorithms either require a VIO algorithm and use a pre-generated map [63, 177], or require lidar as a sensor [52]. None of the point cloud algorithms, in isolation, enable SLAM in an unknown environment with sensors suitable for a small flying robot.

Because the maps generated by the leading SLAM algorithms for small flying robots do not produce maps of the physical obstacles in the environment, the state-of-the-art for having the full navigation stack of localisation, mapping and trajectory planning, is to have a completely separate mapping algorithm to generate a map of the physical environment. These mapping algorithms will be explored in the next section.

What is proposed in this work is to develop one algorithm to perform SLAM with RGBD sensors, where the map that is produced is immediately a physical representation of the environment that can be directly used in trajectory planning. The challenge in this approach is in determining what features to use. The features need to satisfy the requirements for SLAM: robust extraction, robust data association and integration into an estimation approach, as well as the requirements for trajectory planning: an
appropriate volumetric representation of obstructions. The next section discusses 3D mapping methods and the different types of possible 3D representations that could be suitable for both SLAM and obstacle representation.

Table 2.2. Datasets for SLAM testing

| $\#$ | Dataset | What | Sensors | Link |
| :--- | :--- | :--- | :--- | :--- |
| 1 | EuRoC [28] | UAV datasets. Indoors, industrial. | Stereo, IMU | EuRoC |
| 2 | Kitti [76] | Driving. City, rural, highway. | Stereo, lidar | Kitti |
| 3 | ICL-NUIM [86] | Simulated hand carried camera. Indoors. | RGBD | ICL-NUIM |
| 4 | TUM RGBD [212] | Hand carried and ground robot. Indoors. | RGBD | RGBD |
| 5 | TUM MonoVO [58] | Hand carried. Indoors, outdoor park | Mono | MonoVO |
| 6 | NewCollege [208] | Ground robot. Outdoor campus. | Stereo, lidar | NewCollege |
| 7 | FastFlight [213] | High speed quadrotor flight on runway. | Stereo, IMU | FastFlight |

TABLE 2.3. Summary of SLAM algorithms


[^4]TABLE 2.4. Demonstrations and tests of SLAM algorithms

*This column contains references where additional tests on the algorithm have been performed. 1 s indicate the algorithm has been demonstrated in that environment, dataset or system. 0s indicate the algorithm has not been demonstrated, to the author's knowledge Environments are A: Desktop, B: Office, C: Industrial, D: Outdoors around buildings, E: Outdoors around nature, F: UAV carried camera, G: UAV high speed flight, H: Highway on car, I Suburbs on a car, J: Underwater, K: Around a comet.

## Datasets are detailed in Table 2.2

UAV Hardware are cases where the algorithm has been demonstrated in flight onboard a UAV system doing online, onboard SLAM.

### 2.2 3D Mapping

As outlined in the SLAM section, the current state-of-the-art for the autonomous navigation stack on small mobile robots is to have a separate 3D mapping algorithm to the SLAM algorithm. The 3D mapping algorithms assume an accurate pose estimate is given from SLAM to then project observations into a 3D map of the world. The SLAM maps are designed to aid localisation, whereas separate 3D mapping algorithms can be designed for producing a detailed map of the environment to visualise, or for representing obstacles in the environment for trajectory planning and obstacle avoidance. It is the latter application that is of most interest here. Additionally, the goal in this work is to have a map representation that is applicable both as an obstacle representation and for SLAM.

Existing approaches to 3D mapping will first be reviewed in this section, highlighting the current capabilities. The review is intended to reveal the gaps in being able to use low power sensors to generate a 3D map, online, that is useful for both localisation and trajectory planning. To fill the gap, it is identified that a potential approach is modelling 3D objects as features for SLAM and obstacle representations. Modelling objects in the environment also has the benefit of producing a representation that can be used for grasping, object classification, and dynamic obstacles. Therefore a review of 3D object-modelling algorithms is presented to explore different possibilities to achieve both localisation and trajectory planning goals.

### 2.2.1 3D Mapping Algorithms

There are many successful demonstrations of 3D mapping, the leading examples of which are reviewed here. Some of the factors that differentiate 3D mapping algorithms are: the map representation, how the map is built, if the map can be built online, the required sensors and the intended purpose of the map. The ideal 3D mapping algorithm for the goals in this work would be able to generate a compact 3 D representation that has volumetric obstacle information for trajectory planning, using sensors that could be carried on a small mobile robot. This representation would ideally be high in detail, to be useful for a localisation task and would be able to be produced incrementally with computations onboard the robot.

### 2.2.1.1 Point Clouds

A point cloud map representation is a collection of 3 D points representing the physical objects in the environment. These maps are often referred to as dense 3D reconstructions. The point cloud is built by projecting observed 3D points into a global frame of reference, given the pose of the sensor when the observation was made. With accurate pose estimates and direct 3D measurements from a laser scanner, this method of mapping is prevalent for generating detailed maps of environments (e.g. [103]). If lidar is not possible, such as on small flying robots, then 3D measurements from stereo or RGBD can be used in a similar fashion. For example, the depth measurements from stereo or RGBD in ORB-SLAM2 can be stored for each keyframe and then be projected from the keyframes into a common map, giving a result such as Fig. 2.9.

The projection of 3D observations from keyframes, such as with ORB-SLAM2, requires direct 3 D observations in each frame, hence is not applicable for using monocular cameras. However, the


FIGURE 2.9. Dense 3D reconstruction from ORB-SLAM2. The coloured point cloud is constructed by projecting 3D observations from each keyframe (blue icons) in the ORBSLAM2 map. The 3D observations come directly from stereo or RGBD cameras. Image from [163].
depth of pixels with monocular cameras can be estimated by using multiple observations, such as with direct VSLAM techniques, where the map generated is a semi-dense or sparse 3D reconstruction (e.g. Fig. 2.6). There are also specific tools for dense 3D reconstruction for monocular images. These tools take a sequence of images from localised poses and estimate the depths of the observed pixels to produce the point cloud. Maplab [206] is one open-source tool that uses a technique called Multi-View Stereo (MVS) [73] to combine sets of monocular images into virtual stereo pairs to estimate the depth. The depth generation from MVS is similar in concept to monocular direct SLAM but with the goal for generating point clouds. Another open source tool, REMODE [182], uses a probabilistic depth estimation algorithm that determines the depth for each pixel separately. Both Maplab and REMODE require aligned sets of poses to perform the mapping, hence both come integrated with a VSLAM algorithm: ROVIO and SVO respectively. The algorithms are generally designed for off-line processing with a large set of images, but REMODE can run at speeds that may be suitable for online mapping, with parallelisation on a GPU [182]. The two tools can also do further processing of the map to fill holes and generate complete meshes for visualisation. Example results can be seen in Fig 2.10. One important consideration for monocular mapping is that there will be a scale ambiguity if there are no other measurements, such as from an IMU.

Point cloud maps can be used for localisation by matching an observed set of 3D points to the point cloud map. Algorithms such as ICP can be used for this step, as is done for point cloud SLAM (Section 2.1.2). Alternatively, pixel intensities could be matched, as in direct VSLAM (Section 2.1.5).

The point cloud representations are visually impressive (see Fig. 2.10), can produce high detail for mapping and can be used for localisation, but they are are not a suitable representation for trajectory planning. The point cloud only gives information on obstacles at each individual point and provides no information on the obstructions between points: i.e. there is no volumetric representation of obstructions, which is what is needed for trajectory planning. Additionally, point cloud maps are difficult to quickly query to determine if a point on a trajectory is near an obstacle. Finally, point clouds are not a compact representation, making storage of the map a potential issue on resource constrained systems. Therefore


Figure 2.10. Example points cloud dense 3D reconstructions. (a) From Maplab [206]. (b) From REMODE with filtering to fill gaps [182]. Images are from the respective publications.
to generate a map that is suitable for trajectory planning, a volumetric representation is needed.

### 2.2.1.2 Occupancy Maps

The most commonly used volumetric representation of an environment is an occupancy grid, where 3D space is divided up into cubic cells that are either occupied, free or unknown. Observations of 3D points are projected into 3D space, as with point cloud methods, but instead of being grouped together to form a conglomerate point cloud, the observations are used to update the probability of occupancy for the cells in the grid. The occupancy probability is updated both at the location of a 3D point and at all the cells on the ray from the 3D point to the sensor, where the observation is that these cells are free. These updates of occupancy probability are combined from many observations to build up the occupancy map.

For the purposes of trajectory planning, thresholds on the occupancy probability can be used to categorise cells into free, occupied and unknown. With this categorisation, the feasibility of a point on a trajectory can be determined by checking the occupancy of the cell within which it lies. Another benefit of an occupancy grid over point clouds is that the representation combines information from multiple observations by updating the probability of occupancy. This combination of observations allows noisy measurements to be averaged out and can allow the map to update if there are changes in the environment.

Occupancy maps have been shown to be effective for trajectory planning, with online updates and relatively inexpensive computations. They have been widely used in 2D applications (such as [203]) as well as in 3D applications [4, 52, 63, 89, 147, 177]. One commonly used algorithm for generating and storing occupancy maps is OctoMap [91]. OctoMap represents an occupancy grid with an octree: a tree data structure where each node has eight children. For OctoMap, nodes are 3D cells and the children represent the spatial division of that cell into eight sub-cells. The parent cell has an occupancy probability based on the occupancy probabilities of its children (see [91] for details). What this data structure allows is a variable resolution representation by using different levels of the octree: the highest level gives a coarse representation and the lowest level gives a detailed representation. The choice of the number of levels used affects the range of resolutions that can be represented. Having the ability to adjust the resolution can be very useful for robotic navigation: to use a low resolution for
quick collision checking in trajectory planning, or to have low resolutions far from the robot and higher resolutions close to the robot. The octree data structure also allows efficient storage and accessing of cells at a given 3D location by progressing down the octree, rather than searching the whole 3D space.

Occupancy maps require depth observations and are commonly used with laser scanners, RGBD cameras and stereo cameras. The maps can be used for different purposes, depending on the application. Heng et al [89], for instance, uses stereo cameras on a quadrotor to create a robot-centric, local OctoMap, to provide information for obstacle avoidance and trajectory planning. Ait-Jellal et al. [4] also use stereo cameras to build an OctoMap for trajectory planning but build a global map, rather than a local map. The state-of-the-art for autonomous navigation tend to use two occupancy grids: a global occupancy grid that is pre-mapped and a local occupancy map that is centred around the robot and is generated with observations from the robot. The global map is used for planning the long term path of the robot and the local map is used, in addition to the global map, for short term trajectory planning and obstacle avoidance. The local map uses the most recent observations, allowing the robot to react to changes in the environment, dynamic obstacles, or errors in the estimate of the robot pose relative to the global map. Fang et al. [63], for instance, use a pre-mapped 2D occupancy grid of a naval ship for quadrotor path planning and combine this with a 3D local occupancy map for obstacle avoidance. Mohta et al. [147] similarly uses a 2D global map with a 3D local map for autonomous navigation of a quadrotor (see Fig. 2.11), but they generate the 2D map online by updating it with slices from the 3D local map. A combination of 3D global and 3D local maps is also possible, as used by Perez-Grau et al. [177] with a pre-mapped 3D OctoMap that is fused with a local OctoMap for quadrotor trajectory planning.


Figure 2.11. Example combination of global and local occupancy maps from Mohta et al. [147]. (a) Local point cloud observations. (b) Global 2D map in grey and black, with local map bound in the yellow box. (c) A trajectory planned in a combination of the lcocal and global map. Images from [147].

The occupancy mapping algorithms presented above are very effective for representing the obstructions in the environment for trajectory planning. Occupancy grids can also be used for localisation, as described in Section 2.1.2, by matching a set of 3D observations to occupied cells to inform the state of the robot. One example of this is the approach from Gil et al. [79] that uses a hybrid combination
between an occupancy grid for mapping and localisation. The lidar-based technique builds a local occupancy grid and uses scan matching for short term odometry. In addition to the scan matching, a distance transform to the 2D occupancy grid is then used to extract visual features (SURF) for matching adjacent sub-maps in a pose-graph, and for loop closures for global SLAM. Their approach requires lidar, however, and is restricted to 2D applications. Hence, occupancy mapping algorithms can achieve the desired dual use of SLAM and obstacle representation but to date, only in 2D with lidar.

Occupancy maps have been shown to work purely for localisation in 3D pre-mapped environments with RGBD cameras [63, 177]. Using occupancy maps for online SLAM in 3D is more difficult, as the resolution of an occupancy map that can be produced online tends to be low, providing limited detail for localisation. Higher detail grids can be used, approaching the level of detail that has been shown to be effective with point clouds but this then increases the computational load.

### 2.2.1.3 Surfel Grids

One occupancy mapping variant that has been successfully used for 3D SLAM is Surfel Grids [52]. This representation is a slight modification to occupancy maps where each cell contains a surfel: a mean and covariance of the points contained within that cell (for example, see Fig. 2.12.a). While the representation does not provide a more detailed 3D representation, it captures more information on the observations in each surfel, which can aid in localisation. The SLAM approach using surfel grids is described in Section 2.1.2, where surfels are matched for local tracking and surfel maps are fused for global SLAM in a pose-graph. Surfel to surfel matching, using the mean and covariance information, rather than just matching a point to an occupied cell, means that the localisation can be performed without having high resolution. The surfel map used is also multi-resolution, using high detail near the robot and low detail further from it, to match with the concentration and accuracy of observations. For trajectory planning, the surfel grids can be used similarly to occupancy maps.

The surfel grid algorithm has been successfully demonstrated for online mapping, localisation and trajectory planning for a quadrotor by Droeschel et al. [52]. The demonstration used a suite of sensors including stereo cameras, ultrasonic sensors and a lidar on a gimbal. The lidar provides the majority of the 3D observations needed for the surfel grid mapping with a wide field for view. It is not clear if this approach would be equally effective without the use of a lidar, where point cloud observations are less accurate and have a lesser field of view.

### 2.2.1.4 Signed Distance Fields

Another variation on a grid-based representation is to store a signed distance in each grid point. A signed distance is the distance to the nearest surface, with the sign being negative if the grid point is inside an object. Such a representation is called a Signed Distance Field (SDF) and is ideal for trajectory planning as it effectively provides a potential field to inform trajectory planning algorithms on how to move the trajectory to avoid obstacles.

To generate and update SDFs online, a representation called a Truncated Signed Distance Field (TSDF) is used, where the signed distance values are only defined for a small distance from the surface (the truncated region). The TSDF representation allows more efficient updates because any


Figure 2.12. Example map representations. (a) Surfel map from [52]. Ellipsoids represent mean and covariance of points in a 3D grid and are coloured by orientation. (b) ESDF (coloured cells) and mesh from TSDF with Voxblox [171]. One slice of the ESDF is shown, with purple colours being closer to obstacles and green further from obstacles. The grey and white objects are the surfaces extracted from the TSDF. A trajectory is planned through the ESDF and is depicted in yellow. Images are from the respective references.
change to the observation of a surface only requires the update of a small number of points within the neighbourhood of the observation, rather than having to propagate out changes to the SDF throughout an entire volume. New scans can be quickly integrated into the TSDF because they are able to use the projective distance: the distance along the ray from the camera to a 3D point, rather than the absolute Euclidean distance, which would require extra computations.

TSDFs are able to be generated and updated online and can produce strong visualisations by taking the level set of a TSDF through zero distance to create a continuous surface mesh. However, by truncating the region in which the distances are defined, TSDFs are no longer as useful as an obstacle representation for trajectory planning. A Euclidean Signed Distance Field (ESDF) represents the signed distances throughout a 3D volume and uses the Euclidean distance to the nearest surface. Such a representation is what is desired for trajectory planning. An ESDF cannot be built efficiently online, though, because the update of one observation can potentially require updates throughout a large portion of the map. A solution is to efficiently generate an approximate ESDF from a TSDF, as performed in the open source tool: Voxblox [171]. By generating a TSDF map online and using the TSDF map to generate the ESDF map, Voxblox is able to produce, online, a suitable 3D representation for trajectory planning. For efficient mapping and planning in large environments, Voxblox uses a spatial hashing approach [167] to store the map in different blocks of volume and only access those blocks when the robot is within the corresponding volume. See Fig. 2.12.b for an ESDF from a TSDF produced by Voxblox. The capability of Voxblox has been demonstrated with online mapping and planning on a quadrotor with stereo cameras for depth measurements [170, 172].

To enable the online conversion from a TSDF to an ESDF, there is a trade-off in accuracy, with the potential for small errors in the ESDF and limitations on the resolution possible. The TSDF or ESDF could potentially be used for localisation by matching 3D observations to grid values of zero distance,
but it is unclear whether the level of detail possible would make such an approach effective.

### 2.2.1.5 Continuous Occupancy Maps

In grid-based map representations, there is some information lost in discretising the space. For instance, in occupancy grids, each cell is an independent measure of probability; there is no correlation to neighbouring cells, or use of the inherent structure in the environment to inform the map generation. SDFs do propagate information to nearby cells to have linked structures, but they are not a probabilistic representation of the correlations between points. Continuous occupancy methods, in contrast, use the inherent structure of the environment and the correlations between the occupancy of nearby cells, with a probability of occupancy that expands throughout the whole space. What this continuous occupancy representation provides is:

- Contextual information with a correlation between points to better represent the environment.
- Modelling of occupancy between observations to:
- Make inferences on unobserved space.
- Enable sampling of the occupancy at any resolution.
- Variance information to provide uncertainty measures for trajectory planning and exploration.

To use continuous occupancy maps for trajectory planning, the final representation is still an occupancy grid, but this grid is generated from an underlying continuous representation.

The continuous representation is generated from a statistical regression on observed 3D points and query 3 D points (referred to as training points and test points, respectively, in the literature). The query points are where it is desired to determine the value of the representation, and in the case of occupancy mapping, the query points are the occupancy grid points. The query points can then be classified into occupied, free, or unclear, by applying thresholds to the occupancy probability.

## Gaussian Process Occupancy Mapping (GPOM)

One approach to continuous occupancy mapping is Gaussian Process Occupancy Mapping (GPOM) [169], where the continuous function is generated from a Gaussian Process (GP). A GP is a method to fit a parameter-free function to data, where this fit is informed by a prior covariance between all observation points and query points. The prior covariance is defined by a kernel function, the nature of which affects the functional fits that are generated. The kernel functions can be optimised by training hyperparameters [169]. A regression is performed to find the mean and variance of the functional value at the query points, using the prior covariance and observations points. For GPOM, the functional value is the occupancy probability. The functional fit does not give an explicit equation but instead is represented by a combination of the prior covariance and the observation points. The functional fit can only be evaluated by performing the regression on a query point. Any scale of query can be performed, though, and in any part of the environment by selecting the query points accordingly.

A challenge for GPOM is the large computational expense in performing the regression, that scales at $\mathscr{O}\left(n^{3}\right)$, where $n$ is the number of test points. Without any modifications to GPOM, this computational load limits the application to small 2D environments and offline demonstrations. Therefore, developments have been made to GPOM to tackle the computational complexity with a variety of techniques to
break-up the problem. These techniques mainly using the observation that distant parts of the map have minimal correlations and hence can be treated separately. One technique uses localised regional regressions, where a small set of observation points around a query point are used, with each query point running a regression independently [169]. It is more efficient to perform the regression on a batch of query points, through, hence the regional regression concept is expanded to process all query points contained in a 3D cell. The batch regression with these test points uses the observation points in adjacent 3D cells [223]. Another, similar, technique to break up the computations it to perform regressions on different regions of the map and then merge the results together with a Bayesian Committee Machines (BCM) [223]. The test points can also be organised in an octree to allow test points to be pruned (moving to a lower resolution level of the octree) and for quick access of data. The combination of regional regressions, BCMs and test data octrees can enable the use of GPOM in 3D environments with a computation time that is suitable for online operation [223].

## Hilbert Maps

Hilbert maps [187] are a similar concept to GPOM, using kernels and regressions to extract the mean and variance at test points from an underlying continuous representation. They are designed to address some of the computational challenges of GPOM and the scaling issues [187]. The formulation of Hilbert maps allows for linear time update of the map and the use of stochastic gradient descent (SGD) to enable quick regressions. Hilbert maps are an order of magnitude quicker to update than GPOMs if there are no modifications to either algorithm to break up the problem [187]. Similarly to GPOM, techniques can be applied to Hilbert maps to split a problem into smaller Hilbert maps which are then fused together, resulting in a speed increase and the ability to apply Hilbert maps to 3D problems [50, 83].The current, state-of-the-art for GPOM and Hilbert maps shows that for 3D environments, GPOM is quicker [50, 223].

Continuous occupancy mapping algorithms have shown the ability to produce more accurate maps than OctoMap at a similar computational load [223] and have the advantages of modelling unobserved space as well as providing variance information. However, all demonstrations have been with groundbased robots using laser scanners and it is unclear how effective the algorithms would be with noisier sensors. The probabilistic framework lends itself well to including sensor noise and the algorithms have been successfully demonstrated with simulated and real noise on laser scans. Nonetheless, the tolerance of continuous occupancy mapping algorithms to the increased noise from stereo or RGBD is unclear.

### 2.2.1.6 Confidence Rich Maps

A recent method that has been demonstrated with stereo cameras and considers correlations between cells is Confidence Rich Mapping (CRM) [3], which is a variant on occupancy maps. The assumption of cell independence in occupancy maps is relaxed in CRM by storing both a probability of occupancy and a covariance in each cell. When a new observation is made, all of the cells in the view cone are correlated. The view cone is the volume of space from which the camera observations come. Every cell in this cone has been observed as either free or occupied with the same sensor scan, with common noise and common uncertainty in camera pose. Therefore the occupancy values are correlated and this is
included in CRM to build up the covariance of occupancy in each cell. The result is that the mean and covariance in the cells are a representation of continuous occupancy.

CRM has been shown to run at speeds suitable for online operation with stereo cameras and produce more accurate maps than OctoMap and GPOM in 2D test cases [3]. The representation produced by CRM is of high utility to trajectory planning, in particular for risk-aware planning, where the uncertainty represented in the covariance is used for planning safe trajectories [88]. There have yet to be 3D demonstrations of CRM, but the algorithm shows potential for such applications.

The variance information stored in CRM cells could also aid localisation, by giving the measure of the uncertainty of an object at a given location. The map representation produced by CRM is an occupancy map, though, hence for localisation, the same challenges are present in being able to generate a sufficiently detailed map for localisation.

### 2.2.1.7 Summary and Assessment

There are numerous 3D mapping algorithms that can produce a map online that is useful for trajectory planning, with a volumetric representation. Some of the representations provide richer information for trajectory planning, with distance fields in ESDFs [171], or measures of uncertainty in GPOM [169], Hilbert maps [187] and CRM [3].

All of the techniques, other than surfel maps [52], require the pose of the robot to be known and rely on a separate localisation algorithm. The surfel map localisation uses a large set of sensors, including a lidar: a sensor that is difficult to carry for small flying robots.

Many of the map representations could potentially be used for localisation by matching newly observed 3D points to surfaces or occupied cells in the map, but the maps are generally low in detail to enable online mapping and the point-to-environment matching can be difficult without the accuracy of lidar. 3D modelling algorithms that could produce a more detailed representation to be useful for localisation, such as GPOM and Hilbert maps, have only been demonstrated in 3D with the use of lidar.

None of the algorithms presented give the full desired capability: to use light, low powered sensors to produce a map, online, that is suitable for trajectory planning and detailed enough to be used for localisation. Therefore, a review of 3D modelling algorithms is presented below to explore what other options there are for 3D mapping to achieve the desired capability.

### 2.2.2 3D Modelling

Attribution: Part of this section has previously been presented in [156]. All is the work of the author of this thesis.

There are many ways in which 3D objects and structures can be represented and each representation is produced with a different algorithm. The representation is referred to as the 3 D model and the method of producing the representation as 3D modelling. The 3D mapping algorithms described in the previous section use a range of 3 D models, from point clouds, to grid maps, signed distance fields and Gaussian Processes.

The focus in this review is on producing a 3D model that is of utility to localisation, as well as for an obstacle representation.

In particular, the focus here is on modelling 3 D objects, rather than a 3 D map of the entire environment. The motivations for such an approach are as follows:

- To split the mapping problem into sub-problems to simplify the problem, taking inspiration from GPOM and Hilbert Maps.
- To investigate an alternative approach to 3D mapping with occupancy grids, having identified the challenges with achieving localisation and mapping goals with existing 3D mapping algorithms.
- To have a representation that can include dynamic obstacles, by giving dynamics to a 3D object.
- To produce representations that have future utility for interaction tasks, such as grasping [22, 144].
- To produce representations that could be used for classification of objects to support decisionmaking autonomy.
Following this direction of investigation, algorithms for modelling 3D objects are reviewed here. For this discussion, it is assumed that segmentation of an observed point cloud into separate objects has been performed.
The goal for the 3D modelling algorithm is to be effective in the following tasks:

1. Generation and update of a 3D model from point cloud observations.
2. Localisation for matching point cloud observations to a previously mapped 3D object.
3. Computing the distance of a 3D point to the 3D object and the gradient of the distance, for use as an obstacle in trajectory planning.
4. Storage in a database of many 3D objects.

In performing these tasks, the algorithm should balance the requirements to:

- Generate an accurate representation of the true physical object.
- Provide useful and accurate information in localisation.
- Run quickly enough for real-time implementation.
- Have a compact description for storage in a database of 3D objects.

There are many algorithms for generating 3D models, as reviewed by Chang et al. [37]. A subset of these algorithms is reviewed here to assess the potential in achieving the criteria listed above.

### 2.2.2.1 Ellipsoids from Point Clouds

Ellipsoids are a convenient obstacle representation for trajectory optimisation, especially for the algorithm presented in this work [35, 155]. Therefore, there is a strong benefit is representing objects in the map as ellipsoids. Ellipsoids can be modelled from a point cloud observation of an object, by computing a centroid and performing a principal component analysis [97] to find a set of three orthogonal axes along the directions of most variance and the corresponding variance in those directions. These properties are used to define the primary axes and size of the ellipsoid (setting the size along each axis as $3 \sigma$ ). The ellipsoid is then described by its centroid, axes magnitudes and orientation (in quaternions).

This process is very quick and simple but is a strong approximation of the true shape (see Kuether et al. [109] for examples of this technique being applied).

### 2.2.2.2 Gaussian Process Implicit Surfaces (GPIS)

Gaussian Process Implicit Surfaces (GPIS) [51, 226] is a technique similar to GPOM that utilises GPs to model 3D structure [191] . GPIS is designed to model 3D objects, though, rather than an entire 3D environment. As with GPOM, the GP represents a fit of a function to the data and the querying of the function requires a regression process with observation points and query points. In contrast to GPOM, the functional value in GPIS is a non-dimensional signed distance: i.e. increasing positive numbers when moving away from a surface, zero at a surface and negative inside an object. The implicit surface in GPIS is the level set through the test points where the functional value is zero.

GPIS can produce very detailed models (see [226]) but with the use of many observation points and many query points, which lead to large computation times. As with GPOM, the representation of the functional fit is stored in the observation points and prior covariance, which needs to be evaluated by performing a regression at query points. A mesh could be fit to the implicit surface to give a more compact representation and an ellipsoid could be fit around such a mesh for a representation designed for trajectory planning but such representations lose the ability to maintain and update the underlying GP functional fit. The advantage of maintaining the GPIS representation is that it can easily be updated with new measurements and take into account the uncertainty in measurements. Maintaining the variance information is beneficial if GPIS was to be used for localisation. Achieving sufficient detail for localisation has to be balanced with computational cost, though, as the complexity scales with $O\left(n^{3}\right)$, as for GPOM. Even though the scale of modelling a single object is smaller than for GPOM, the desired accuracy can still lead to a large number of query points.

### 2.2.2.3 Non-Uniform Rational B-Splines (NURBS)

Non-Uniform Rational B-Splines (NURBS) are a common way to generate and represent 3D curves and surfaces, in particular in CAD programs [180]. The strength in NURBS comes in their ability to represent a large range of objects, with a relatively small set of control points, where the control points can be moved to have local control of the curve or surface [179]. The formulation can be used to fit surfaces to point clouds of data and has seen use in applications such as reverse engineering [127]. The textbook by Piegel et al. [181] is an excellent reference for the basics and a detailed overviews of NURBS.

The level of detail possible with NURBS comes with the same trade-off between computational speed and detail. At one end of the scale, highly detailed models for reverse engineering can be produced but at large computational expense. At low computation times, NURBS can still produce good, representative models but without small scale detail.

To include multiple observations, a NURBS surface can be efficiently updated by adding or modifying control points but without a probabilistic representation. Localisation could potentially be performed with NURBS by matching observations to the surface and using surface curvature information to help with data association. For obstacle representation, the distance from a point to the surface would need
to be computed, or alternatively an ellipsoid could be fit around the surface for a more efficient obstacle representation.

### 2.2.2.4 Sphere Meshes

Instead of a single ellipsoid, a large set of overlapping spheres can be used to describe a 3D object; a method called sphere meshes [215]. The spheres can vary in radius to both represent large volumes of an object and smaller details. The number of spheres can become large when there is a lot of detail to represent, hence to make the representation more efficient, the spheres can be combined connected with edges and triangular patches. Additionally, the trade-off between accuracy and computational speed can be tuned by adjusting the minimum size of the spheres. The spheres provide a convenient representation as a set of obstacles for trajectory planning, but it is not clear that sufficient detail could be produced to be useful for localisation, with a computation time that is feasible for online operation.

### 2.2.2.5 Medial Axis Transform

Medial Axis Transforms [7, 38] use a similar concept of overlapping spheres but using a core representation of a set of axes. These axes represent a skeleton through the centre of the object and are medial axes: meaning they are equidistant to more than one point on the surface. The distance from the medial axis to the boundary of the object can be used to set the radii of a set of overlapping spheres, centred on the axis. The Scale Axis Transform [78] is a modification to the Medial Axis Transform, allowing the resolution of the representation to be reduced to produce a more simplified medial axis but with less accuracy. Representing the object with its medial axis can provide a more compact representation than a set of spheres, but the challenges remain for providing enough detail to be useful for localisation.

### 2.2.2.6 Polygon Mesh

3 D computer graphics is a field that has driven many areas of 3 D modelling and from that field a dominant representation is using polygons to represent the surface. Triangles are the polygons normally used and they are meshed together to produce the surface of an object. The higher the detail, the smaller and more numerous the polygons. The polygons are represented by their corners, stored in order to denote the inside and outside directions. This approach of modelling a 3D object can be viewed as a method of simplifying a point cloud by grouping points into a smaller set of polygon corners.

Being a surface representation, the utility of polygon meshes for localisation is similar to that for NURBS, with observed 3D points matching to surface points and surface normals and a trade-off between computation time and the level of detail. For obstacle representation distances would need to be sampled to the polygon surface, as with NURBS. It is not clear, though, how the polygon mesh would be efficiently updated with new observations. The field of computer graphics could be further investigated to see what techniques may be transferable to the problem of 3 D mapping.

### 2.2.2.7 Summary and Assessment

Table 2.5 presents an assessment of each of the algorithms discussed above. No one algorithm strongly satisfies all criteria, with each algorithm having particular strengths and weaknesses. OctoMap is one
algorithm that stands out but has the limitations described in previous sections for use in localisation. GPIS shows strong potential but with high computational load. NURBS presents a balance across all criteria that may make it suitable for the intended application.

Three algorithms are selected for further investigation in Chapter 3. Firstly the ellipsoids from points clouds algorithm is selected, for its simplicity and speed and immediate suitability for trajectory planning. A limitation of generating ellipsoids from points clouds is the accuracy of the generated rigid body. GPIS is the second algorithm investigated, aiming to maximise the accuracy of representation and have a strong update capability. Finally a NURBS algorithm is developed, with the goal to have a suitable compromise between accuracy and speed. The OctoMap method[91], while also presenting a good compromise of performance, was not investigated due to the limited potential for localisation utility when not using lidar.

TABLE 2.5. Assessment of algorithms for 3D modelling: generation, update, localisation utility and trajectory planning utility

| Algorithm | Representation | Suitability for Database* | Accuracy | Update Capability | Computational Speed | Localisation Utility Potential | Traj. Planning Utility Potential |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Ellipsoids from Point Cloud | Centroid, axes size and orientation | High | Low | Moderate (weighted averaging) | High | Low (centroid and orientation matching, but inconsistent) | High (shapes ideally suited) |
| GPIS | Signed distance from GP at query points | Low | $\begin{gathered} \text { Moderate to } \\ \text { High }^{++} \end{gathered}$ | High (Gaussian Process) | Low | High (probabilistic match of points to surface) | Moderate (sampling query points from GP) |
| GPIS with Ellipsoid Fitting | Centroid, axes size and orientation | High | Moderate | Moderate (weighted averaging) | Low | Moderate (centroid and orientation matching) | High (shapes ideally suited) |
| NURBS | Control points, and knots | Moderate | Moderate to High ${ }^{++}$ | $\begin{gathered} \text { High } \\ \text { (adjust control } \\ \text { points) } \\ \hline \end{gathered}$ | High | Moderate to High (matching points to surface) | Moderate (computing distance to surface) |
| Sphere meshes | Spheres, edges and triangles | Low | Moderate | Moderate (adjust sphere locations) | Moderate | Moderate (match many sphere centroids) | Low to Moderate (simple objects, but too many to be efficient) |
| Scale/Medial Axis Transform | Medial axis (series of lines) | Low | Moderate to High ${ }^{++}$ | Moderate (adjust lines) | Unclear | Moderate | Low (need to generate the 3D object from the medial axis) |
| OctoMap | Occupancy grid | Moderate | Moderate to $\mathrm{High}^{++}$ | High (probabilistic occupancy) | High | Moderate (matching points to occupied cells) | High (collection of simple shapes. Scalable representation) |
| Polygon mesh | Surface patches | Moderate | High | Unclear | Moderate | Moderate to High (match surface patches and normals) | Low to Moderate (simple objects, but too many to be efficient) |

*An assessment of how readily the representation could be stored in a database.
${ }^{++} H i g h e r ~ c o m p u t a t i o n a l ~ e x p e n s e ~ f o r ~ g r e a t e r ~ a c c u r a c y ~$

### 2.3 Trajectory Optimisation

With a physical representation of the obstacles in the environment, there is now sufficient information to optimise obstacle-free trajectories through the environment, for a given optimisation goal such as smoothness. In generating optimal trajectories there are a number of competing goals that need to be balanced:

- Optimising for a trajectory cost (e.g. minimum length, time or acceleration)
- Ensuring clearance from obstacles
- Ensuring dynamic-feasibility
- Having moderate computation cost

Balancing these goals becomes more challenging in 3D environments with complex obstacle fields and dynamic obstacles. Reviewed below are a range of trajectory optimisation approaches to deal with obstacles. First a general overview of trajectory optimisation approaches is presented. Then, approaches to account for obstacles are reviewed, followed by a discussion on approaches for handling dynamic obstacles. Consideration of dynamic-feasibility depends on the hardware system that is being used. A quadrotor UAV is a capable example of a small flying robot, hence the algorithms used for planning quadrotor trajectories are reviewed, focusing on how to achieve dynamic-feasibility for high-acceleration trajectories.

### 2.3.1 Optimisation Approaches

Optimisation of trajectories, with consideration for obstacle and performance constraints, can be done in a range of ways, such as sampling-based algorithms, non-linear programs with inequality constraints and collocation methods. Each of these are direct optimisation methods, using simplification techniques to directly solve the optimisation problem. Direct optimisation methods are in contrast to indirect optimisation methods which solve the problem with Lagrange multipliers and Hamiltonians. This section will briefly outline the different categories of direct optimisation, being the category of methods focused on in this work.

The first category of direct optimisation approaches is sampling-based algorithms, These algorithms include: Rapidly expanding Random Trees (RRT) [110], Fast Marching Trees (FMT) [95] and Probabilistic Road Maps (PRM) [40]. These algorithms take samples in the problem space and connect the new samples to past samples if such a connection is feasible with regards to constraints. Genetic evolutionary algorithms also fit in this category, with numerous generations tested and culled if they are found to be infeasible[43]. The second group of methods is Mixed-Integer Linear or Non-Linear Programming (MILP or MINLP), where a set of inequality constraints are imposed on an optimisation problem. Binary variables are used to turn linear inequality constraints on and off for a given obstacle or performance constraint to have only the closest sides to the trajectory active. The binary variables become part of the larger optimisation problem that is solved. The final group of methods is collocation, also referred to here as polynomial optimisation. In these methods the trajectory is approximated with a set of basis polynomials, controlled by coefficients to shape the polynomial to the desired trajectory. The coefficients are optimised to minimise a trajectory-based cost function and satisfy constraints. The term collocation
relates to the steps of forcing the polynomial to match (collocate) the state as described by the full, non-simplified dynamics, at discrete points along the trajectory. There is a wide range of such methods and readily available tools (such as GPOPS on MATLAB [189]). Refer to references [43, 188, 197] for a review of the different collocation optimisation methods and associated advantages and disadvantages.

In the following discussion, it is important to differentiate between the terms path and trajectory. A path is a sequence of positions, without any consideration of time or dynamics. A trajectory, in contrast, is a sequence of positions and associated times. Therefore, trajectories define velocity, acceleration, and higher derivatives in addition to the positions. Some algorithms produce only a path but generally flying robots required velocity and acceleration information, and therefore a trajectory is needed.

### 2.3.2 Planning with Obstacles

Attribution: The review in this section combines parts of work that have been presented in [35] and [154]. All is the work of the author of this thesis.

There is a range of techniques to deal with the challenge of trajectory optimisation with obstacle constraints; a problem that is inherently non-convex and hence is difficult to solve. These techniques can directly tackle the full problem with the complete representation of the obstacles, simplify the problem into convex sub-problems, or modify the problem to plan within convex, free-space regions. In addition to the method of handling obstacles, the method of representing obstacles is an important consideration that will be discussed.

### 2.3.2.1 Mixed-Integer Approaches

One common approach is to represent obstacles as convex polygons and use binary variables in a MINLP or MILP. The binary variables turn each side of the polygon on and off as an inequality constraint to have only the closest sides to the path active [6, 10, 14, 20, 30, 173, 175]. Richard et al. [10] for example uses MILP to solve for a minimum-fuel path for microsatellites manoeuvring around obstacles outside of the International Space Station. The techniques use branch and bound algorithms [10, 173] to improve the efficiency of the algorithm but with more and more complex environments, the problem becomes more difficult to solve.

### 2.3.2.2 Sampling-based Approaches

Sampling-based techniques, such as genetic algorithms [125, 174] and RRT [110] can successfully plan paths around obstacles. Luo et al. [125] use genetic algorithms with spheres of safety and cones around lines of approach as obstacle constraints. Different branches, or generations of solutions are culled if they violate the obstacles. Generally these techniques have been demonstrated for static obstacles only [25, 125].

Branching and tree-based sampling planners have shown impressive results in complex obstacle-rich environments. Earlier sampling-based planners, such as RRT generated obstacle-free paths, without optimisation of the path or consideration of vehicle dynamics. There have been a large number of extensions to RRT though, such as RRT* [102], MRRT-S [123, 124] and a host of planners in the Open Motion Planning Library [94], that do include optimisation of the path and some of which include dynamics considerations in linking samples (Kino-FMT* [95]). These planners provide all the components desired in the trajectory optimisation, yet the optimisation is asymptotic, meaning the algorithm takes longer to achieve an optimised trajectory, in contrast to algorithms that are immediately solving an optimisation problem.

### 2.3.2.3 Hierarchical Approaches

Sampling planners do often play a large role in the first stage of a hierarchical planner, when a dynamically optimised solution is not required. In hierarchical planers, an algorithm like RRT generates a global, collision-free path, from which waypoints are extracted for a lower level trajectory optimiser, such as in [5, 26, 63]. In these approaches, the lower level optimiser is a polynomial optimisation approach that solves for the polynomial coefficients to minimise a trajectory cost such as minimum acceleration or minimum snap (the fourth derivative of position with respect to time), while complying with the boundary conditions at the waypoints. A separate polynomial is used for each segment between waypoints, with continuity constraints enforced. These techniques are discussed in more detail in Section 2.3.4. Sampled points along the global path and trajectory are checked for violations against an obstacle representation, commonly an occupancy grid such as OctoMap [91]. If an optimised solution is found to be in a collision, an extra waypoint from the collision-free global path is added in the middle of the segment that has a collision and the trajectory is planned again in an iterative process. While effective, this approach does not consider the dynamic-optimality of the overall problem.

### 2.3.2.4 Polynomial Optimisation Approaches

Polynomial optimisation algorithms can optimise the overall problem to have dynamically smooth trajectories that are also collision-free. These algorithms can work for a range of obstacle representations, and the choice of representation has a large impact on the performance and capability of the algorithm. Therefore, a range of different obstacle representations that have been used for trajectory optimisation is reviewed here. Oleynikova et al. [170] utilise and ESDF in a polynomial optimisation by adding a collision cost to the trajectory cost and using gradient descent to optimise. The collision cost is a discretised path-integral that samples the obstacle distance and gradient from the ESDF. This has been shown to run effectively and online, for short indoor and outdoor trajectories.

Potential fields, as employed by Munoz and Fitz-Coy[160] are another obstacle representation, similar to ESDFs, that have been used in a variety of forms to plan trajectories around obstacles [36, 113, 116, 131, 135, 196, 227]. The goal location is given an attractive potential and obstacles present repulsive potentials. Most of the techniques that have been demonstrated are limited to two-dimensional problems with a static obstacle field.

Primitive shapes can also be used to represent obstacles in the environment, such as a collection of spheres, ellipsoids, cylinders and cubes, as will be described in this work. The obstacles can be considered in a similar way to an ESDF, by sampling collisions, computing a gradient of cost and including the cost of collision in combination with the trajectory cost for the overall optimisation. With this approach, a mix of obstacle representations can be considered, such as combining a pre-mapped ESDF with ellipsoid obstacles for newly added obstacles in the environment.

Polynomial optimisation algorithms can be effective in solving collision-free, dynamically-feasible trajectories, yet there can be challenges in solving what is inherently a non-convex problem. Solutions can take a lot of time to compute and may not be successful in every scenario.

### 2.3.2.5 Simplification Approaches

Instead of solving the complete non-convex problem with full obstacle representation, simplification methods can be used. Blackmore et al. [18] use a disjunctive convex program, where the overall nonconvex problem is split into convex sub-problems. These sub-problems gradually build up to the complete constraint set, through branch and bound algorithms. Other techniques have been applied to similarly break up and simplify the problem and then gradually build up the complexity, such as Lu and Liu [122] with Second-Order Cone Programming (SOCP) and Eren et al. [60] where the problem is modified in numerous ways to overcome each source of non-convexity. Another example by Kobilarov et al. [105] uses a technique they term homotopy, where the initial solution is obstacle-free and then the problem is iteratively deformed to the final orientation by growing the size of obstacle constraints, keeping the trajectory in the same homotopic class.

### 2.3.2.6 Free-Space Planning Approaches

Another approach to deal with the non-convexity of planning with obstacles is to restrict planning to be within free-space and hence within sets of convex regions. Landry [114] uses a set of hyperplanes to define the convex regions, using an iterative region inflation algorithm. Semi-Definite Programming is then used to produce optimal trajectories, where each segment is restricted to stay within one convex region. The region assigned to each segment is selected with a mixed integer algorithm. This approach can plan through dense obstacle fields but takes more than 10 minutes to solve on a commodity computer. Lioanno et al. [121] similarly use inequality constraints on individual segments. The constraints force the trajectory to remain in a corridor and are included in a constrained quadratic program.

Another free-space representation is with overlapping sets of spheres. Baldini et al. [11] generate an initial plan with Spherical Expansion giving to give a collision-free path. Spheres are expanded around sampled points to grow to the maximum radius available in free-space. Sequential Convex Programming is then used to optimise the trajectory with the constraint to stay within the spheres. The algorithm that will be described in this work can also include convex position constraints, to force the trajectory to stay within ellipsoidal or cylindrical regions, with the same approach as used for obstacle constraints.

One limitation of all of the approaches described above is that the trajectory is discretised for collision or constraint checking, hence there is a chance of a collision occurring in-between sample
points. This risk can especially be an issue for environments with thin obstacles or flights of high velocity and acceleration. The trajectory discretisation is in time, rather than in space, which potentially leaves large regions of unsampled trajectory with large changes in direction or large velocities. The approach by Campos-Macias et al. [32] provides a way to avoid this time-discretisation limitation by sampling in space for collisions. The technique starts with a globally planned, high clearance path, represented by a set of waypoints and computes the minimum distance to each straight-line segment between waypoints, with spatially discretised sampling. Subwaypoints are then created along the straight-line segments with inequality constraints on position, velocity and acceleration. The constraint values, location of the subwaypoints and time allocations to subsegments are computed to ensure that the optimised trajectory will stay within cylindrical bounds. By setting the constraint values so that the cylindrical bound for each segment is below the computed minimum distance to the nearest obstacle, a collision-free path can be assured. The spatially discretised sampling only occurs once, so it is possible to have a fine discretisation to ensure accurate computation of the minimum distance. Another approach to ensure collisions are not missed is to sample in arc-length increments that are equal to the resolution of the obstacle representation, as done by Oleynikova et al. [170].

### 2.3.2.7 Summary and Assessment

Of the range of techniques discussed above, the best approach depends on the given application and the nature of obstacles, vehicle dynamics and computational resources. In general though, for navigating through medium to large areas that are dense with obstacles, a combination of the best of samplingbased planners with polynomial optimisation algorithms is most suitable: to efficiently generate an obstacle-free global path and then efficiently optimise the trajectory along it to be dynamically smooth. The trajectory optimisation algorithm described in this work takes a middle ground between samplingbased planners that can produce non-optimised paths efficiently through very complex obstacle fields and trajectory optimisers that can operate over a small distance with few or no obstacles. The algorithm produces dynamically-optimised trajectories through complex obstacle fields, allowing either a single stage planner, or a reduced set of waypoints from the globally planned path.

### 2.3.3 Planning with Dynamic Obstacles

Attribution: The work in this section has previously been presented in [155]. All is the work of the author of this thesis.

Many of the algorithms described above are not able to handle dynamic obstacles as they are designed for static obstacle fields. There are three main themes of considering dynamic obstacles in existing work: reactive local planning or control, rapid replanning and modelling the dynamics of the obstacles in the generation of a trajectory. With each of these approaches there are perception requirements to be able to detect obstacles and potentially model their dynamics.

### 2.3.3.1 Reactive Local Control

Reactive planning algorithms adjust a trajectory over a short time window to adapt to changes in the environment, such as a dynamic window approach [69] which selects from a tree of short term control inputs to ensure clearance of obstacles. Forcing functions could also be placed in the position controller, to have short term obstacle avoidance in the control loop [75]. These reactive approaches require the efficient detection of new obstacles and the ability to model the obstacles in a method that is efficient for collision checking or as a forcing function.

### 2.3.3.2 Rapid Replanning

Rapid replanning approaches work on similar principles of updating a plan to adapt to changes but plan longer-term trajectories and work at a slower rate. The techniques rely on quick planning times, along with an efficient method to update the obstacle representation in the whole environment. Some of the approaches described below are designed to enable high-speed flight through unknown static obstacle fields: a capability that can similarly apply to lower speed flight through dynamic obstacle fields. A common approach to quickly generate trajectories is to use motion primitives: a small set of simple trajectory pieces that can efficiently be combined together to form a complete trajectory [158, 176]. Such techniques have shown rapid replanning capability, including a quadrotor reacting to return a ping pong ball [158] (Fig. 2.13.b) and fast flight around obstacles [176] (Fig. 2.13.b).


Figure 2.13. Rapid replanning examples. (a) Sets of candidate trajectories to intercept a ping-pong ball from [158]. (b) Candidate trajectories taking obstacles into account from [176]. Images are from the respective publications.

A similar approach to using motion primitives is pre-computing a library of potential trajectories and selecting from the library in flight. Barry et al. [13] use this approach, along with an efficient way to detect obstacles, called push-broom stereo. A trajectory library is generated for a fixed-wing UAV with off-line optimisation and flying by hand. When replanning online, the trajectories in the library
are transformed to the current position and yaw, then the trajectory that maximises the minimum obstacle clearance is selected. Push-broom stereo only considers obstacles at one depth and uses the forward motion of the vehicle to scan through the environment. While shown to be very effective for online, high-speed flight, the approach has some limitations if applied to dynamic obstacles that aren't detected at the push-broom depth, being closer or passing through the detection region too quickly.

Another set of approaches to efficiently update the obstacle representation for path planning are to operate in the image space [21, 134]. Matthies et al. [134] detect obstacles from stereo images in the disparity space (disparity between pixels in a stereo image pair) and also plan trajectories in that space, giving a computationally efficient approach. Brockers et al. [21] similarly use stereo vision but project observations onto an egocylindical space. This egocylindrical space gives a 2.5 D representation that allows for efficient collision checking and motion planning. The demonstrations show 2 Hz update and planning rates on a quadrotor, enough to react to obstacles in some environments.

ESDFs can also be rapidly updated to use for online dynamic obstacle avoidance, such as with Voxblox [171], which has been demonstrated to be able to update at 4 Hz . With an environment representation that can be updated to changes in the environment, Oleynikova et al. [170] replan trajectories with a polynomial optimisation algorithm. This approach has not yet been demonstrated with dynamic obstacles but has the potential to do so for slow-moving obstacles.

To enable robust performance with a range of obstacle dynamics, a hybrid approach can be taken: combining a reactive controller with a rapid trajectory planner, as done by Allen et al. [5]. The reactive controller [75] provides position adjustment for fast moving obstacles and reduces how often a trajectory needs to be replanned. A pre-computed roadmap around the static obstacles is then used for quick replanning of trajectories. This has shown to be effective with high-speed obstacles such as a fencing blade, using a Vicon motion tracking system to detect the obstacles.

The need for a hybrid system can be waived if the full trajectory planning can be done at a high enough speed to quickly react to obstacles in the control loop. Potential for such capability has been shown with Group Marching Trees (GMT*) [93], which modifies FMT* to expand groups of state samples in parallel. By implementing GMT* with GPUs, paths can be planned through complex obstacle fields at up to 100 Hz .

### 2.3.3.3 Modelling Obstacle Dynamics

If the motion of dynamic obstacles can be predicted, with some uncertainty characterisation, then there is less need for reactive, rapid replanning and a full trajectory can be generated that considers the likely motion of the obstacle. The method of detection and modelling dynamics obstacles is itself a large topic of research, but for the purposes of looking at trajectory planning approaches, if knowledge of the dynamic obstacles is assumed, generally with constant velocity models, then there are a variety of techniques to consider.

The constant velocity motion of an obstacle could be used to define collision cones, where the axis of the cone extends in the velocity direction and the radius of the cross section grows as a function of uncertainty or likely deviation from constant velocity. Collision cones have been integrated into potential field approaches [113] and with genetic algorithms [36]. The collision cones can be quite conservative though, blocking more physical space than what is actually occupied by an obstacle at any given moment.

By instead defining a collision cone in the velocity space, a less conservative representation can be used, called a Velocity Obstacle [64]. Tree-based searches with such obstacle representations can allow for quick planning, which can be required as the Velocity Obstacles assume constant velocities for both the robot and obstacles. In the case of general motion of a dynamic obstacle, any technique that uses a constant velocity assumption will need to regularly replan but at a lesser rate than reactive, rapid replanning approaches.

For considering more complex obstacle dynamics, techniques that plan trajectories for multiple vehicles could be adapted to the problem, as other vehicles represent dynamic obstacles. Many of these approaches are inspired by the flocking behaviour of birds and schools of fish. Xue et al. [227] includes both dynamic obstacles and planning for multiple vehicles, where an avoidance strategy is implemented on top of a potential field only once an obstacle is within close proximity. A potential field could instead be modified using gyroscopic forcing, as demonstrated by Chang et al. [36] to maintain separation between pairs of vehicles in a swarm of many vehicles. These approaches have only been demonstrated in 2 D and are reactionary during close encounters, rather than planning a complete, obstacle-free trajectory. Other approaches that also plan trajectories for all vehicles have been demonstrated with MINLP and binary variables [6, 14, 30, 173, 175, 194], combinations of probabilistic roadmaps [214] and resource allocation systems [193]. These techniques centrally coordinate the trajectories of all vehicles in 2D. More recent demonstrations have shown real-world demonstrations of multi-robot planning for in 3D, such as the 1000 quadrotors from Intel [101] and Ehang [54], which use pre-planned, centrally controlled trajectories. The Mixed Integer approach has also been extended to Mixed Integer Quadratic Programs to coordinate the trajectories of swarms of 16 quadrotors flying through windows [112]. In contrast to these centralised approaches, Vasarhelyi et al. [220] demonstrate a decentralised approach for quadrotors with 30 vehicles flying collision-free trajectories around obstacles, based on principles of repulsion and alignment of velocities. The principles in these algorithms could potentially be adapted to apply to a single robot avoiding dynamic obstacles, with centralised approaches requiring the other vehicles to have known, fixed trajectories.

There are fewer approaches that are designed for a single robot and motion models of the dynamic obstacles. One example by Ousingsawat et al. [174] models the estimated trajectory of dynamic obstacles along with position uncertainty to use in evolutionary optimisation. The work described here looks to account for a range of motion models of dynamics obstacles, along with the uncertainty of their position, to use in a polynomial optimisation of a complete trajectory.

### 2.3.3.4 Summary and Assessment

Different perceptual and computational challenges come when comparing approaches to update the complete obstacle environment, with approaches to detect and model the dynamics of obstacles. It can be more costly to efficiently update a full environment with highly dynamic obstacles, yet detecting and tracking the movement of such obstacles can also be a large challenge. Preference is given here to modelling obstacle dynamics as it allows for a more optimal complete path, can nicely capture uncertainty and may require less frequent replanning.

For trajectory planning, a combination of techniques described above gives the most reliable system. The recommended approach is to use constant velocity models with a planner that considers the
obstacles in the complete trajectory, rather than in pair-wise interactions. The obstacle representation should account for uncertainty and be time-dependent, i.e. the space that the obstacle no longer occupies should be free. The planner should be relatively quick as well, to enable frequent updates in the case of obstacle deviation from constant velocity. An additional, reactive controller may also be required to safely avoid nearby and high-speed obstacles and to enable the planner to run at a lower rate. Components of such a system are present in the techniques discussed above. This work looks to develop a trajectory optimiser that can fill the role of the planner in the dynamic obstacle avoidance system.

### 2.3.4 Trajectory Planning for Quadrotor UAVs

Attribution: Parts of this section have been presented in [154] and [192] and is the work of the author of this thesis.

Two critical components of achieving autonomous flight of quadrotors are 1) trajectory planning, to produce dynamically-feasible, collision-free trajectories and 2) trajectory tracking controllers to closely follow the planned trajectory.

Most approaches for trajectory planning and control for quadrotors utilises a differential flatness transform [164, 219], which allows direct mapping from the flat outputs of $x, y, z$ and $\psi$, (where $\psi$ is yaw), plus their derivatives, through the full quadrotor state to the flat inputs: the input RPM-squared (Revolutions Per Minute, squared) for each motor. A continuous trajectory planned in the flat output space transforms to a continuous trajectory in the flat input space, providing a convenient method to ensure a dynamically-feasible trajectory (given the RPM magnitudes are within their limits). A detailed overview of the differential flatness transformation is presented in Chapter 5.

Existing methods used for trajectory optimisation with quadrotors will be discussed, describing in more detail some of the techniques mentioned in the previous section. The discussion will include: approaches to generate waypoints and methods to achieve aggressive (high-accelerations) flight. The controllers to track such trajectories will then be described.

### 2.3.4.1 Trajectory Planning

The state-of-the-art in trajectory planning for quadrotors take a polynomial optimisation approach: they optimise the coefficients of piecewise polynomials to minimise the integral of snap squared (the ${ }^{4 t h}$ derivative of position with respect to time) over multiple segments. These segments are planned between a set of waypoints, with an outer loop optimisation run to reassign the time allocated to each segment to minimise a weighted cost function combining snap and time. There have been numerous impressive demonstrations using this approach, such as by Mellinger et al. [139] and Lioanno et al. [121] flying through a narrow vertical window (Fig. 2.14.a), Allen et al. [5] flying through tight indoor constraints, Thomas et al. [216] achieving perching type orientations, Fang et al. [63] demonstrating onboard autonomous navigation through a naval ship (Fig. 2.14.b) and Bry et al. [26] achieving fast flight through crowded indoor environments (Fig. 2.14.c).


FIGURE 2.14. Demonstrated applications of quadrotor trajectory planning. (a) Flight through a narrow vertical window [121]. (b) Autonomous navigation through a naval ship [63]. (c) Flight through crowded indoor environments [26]. Images are from the respective publications.

The optimisation problem can be solved in a number of ways, in particular, the optimisation of snap, which is initially posed as a quadratic program constrained by the boundary conditions. A key step is in the inversion of a matrix incorporating the boundary conditions and cost function, a matrix that can become ill-conditioned and difficult to invert. Here, Fang et al. [63] make adjustments to the matrix to make it invertible. In contrast, Bry et al. [26] modify the problem via substitution to solve for the free derivatives at the waypoints, instead of the polynomial coefficients. This substitution inherently enforces the boundary conditions translating the problem into an unconstrained optimisation. Other approaches, such as by Thomas et al. [216] use commercial optimisers to generate solutions. Work presented here has found that the approach of Bry is quick, robust and does not require access to software licenses.

Gradient descent is used by Bry et al. [26] and Fang et al. [63] to perform the outer loop time optimisation, with the segment times as the decision variables. Another approach used by Allen et al. [5] is to incorporate the optimal time in the selection of the waypoints between which the trajectory is planned. They pre-plan a roadmap with RRT* [102] and then use Kino-FMT* [95] to plan on the map, producing waypoints with optimal times between them.

Snap is minimised in the trajectory optimisation algorithms because snap maps directly to motor RPM squared, in the quadrotor differential flatness transformation, as elaborated in Chapter 5 . Hence by minimising snap, the control effort is minimised, giving a trajectory that is easier to track. Nonetheless, other cost functions could be more appropriate for a given application, for instance minimising time, with constraints on controls to ensure feasibility. Hehn et al. [87] formulate a minimum time problem and plan the jerk (the ${ }^{3 r d}$ derivative with respect to time) of the trajectory. The result is bang-singular paths (segments of full control and no control), where control constraints are accommodated by decoupled constraints on jerk and acceleration, which are included in the optimisation. While very quick to compute, the formulation is designed for low speed, close to hover. Spedicato et al. [210] similarly minimises time in a gradient descent approach with the trajectory parameterised about a reference trajectory with transverse coordinates. These approaches have only been demonstrated in simulation.

### 2.3.4.2 Generating Waypoints

Each of the algorithms mentioned above requires a set of waypoints between which to plan. The waypoints could be manually determined, or a sampling-based planner could generate feasible paths, such as with RRT, which can then be reduced to a minimum set of waypoints, such as in [26, 32, 170]. Fang et al. [63] plan candidate 2D paths and then use an optimiser to select the best 4D path (position and yaw), after which waypoints are down-sampled. As discussed above, Allen et al. [5] generates waypoints with a combination of RRT* and Kino-FMT* to get the waypoints.

If the trajectory optimisation algorithms perform better with a sparse set of waypoints, then a line simplification algorithm can be employed to reduce the number of waypoints used, while maintaining a similar shape. The Ramer-Douglas-Peucker algorithm is one common method that will reduce the number of waypoints with the magnitude of reduction being controlled by a user input setting [90]. This approach of using line simplification is useful in a teach-and-repeat scenario where a pre-flown trajectory has a dense set of positions recorded.

### 2.3.4.3 Aggressive Manoeuvres

As quadrotors can fly at higher speeds in obstacle-rich environments, there comes challenges in planning and executing high-acceleration manoeuvres, which are referred to as aggressive manoeuvres. Such capabilities are of use for changing direction in confined spaces and for dynamic obstacle avoidance.

Aggressive manoeuvres for quadrotors can be achieved with several different approaches. Switching controllers is one such approach, where a sequence of distinct control stages is used in sequence to complete a particular manoeuvre. These stages tend to include a trajectory-following launch stage, a ballistic coasting phase and a recovery phase. Lupashin et al. [126] used a 5 step bang-bang manoeuvre sequence, with learned tuning parameters to achieve multiple flips (Fig. 2.15.a). Mellinger et al. [140] similarly learned parameters and employed a 3 stage controller, that was used to perform flips, flights through windows and flights through moving hoops. Both of these approaches designed custom PID controllers for each control stage. Falanga et al. [62] more recently used a 3 stage sequence for highspeed flight through windows (Fig. 2.15.b), utilising the auto-recovery work from Faessler et al. [61] in the final stage. The recovery approach of Faessler et al., itself, is a multi-stage control algorithm, going through 5 stages of control to recover from a large range of orientations (including being hand thrown), to eventually hold position. The recovery sequence uses a range-finder and down-facing camera.

Some of the most aggressive manoeuvres demonstrated, including a split-S have been with autonomous helicopters, where the specific control sequence for a manoeuvre is learned from an expert pilot by matching a model to the flight data [1, 74]. The helicopters have higher control authority than quadrotors though, with reverse thrust capability from controlling the blade pitch.

While effective, the approaches mentioned above are custom solutions for particular sets of manoeuvres. For general operation, it is advantageous to have a single control scheme for a range of manoeuvres, including non-aggressive flight. This capability can be achieved with a trajectory planner, a capable trajectory tracking controller and the use of waypoints to constrain the flight to achieve given manoeuvres. Thomas et al. [216] use the trajectory planning approach from Mellinger et al. [139], with a specification of final states to perch vertically, or inverted. The demonstrations show successful


Figure 2.15. Examples of aggressive manoeuvres for quadrotors. (a) A sequence of flips that are planned (i), initially flown (ii), and then flown after learning parameters for a 5 step manoeuvre (iii) [126]. (b) Three stage control sequence for flight through a narrow window [62]. (c) Execution of a planned trajectory to perch vertically [216]. Images are from the respective publications.
manoeuvres operating about a single axes(Fig. 2.15.c). Loianno et al. [121] use a similar approach to fly through vertical narrow window slots, with waypoints used to force the flight through the gap (Fig. 2.14.a). Constraints on control and position are included in the planning to achieve a dynamicallyfeasible trajectory. Neunert et al. [165] also constrains the trajectory with waypoints for flight through windows, incorporated in a Model Predictive Control framework.

### 2.3.4.4 Controls

A hierarchical control architecture is the most commonly used for tracking a planned trajectory. This architecture consists of an outer-loop position controller and an inner-loop attitude controller, as depicted in Fig. 2.16. The differential flatness transformation again plays a key role: the outer loop controller gives a desired thrust vector ( $\mathbf{T}_{s p}$ ); this vector is transformed, through part of the differential flatness transformation, to a desired attitude ( $\mathbf{q}_{s p}$ ) and thrust magnitude ( $T$ ) for the inner loop attitude controller to track. The output from the attitude controller is the torques ( $\tau$ ) and net thrust for the motor controller to track.

Despite the frequent and successful use of the differential flatness transform, there are known singularities: 1) occurring when there is zero desired thrust (when the desired acceleration is fully achieved by gravity) and 2) when the desired thrust vector is in the $x y$ plane and aligned with the


Figure 2.16. Hierarchical controller, with the differential flatness transform a key link between the outer position controller and inner attitude controller. Subscript $s p$ denotes Set Point, $\psi_{s p}$ is the yaw set point. The thrust set point $\mathbf{T}_{s p}$ is transformed to the attitude set point: the quaternion $\mathbf{q}_{s p}$
desired direction of travel (e.g. pitched forward at $90^{\circ}$ ). The first singularity is a fundamental limitation of the transformation, which is founded on the notion that the desired thrust direction sets the quadrotor attitude. The second singularity and the sensitivity of states near this singularity is something that can be managed. The differential flatness transformation will be analysed in detail in Chapter 5.

The controller by Lee et al. [118] is widely implemented to strong success [5, 26, 121, 139, 216] and uses a PD controller with feedforward acceleration in the position controller and a PD controller with feedforward angular acceleration and gyroscopic correction in the attitude controller. Variations and simplifications to the controller by Lee et al. [118] are also used, such as removing the feedforward terms in the attitude controller [139]. Other controllers use mixes of PID controllers. The PX4 flight software (used by [52, 63]) has an independent PD for $x, y$ and $z$ position tracking, then a PID for each of the rotational axes [138].

Another approach is to use Sequential Linear Quadratic problems in a Model Predictive Control framework to compute time-varying feedback gains for the position controller [165]. Alternatively, a Linear Quadratic Regulator (LQR) approach can be taken, which still results in PD controllers but has the advantage of tuning of the $Q$ and $R$ matrices to solve for the feedback gains, rather than tuning the gains directly [61, 114]. In addition to the LQR controller, Faessler et al. [61] add feedforward terms in the attitude controller and employ iterative thrust mixing to solve for the propeller torque coefficients, which are modelled as functions of propeller rotational speed.

To further improve trajectory tracking, a feedback controller can be placed around the motor RPM, to be run on the Electronic Speed Controllers (ESCs). Bangura [12] takes this approach and also includes aerodynamics modelling of the propellers in the position controller.

### 2.3.4.5 Summary and Assessment

For high-speed flight in a range of cluttered environments, it is desirable to have one planner that can produce dynamically-feasible trajectories, consider obstacles and can plan aggressive manoeuvres without the need for a switching controller. The literature suggests that a minimum snap trajectory optimiser, with considerations for obstacles, is a strong candidate for such applications. For tracking the trajectory, the frequent and successful use of the controller by Lee et al. [118], makes it stand out as the best approach for agile flight.

### 2.4 Complete Systems

This section reviews examples of complete systems that demonstrate the desired capability: online mapping and trajectory planning in 3D to enable autonomous navigation in unknown environments. While there are many ground-based systems, the focus will be on systems that navigate in 3D, with most examples from quadrotor systems, due to the recent strong research interest in the flying platforms.

A key trend in systems capable of online mapping and planning is heterogeneous algorithms: a split of different algorithms and environment representations for mapping as well as a hierarchy of algorithms for planning.

The state-of-the-art for mapping, planning and obstacle avoidance on quadrotors is the same as has been described in previous sections. The combination of approaches on particular systems varies though, as will be explored here for a range of state-of-the-art systems.

### 2.4.1 Software

The different algorithms for localisation, local mapping, global mapping, local planning, global planning and control, all tend to run as different processes, with only the minimum required information exchanged between them. With similarly defined input and output, each component could potentially be exchanged with another, similarly capable algorithm. This type of architecture is at the core of the Robot Operating System (ROS) [186], a middle-ware for handling standard communications between different robotic processes, such as is used by Perez-Grau et al. [177] and Jung et al. [99].

### 2.4.2 System Examples

Examples of complete systems are summarised here to highlight the combination of localisation, mapping and planning that is possible.

Mohta et al. [147] demonstrate high-speed autonomous flight of a quadrotor both indoors and outdoors. They use forward facing stereo cameras for VIO, using the SVO algorithm and fuse with a dedicated accelerometer and gyro and a laser altimeter. There is no global localisation. A 2D laser scanner on a 1D gimbal is used to build a local occupancy map, which is gradually used, online, to build a 2D global occupancy map. A* is used to produce a global path on a hybrid of a local 3D occupancy grid and a global 2D occupancy grid. They then use a region inflation approach to produce convex regions for planning obstacle-free trajectories in the local planner (similar to techniques described in section 2.3.2.6). Weightings are applied to the trajectory to push it away from obstacles in the local map. All algorithms run online and with impressive high-speed results (up to $4 \mathrm{~m} / \mathrm{s}$ indoors amongst obstacles). A photograph of their system can be seen in Fig. 2.17.a.

Perez-Grau et al. [177] use an integrated stereo and IMU sensor for VIO and also have an RGBD sensor which they use for localisation and mapping. The localisation is with point cloud matching to a pre-mapped 3D probability grid, to include in a Monte Carlo localisation approach. They use a global occupancy grid and a local occupancy grid built in real time around the drone with the RGBD point cloud. For global planning, they use Lazy Theta* on a combination of local and global maps, and for local planning, they also use Lazy Theta* but over a shorter time horizon. They demonstrate obstacle avoidance indoors at speeds of up to $1.5 \mathrm{~m} / \mathrm{s}$.


Figure 2.17. Example quadrotor systems. (a) Photograph of the quadrotor from Mohta et al. [147]. (b) Diagram of the quadrotor from Droeschel et al. [52]. Images are from the respective publications.

Droeschel et al. [52] use a large sensor suite, with GPS, a down facing optical flow camera, a spinning lidar, two fish-eye stereo pairs and eight ultrasonic sensors, as depicted in Fig. 2.17.b. Despite all the sensors, similar algorithms to the systems above are used. The stereo cameras are used for VIO, and the laser scan is matched with a multi-resolution surfel grid for localisation. The surfel grid is also the local map representation, centred around the robot. SLAM with Pose-graph optimisation is used to combine local maps to produce a global occupancy map. Four layers of planners are used, with first a mission planner to get sparse waypoints, then a global plan with $\mathrm{A}^{*}$ at 0.2 Hz in the global, static environment, then a local plan in the obstacle grid with the generation of a graph and an $A^{*}$ search on the graph. Reactive obstacle avoidance with forcing potential fields are then used to push the trajectory away from newly observed obstacles. The system shows the capability to explore and map unknown areas.

Fang et al. [63] also use an obstacle map for localisation. Their localisation routine fuses information from multiple sensors: a down facing optical flow sensor, a laser altimeter, VIO with RGBD cameras, then matching the RGBD point cloud to a known global OctoMap. A local occupancy grid is created around the robot with the RGBD point cloud. For planning, a 2D slice of the global OctoMap is used to produce a Voronoi diagram, on which $\mathrm{A}^{*}$ is used to generate the global plan. Local planning uses CHOMP to produce waypoints between which polynomials are optimised and local path library used in a receding horizon control for reactive obstacle avoidance. They show autonomous navigation within a naval ship, through tight confines.

Oleynikova et al. [172] use forward facing stereo cameras and IMUs for VIO and have no global localisation. RGBD point clouds are used to build TSDFs online and then quickly convert them to ESDFs for use in motion planning. Global planning is done with informed RRT* and local planning with polynomial optimisation (as in [26]) to a given planning horizon, with the addition of obstacle costs from the ESDF in the optimisation. They also have a higher level of exploration planning, to select the next goal for moving in an unknown environment. Their demonstrations show successful flight through unknown indoor and outdoor environments.

### 2.4.3 Current State-of-The-Art

The state-of-the-art from the systems presented above is to take a heterogeneous approach to both mapping and planning. The particular combination of components can affect the performance of the overall system, but for comparisons later in this work, ORB-SLAM2 [163] is considered as the leading VIO algorithm with Voxblox [171] as the leading 3D mapping algorithm. For trajectory optimisation, the approach of Bry et al. [26] is considered the leading algorithm, with control from a controller similar to that described by Lee et al. [118].

### 2.5 Summary and Identification of Gaps

From the review of the components of the navigation stack, there were a number of gaps identified in the capability for small flying robots, in particular with restrictions for onboard computation and light-weight, low-powered sensors. These gaps are summarised below:

1. The state-of-the-art in SLAM algorithms that are applicable for 3D navigation of small flying robots, with light and low-power sensors, do not produce a map that is a suitable representation of the 3D environment for trajectory planning.

- For algorithms that do not require lidar:
- The state-of-the-art in 3D mapping algorithms require localisation.
- 3D mapping algorithms do not provide sufficient detail to create a map that can be used for localisation in addition to obstacle avoidance.

2. Leading planning approaches are heterogeneous, splitting global path planning and trajectory optimisation. There is a gap in algorithms that can provide the middle-ground: to produce a global, dynamically-optimised trajectory.
3. There is a limited capability in trajectory optimisation algorithms to consider motion models of dynamic obstacles in 3D. Doing so could produce more informed and less conservative trajectories.
4. There has not been an in-depth analysis of the impact of the singularities in the differential flatness transformation and the different strategies to address them.
5. There has not been an analysis of the impact of obstacle avoidance strategies on the dynamicfeasibility of trajectory optimisation algorithms.

In contrast to the leading, complete systems that take a heterogeneous approach to both mapping and planning, the system proposed here aims to take a homogeneous approach. First, the aim is to have one map representation that can be used in SLAM as well as for representation of obstacles. The goal in this approach is to have a more efficient system with less duplication of processes, as well as a representation with future applicability. Secondly, the trajectory planning algorithm proposed looks to provide a similar capability to both a global planner and a trajectory optimiser, by enabling obstacle-free planning over a large horizon and with the ability to include information about dynamic obstacles. For operation in large environments, a separate global planner may still be required. Similarly, in highly dynamic, uncertain or crowded environments, a reactive collision avoidance planner may also be required but only for a sparse set of waypoints. Specifically, the work in this thesis addresses the gaps identified above in the following ways:

Gap 1 A SLAM algorithm is developed that models 3D objects with NURBS to use as both landmarks for localisation and as obstacles for trajectory planning. Distinct objects are modelled, rather than the whole environment, providing a balance between computational load and detail, while also being a representation with future applicability to dynamic obstacles, object grasping and object classification. Additionally, surfaces are modelled, allowing objects to be represented at multiple resolutions to suit the purposes of localisation and trajectory planning. The approach is designed to use 3D observations of the environment from RGBD cameras.
Gap 2 A trajectory optimisation algorithm is presented here that is capable of planning in complex obstacle fields over large horizons while optimising dynamics.
Gap 3 The trajectory optimisation algorithm considers dynamic obstacles with a motion model and propagation of uncertainty.
Gap 4 The range of methods for handling the singularities in the differential flatness transformation are analysed in detail to highlight where problems occur. More robust methods are proposed and tested in flight.
Gap 5 A review is performed of trajectory optimisation for quadrotors flying near obstacles. This review includes flight tests to assess the dynamic-feasibility of trajectories and how different strategies of considering obstacles affects the dynamic-feasibility.

The remaining chapters in this thesis present the work outlined above. The proposed SLAM algorithm is first described in Chapter 3. Next, the trajectory optimisation algorithm is described and analysed in Chapter 4. Applications of the algorithm to quadrotors are described in Chapter 5, including an analysis of the differential flatness transformation. Flight tests assessing dynamic-feasibility and differential flatness transformations are then presented in Chapter 6. Each component of the system is brought together in Chapter 7 to show a complete autonomous navigation system and compare to the state-of-the-art in a novel simulation framework.


## Localisation and MAPPING With 3D ObJECT REPRESENTATIONS



The goal of the work in this chapter is to combine the odometry, localisation and mapping layers of the autonomous navigation stack by developing a SLAM algorithm that produces a map of 3D obstacles. Modelling 3D obstacles, in contrast to the complete environment, provides a way to divide the mapping problem, as well as producing a representation with uses beyond localisation and mapping: for dynamic obstacles, object interaction and object classification. The central part of this goal is the method of representing the 3 D objects in the environment. The one 3 D representation needs to be useful for mapping, localisation, and obstacle representation. This chapter first reviews three candidates for 3D object representation. The review includes analysis of a complete implementation of ellipsoid models for SLAM and obstacle representation, followed by an assessment of the potential for Gaussian Process Implicit Surfaces (GPIS). Non-Uniform Rational B-Splines (NURBS) are then reviewed as a 3D object representation. The review justifies the selection of NURBS to apply to SLAM and trajectory planning. The method of applying NURBS to these problems is described in detail, in an algorithm defined as NURBS Localisation And Mapping (NURBSLAM). The chapter ends with test results on simulated data that demonstrate the concept of using NURBS objects for mapping, localisation and trajectory planning.

### 3.1 Review of Candidate 3D Object Representations

The literature review in Section 2.2 presents a range of algorithms to model 3D objects and assesses their applicability for use in autonomous navigation. Three candidate algorithms were identified: Ellipsoids, GPIS and NURBS. This section will describe how these 3D modelling algorithms can be applied to autonomous navigation, specifically to perform the tasks of:

- Data association
- Mapping
- Modelling 3D objects from point clouds
- Updating 3D objects from multiple observations
- Localisation
- Obstacle representation for trajectory planning

The desired capabilities of the 3D modelling algorithm are to:

1. Accurately model a 3D object with multiple observations.
2. Provide a representation that is useful for localisation.
a) Enable reliable and accurate matching from multiple observations to give information on the movement of the robot.
3. Provide a representation that is suitable as an obstacle representation.
a) Geometric paths through the 3 D shape should be correctly detected as being in a collision.
b) The query of collisions for a geometric path should be quick.
c) Ideally the representation provides a gradient of collision violation.
4. Have minimal computation load.

Each algorithm will be assessed with regards to these capabilities, and the strengths and weaknesses highlighted. The analysis of the three algorithms will justify the selection of one algorithm, NURBS, for further development in Section 3.2. For each algorithm, it is assumed that the input is a segmented point cloud, with each object identified in different segments. The algorithms then operate on one object at a time. The segmentation approach is not described in detail in this work, but more information can be found in Kuether et al. [109].

### 3.1.1 Ellipsoids - Full Application to SLAM and Trajectory Planning

Attribution: The theory and results in this sub-section were previously presented in [109] and [156]. The components of those publications that are presented here are the work of the author of this thesis unless otherwise stated.

The primary motivation for using ellipsoids in SLAM, herein referred to as Ellipsoid-SLAM, is the suitability of ellipsoids for obstacle representation in trajectory planning algorithms and the ability to quickly generate the objects. This section describes how ellipsoids are used for modelling, and obstacle representation as well as how they can be used in SLAM. Simulated test cases demonstrate the
approach, before assessing performance with real-world data. These tests provide a thorough evaluation of Ellipsoid-SLAM, allowing the limitations to be elaborated, and an overall assessment made.

### 3.1.1.1 Ellipsoid Modelling

Ellipsoids are fit around a point cloud of an object by first finding the centroid, then by using principal component analysis (PCA) to find the orientation and size of the ellipsoid axes. The size of the axes is set by the PCA variance, $\sigma^{2}$, along each axis, which is used to compute a $3 \sigma$ value as the axis size. The resulting 3D body is represented by the centroid, $\boldsymbol{x}_{c}$, the orientation, $\mathbf{q}$ and the axes sizes, [ $a_{1}, a_{2}, a_{3}$ ], as depicted in Fig. 3.1. A significant limitation of this approach is that the $3 \sigma$ ellipsoid from PCA is not assured to capture all points. This property allows outliers from the segmentation to be rejected; however there can also be points remaining outside of the 3D model, as shown in Fig. 3.2 with an example extraction from a point cloud. Further inflating the ellipsoid could capture these points, but would also occupy more free-space, giving a highly conservative representation.


Figure 3.1. Ellipsiod model described by centroid, $\boldsymbol{x}_{c}$, orientation with respect to the global frame, and axes sizes, $a_{1}, a_{2}, a_{3}$


Figure 3.2. Example of ellipsoids being extracted from a segment of a point cloud. (a) Full point cloud, with one segment highlighted and the corresponding ellipsoid. (b) The segment of the point cloud and ellipsoid in isolation.

### 3.1.1.2 SLAM with Ellipsoids

For data association, the centroid, size and orientation are used as descriptors to match different ellipsoids. Once matched, localisation is performed by using both the centroid and quaternion orientation. The error between an observation and a stored ellipsoid (i.e. the errors use to update the state estimate in SLAM) is both a translational error: the distance between centroids, and an orientation error: the angular error between orientations. By using the orientation, the 3D rigid body ellipsoid features provide more information for the SLAM process than point features.

An Unscented Kalman Filter (UKF) is used to fuse the observations and estimate the state. The UKF is chosen to handle the non-linearities in both the process model for the attitude dynamics and the observation model for computing the orientation error of the ellipsoids. The orientation errors of the ellipsoids is computed with quaternions in the global frame by evaluating the quaternion difference between the observed orientation, $\hat{\mathbf{q}}_{c}$, and the stored orientation, $\mathbf{q}_{c}$. This quaternion error, $\mathbf{q}_{e}$ is converted to rotation vector format, $\mathbf{v}_{e}$, to be considered in the UKF update. The conversion to rotation vector format uses the quaternion logarithm (see Appendix C for details):

$$
\begin{align*}
& \mathbf{q}_{e}=\mathbf{q}_{c}^{-1} \otimes \hat{\mathbf{q}}_{c}  \tag{3.1}\\
& \mathbf{v}_{e}=2 \ln \left(\mathbf{q}_{e}\right) \tag{3.2}
\end{align*}
$$

An approach outlined in [108] is then used to average the attitude from the unscented sample points after the UKF update step. See [222] for more details. Through this update, the orientation of an ellipsoid provides what is effectively another observation to fix the attitude of the robot.

### 3.1.1.3 Ellipsoid-SLAM Simulations

A set of simulations were run to isolate the concept of using ellipsoid features for SLAM. Observations are made directly of ellipsoids, with added noise. Two such test cases are shown in Fig. 3.3.a (Small test case) and Fig. 3.4 (Large test case). SLAM is performed with no prior knowledge of the environment, and the robot travels on a pre-set trajectory through a simulated environment of ellipsoids. Zero-mean random noise was added to the observations with maximum deviations of 5 cm in position and axes magnitudes as well as 0.05 radians in attitude. The Small test case, in Fig. 3.3.a, uses a polynomial trajectory as the path of the robot, along with a regular yawing motion back and forth. A double integrator, constant velocity motion model is used to propagate dynamics in the update step of the UKF. The Large test case, in Fig. 3.3, uses a trajectory of a hand-carried camera to give simulated force and torque commands to the double integrator motion model. While the trajectory is hand carried, the map of ellipsoids and observations are simulated. A top-down view of the Large test case is shown in Fig. 3.5 with the orientation of the camera view direction displayed.

The SLAM paths generally tracks the true trajectory well, as presented in Table 3.1 with the Root Mean Square (RMS) and maximum errors.

The simulations show the approach can successfully track the state of the robot through a 3D trajectory, but with sensitivity to noise decreasing accuracy, and an accumulation of drift towards the end of a trajectory. Mapping is also successfully demonstrated but with substantial drifts in the


FIgURE 3.3. Ellipsoid SLAM simulated examples. (a) Small simulated Ellipsoid-SLAM test case. The blue path is the true path, and red is the tracked path. Blue ellipsoids are the true map, and black ellipsoids are the mapped estimates. A polynomial trajectory is used with regular yawing of the camera back and forth. (b) Trajectory planning around the obstacles (grey ellipsoids) that were mapped in the SLAM process. The black rectangle is the bounds in which to consider ellipsoids. In both, the trajectory progresses from the bottom of the image to the top.

TABLE 3.1. Tracking and mapping errors for simulated Ellipsoid SLAM examples.

|  |  | Small test case |  | Large test case |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | Parameter | RMSE | Max Error | RMSE | Max Eror |
| Robot State | Position $(\mathrm{m})^{*}$ | 0.044 | 0.090 | 0.054 | 0.205 |
|  | Velocity $(\mathrm{m} / \mathrm{s})^{*}$ | 0.011 | 0.051 | 0.066 | 0.272 |
|  | Attitude $(\mathrm{deg})^{+}$ | 0.980 | 4.930 | 3.070 | 15.977 |
|  | Angular Velocity $(\mathrm{deg} / \mathrm{s})^{*}$ | 1.187 | 2.2922 | 1.270 | 7.638 |
| Features | Position $(\mathrm{m})^{*}$ | 0.058 | 0.103 | 0.071 | 0.184 |
|  | Orientation $(\mathrm{deg})^{+}$ | 1.422 | 1.680 | 3.039 | 11.647 |
|  | Axes Magnitude $(\mathrm{m})^{*}$ | 0.003 | 0.008 | 0.003 | 0.012 |

*Position, velocity and angular velocity errors are taken as the magnitude of the associated 3 D vectors.
${ }^{+}$The attitude and orientation error is taken as the rotation magnitude for the quaternion representing the rotation from the true attitude to the estimated attitude
positions of the ellipsoids. These errors in ellipsoid positions can be problematic when using the map for trajectory planning.

### 3.1.1.4 Trajectory Planning with Ellipsoid Objects

Ellipsoids are the ideal representation for trajectory planning, as elaborated in Chapter 4, allowing trajectory optimisation algorithms to use the map of ellipsoids directly. To limit the number of ellipsoids


Figure 3.4. Large simulated Ellipsoid-SLAM test case. The blue path is the true path, and red is the tracked path. Blue ellipsoids are the true map, black ellipsoids are the mapped estimates. The trajectory is from a hand carried camera, but all observations are simulated as direct observations of ellipsoids.


Figure 3.5. Top-down view of Large simulated Ellipsoid-SLAM test case. The blue path is the true path, and red is the tracked path, with the arrows indicating the view direction of the camera. Blue ellipsoids are the true map and black ellipsoids are the mapped estimates. The trajectory is from a hand carried camera, but all observations are simulated as direct observations of ellipsoids.
that need to be considered, a volume of operation is defined, from the starting and goal location. Only ellipsoids within that volume are considered in trajectory optimisation. An example of trajectory planning with ellipsoids is shown in Fig. 3.3.b.

### 3.1.1.5 Ellipsoid-SLAM with Real-World Data

The simulation results demonstrated above use direct observations of ellipsoids. A critical part of using 3D objects for SLAM, though, is accurately extracting objects from the point cloud scans that can reliably be matched across multiple observations. Therefore, point cloud observations were recorded with a real RGBD camera to assess SLAM integrated with feature extraction. A hand-carried Asus Xtion Pro RGBD camera was used to record a data set around the Texas A\&M Land Air and Space Robotics Lab. Ground truth was provided by a Vicon tracking system, which was also used to give simulated IMU information by differentiating the tracked position and attitude. The RGBD data was processed to first segment the image; then segments were extracted as regions of interest to be modelled as ellipsoids. These segmentation steps are work from the primary author in [109] but are depicted in Fig. 3.6 for context. The image processing, demonstrated on a video stream, is available at https://www. youtube.com/watch?v=jA-yeVXIbG4.


Figure 3.6. Image processing pipline to extract and model ellipsoid objects from RGBD images. Work by primary author in [109]

The link, https://www. youtube.com/watch?v=wH860TOjB7A, presents a video of Ellipsoid-SLAM running on the dataset. The UKF integrates only angular information from the simulated IMU data, i.e. no simulated accelerometer data is used. Additionally, only the centroid of the ellipsoids was used in the UKF state, as the inclusion of ellipsoid orientation led to inferior results, the reasons for which are discussed below. Fig. 3.7 shows the final trajectory and map. While the path returns correctly to the starting position, the error is substantial throughout the trajectory, especially at the corners. The tracking error is displayed in Fig. 3.8, where the mean error is 0.18 m in position and $7.1^{\circ}$ in attitude, despite the angular velocity information from the simulated IMU data. There are also many more ellipsoids than true physical objects, with multiple ellipsoids sometimes representing one object. The poor performance of the algorithm highlights some important limitations of using ellipsoids as 3D objects for SLAM.


FIGURE 3.7. Ellipsoid-SLAM trajectory and map with real data from a hand carried RGBD camera and simulated IMU information. True trajectory is in blue from a Vicon tracking system. SLAM trajectory is in red. The arrows indicate the view direction of the camera. The black ellipsoids are the mapped features. The trajectory starts at the green dot near $x=0, y=-1.7$ and travels anti-clockwise.


Figure 3.8. Tracking errors from the Ellipsoid-SLAM test on real data. The attitude error is the angular component of the quaternion difference between the true and estimated orientation.

### 3.1.1.6 Limitations with Ellipsoids for Localisation and Mapping

The main limitation with using ellipsoids for mapping and localisation is that when observing multiple faces of an object, only the surface is ever observed; hence the ellipsoids are only ever surface ellipsoids (e.g. see Fig. 3.9.a). When trying to match multiple observations of the same object, different parts of the surface are being compared; hence there should not be an expectation that either the centroid or the orientation is the same. Taken to the extreme, for a rectangular prism in Fig. 3.9.b-d the centroids and orientations of adjacent faces are completely different. The outcome is a low quality of localisation update and the generation of many more ellipsoids than there are physical objects. This inconsistency in ellipsoid orientation is what caused the tracking results to be inferior when orientation was included in the UKF state. These results, along with the poor representation accuracy are the main components that limit the effectiveness of using ellipsoids as 3 D objects for localisation and mapping. An insight from these investigations is that it is always the surface of 3 D objects that is observed; hence there should be a focus on 3D surface modelling methods.


Figure 3.9. Illustration of issues with Ellipsoid-SLAM. (a) An ellipsoid is only a surface ellipsoid (an ellipsoid representing just the points on the surface and not the full volume of an object), red is the true object, blue dots are the simulated observations and black is the modeled ellipsoid. (b)-(c) Possible observations of a prism, associated axes intersecting at the observed centroid. (d) A third view of the prism, showing the centroid and axes from (b) and (c) are very different, despite being from the same object.

### 3.1.1.7 Assessment - Ellipsoids for SLAM and Trajectory Planning

Ellipsoids provide an ideal 3D representation of obstacles for trajectory planning, can be quickly modelled from a point cloud observation and can be effectively used as landmarks in a SLAM algorithm. However, ellipsoids do not provide a consistent representation of observed 3D objects, as observations are always of the surface, hence do not capture the full shape of an object. The inconsistency in representation means that the ellipsoids are not reliable features for SLAM. This limitation makes ellipsoids unsuitable as a representation for combining SLAM and 3D mapping of obstacles.

### 3.1.2 Gaussian Process Implicit Surfaces - Assessment of Potential for SLAM and Trajectory Planning

Attribution: A majority of this section was previously presented in [156] and is the work of the author of this thesis.

An approach that does model surfaces of objects, and can make inferences about the unseen part of a 3D object, is Gaussian Process Implicit Surfaces (GPIS). GPIS is a technique that utilises Gaussian Processes[191] to model the surfaces of 3D objects. Refer to Williams[226] and Dragiev[51] for detailed overviews of GPIS. In this section, the theory behind GPIS is summarised to provide the context to assess the potential of GPIS for use in SLAM, 3D mapping and obstacle representation.

### 3.1.2.1 Surface Modelling with GPIS

For 3D objects, the goal of GPIS is to represent the object with an implicit surface: a level set of a three dimensional function, $f(x, y, z)$. The function is modeled by a Gaussian Process and represents where an $(x, y, z)$ location is with respect to the object:

$$
f(x, y, z) \begin{cases}>0, & \text { inside object }  \tag{3.3}\\ =0, & \text { on surface } \\ <0, & \text { outside object }\end{cases}
$$

The process for GPIS is depicted in Fig. 3.10, where a series of observations are made of a simplified object, and are assigned $f(x, y, z)=0$ (Fig. 3.10.a). Next an internal control point is created in a location known to be inside the object and is assigned $f(x, y, z)=1$. A spread of external control points are then placed around the object, with $f(x, y, z)=-1$ (Fig. 3.10.b). The intention is to fit the function to the set of observation points, external control points and the internal control point, using Gaussian Processes. The level set at $f(x, y, z)=0$ will represent the fit to the surface of the 3 D object (Fig. 3.10.c). Care needs to be taken in ensuring the placement of the internal and external control points are correctly inside and outside the object.

The main step of the Gaussian Processes fit, is to evaluate a covariance function for the pairing of each point where we want to query the value of the resulting function. A kernel function is used to compute the covariance. The selection for the kernel function is important in controlling the nature of the resulting function. Refer to Gerardo-Castro[77] for some examples on different kernel functions, their use and their importance. One example kernel function is[226]:

$$
\begin{equation*}
k\left(x_{1}, x_{2}\right)=2\left\|x_{2}-x_{1}\right\|^{3}+3 \kappa\left\|x_{2}-x_{1}\right\|^{2}+\kappa^{3} \tag{3.4}
\end{equation*}
$$

where $x_{2}$ and $x_{1}$ are two of the query points, and $\kappa$ is a tuning parameter, set to be near the largest value of $\left\|x_{2}-x_{1}\right\|$, the two-norm of the vector difference [226]. The kernels are used to compute the covariance between every pair of points.


Figure 3.10. GPIS surface generation for a sphere. The true sphere is in black, and observations are the red triangles. The internal control point is black and the external control points are blue. (a) Observations of the object, and the internal control point. (b) Setting of external control points around the observations. (c) Extracted implicit surface from GP fit, as the red mesh. [156]

If we want to get a good representation of the surface in the level set, we need to have a sufficient number of the query points. For example, in Fig. 3.10 a grid of 10 by 10 by 10 query points is used to generate the surface. The kernel function (Eqn. 3.4) needs to be evaluated for every pairing of a combined set of 1000 query points, plus the observation points and control points to give the upperdiagonal of a covariance matrix $\Sigma$. The example in Fig. 3.10 has 50 observation points, 52 external control points, and 1 internal control point, making $\Sigma$ of dimensions $1103 \times 1103$. The output of the Gaussian Process fit is the predicted functional values at the query points, $f_{\text {pred }}$, and the associated covariance, $\hat{\Sigma}_{\text {pred }}$ :

$$
\begin{align*}
f_{\text {pred }}(u) & =\Sigma_{u x}^{T} \Sigma_{x x}^{-1}[x]  \tag{3.5}\\
\hat{\Sigma}_{\text {pred }} & =\Sigma_{u u}-\Sigma_{u x}^{T} \Sigma_{x x}^{-1} \Sigma_{u x} \tag{3.6}
\end{align*}
$$

Here $u$ represents the query points, and $x$ the set of observation points and control points. $\Sigma_{u u}, \Sigma_{u x}$, and $\Sigma_{x x}$ are components of the $\Sigma$ covariance matrix:

$$
\Sigma=\left[\begin{array}{ll}
\Sigma_{x x} & \Sigma_{u x}^{T}  \tag{3.7}\\
\Sigma_{u x} & \Sigma_{u u}
\end{array}\right]
$$

The implicit surface is then extracted from the level set of the query points and $f_{\text {pred }}$ functional values ${ }^{1}$, and is shown in Fig. 3.10.c.

[^5]
### 3.1.2.2 Strengths and Limitations of GPIS

One of the strengths of GPIS over the ellipsoid method is that the algorithm produces a prediction on the object beyond what is observed, to give a full 3D object (Fig. 3.10.c). The prediction is beneficial as there will never be a single observation of the complete object and the predicted surface presents possible obstructions (with an uncertainty) for the trajectory planner.

Another strength of the GPIS method is that the final representation is probabilistic: a covariance $\hat{\Sigma}_{\text {pred }}$ is defined over the queried space. The probabilistic representation is useful for having an informed update of the object and providing an uncertainty measure to fuse into both the localisation updates and the trajectory planner. This probabilistic information is lost, though, if it is desired to have a compact representation of the surface that is useful as an obstacle (i.e. by storing the implicit surface vertices or fitting an ellipsoid to the vertices).

A useful trait of GPIS is that it can form surfaces around generic objects, enabling it to have an accurate representation of the object. An accurate object model is important for both localization, to match observations to a stored model, and to correctly represent an obstruction for the trajectory planner. The accuracy comes at a considerable computational cost though, with the number of kernel evaluations (Eqn. 3.4) requiring a substantial number of operations (108,356 evaluations in the example in Fig. 3.10). The time to perform the computations is prohibitively large, and in the current form renders the technique unsuitable for the intended purposes. Future developments could be made to increase the computational speed of the technique, such as by taking inspiration from the methods employed with GPOM to divide the computations into many sub-problems. These avenues of investigation are left as future work.

### 3.1.2.3 Assessment - GPIS for SLAM and Trajectory Planning

GPIS has the benefits of being able to make predictions about unknown space and provides a continuous representation of the environment. The algorithm incorporates uncertainty information as well as providing a measure of uncertainty for the output map: useful traits for both mapping and localisation. The challenge with GPIS though, is that there is a large computational load to generate surfaces, and there is not a compact method of storing information from the fitted surface. These traits make GPIS, in its current state, difficult to implement for the desired combination of SLAM and 3D mapping of obstacles.

### 3.1.3 Non-Uniform Rational B-Splines - Assessment of Potential for SLAM and Trajectory Planning

Another method to represent the surface of an object, which is quicker to generate than GPIS, is using Non-Uniform Rational B-Splines (NURBS). This section describes the theory behind NURBS for modelling and manipulating surfaces of 3D objects, before an assessment of the potential for the algorithm for use in SLAM and trajectory planning. The details of NURBS relevant to the discussion are presented. Refer to [181] for an in-depth description of NURBS.

### 3.1.3.1 NURBS Background Theory

The main components of NURBS for modelling of surfaces are:
Parametric Mesh This is a set of 2D parametric values ( $s, t$ ) ranging from 0 to 1 , as depicted in Fig. 3.11. The number of points in the mesh $\left(m_{s} \times m_{t}\right)$ is the number of points that will be used for fitting a surface.
Control Points These points, $\boldsymbol{\rho}$, define the shape of the surface, and are ordered in a grid similar to Fig. 3.11, with $n_{s}$ and $n_{t}$ control points in each direction of the grid. The locations of these points affect the shape of the surface in a neighbouring area. Each control point has an associated weighting with it, scaling how influential it is on the surface.
Knot Vectors The set of knots control the locations where polynomial segments merge together in each parametric direction. The distribution of these knots affects where the surface can accurately capture large curvature. The degree of the polynomial segments, $p$, and the number of control points affects the number of knots: $n_{s}+p+1$, and $n_{t}+p+1$ for each parametric direction.


FIGURE 3.11. Parametric mesh required for NURBS input data points and control points. In this case $m_{s}=10$ and $m_{t}=15$.

NURBS curves are the basis from which NURBS surfaces are built. A NURBS curve is defined with a single parametric value, $s$, with each point defined by:

$$
\begin{equation*}
C(s)=\frac{\sum_{i=0}^{n_{s}-1} w_{i} \boldsymbol{\rho}_{i} B_{i}^{p}(s)}{\sum_{i=0}^{n_{s}-1} w_{i} B_{i}^{p}(s)} \tag{3.8}
\end{equation*}
$$

Here, $i$ denotes the indices stepping along control points, $\left(\boldsymbol{\rho}_{i}\right)$ in the parametric direction, $s$. The weighting for each control point is $w_{i}$, and the $B_{i}^{p}$ terms are the basis blending functions of degree $p$. These basis functions are polynomials that are defined for each $s$ value using the knot vector $\bar{u}$ : $u_{i}, \quad i=0, \ldots, n_{s}+p$ as below:

$$
\begin{align*}
& B_{i}^{p}(s)=\frac{s-u_{i}}{u_{i+p}-u_{i}} B_{i}^{p-1}(s)+\frac{u_{i+p+1}-s}{u_{i+p+1}-u_{i+1}} B_{i+1}^{p-1}(s)  \tag{3.9}\\
& B_{i}^{0}(s)= \begin{cases}1, & \text { if } u_{i} \leq s \leq u_{i+1} \\
0, & \text { otherwise }\end{cases} \tag{3.10}
\end{align*}
$$

For NURBS surfaces, this concept is expanded to two parametric directions, $s$ and $t$ and is given by:

$$
\begin{equation*}
S(s, t)=\frac{\sum_{i=0}^{n_{s}-1} \sum_{j=0}^{n_{t}-1} w_{i j} \boldsymbol{\rho}_{i j} B_{i}^{p}(s) B_{j}^{p}(t)}{\sum_{i=0}^{n_{s}-1} \sum_{j=0}^{n_{t}-1} w_{i j} B_{i}^{p}(s) B_{j}^{p}(t)} \tag{3.11}
\end{equation*}
$$

where $j$ is the index for the second parametric direction. There are $n_{s}$ control points in the $s$ parametric direction and $n_{t}$ in the $t$ parametric direction. The basis functions $B_{j}^{p}(t)$ are defined as in Eqn. 3.9, but evaluate a given $t$ parameter and use the knot vector for the $t$ parametric direction, $\bar{v}$.

A surface is entirely defined by the control points, weighting and knot vectors. To evaluate the points on a surface, a set of parametric coordinates, in $s$ and $t$ are selected and used in Eqn. 3.11. This evaluation is performed point by point, where the process starts with determining the knot span in which the parametric point lies. The knot span informs which control points need to be considered. Next, the basis-blending functions in the vicinity are evaluated (only a small set of local blending functions are non-zero). Finally, Eqn. 3.11 is used to get the point on the NURBS surface. Refer to [181] for more details on these steps. By evaluating a set of points in a parametric mesh, a whole surface can be represented. This surface evaluation method is a strength of NURBS: the underlying representation can be sampled at any resolution. The sampling could be at a low resolution for trajectory planning and a high resolution for use in localisation.

### 3.1.3.2 NURBS Surface Fitting

The goal for fitting NURBS surfaces is to solve the system of equations in Eqn. 3.11 to determine the $n_{s} \times n_{t}$ control points, $\boldsymbol{\rho}_{i j}$, to best fit the $m_{s} \times m_{t}$ data points $\mathbf{D}_{k l}$. The approach taken to do the fit employs a succession of NURBS curve fits, as outlined in [181]. The main steps, as implemented in this work will be outlined here. The first step for surface fitting is to ensure the data is organised in a parametric mesh (Fig. 3.11). Given data, $\mathbf{D}_{k l}$, the desired number of control points ( $n_{s}, n_{t}$ ) and desired degree ( $p$ ), the steps to fit a surface are [181]:

1. Determine the parametric values to assign to each data mesh-point based on the average spacing between the points.
2. Determine the knot vectors, based on the data parametric values (to have at least one data point between each knot).
3. Perform least-squares fits of NURBS curves to rows of data.
4. Perform least-squares fits of NURBS curves to columns of control points from the curves computed in step 3.

The results is a mesh of control points and two knot vectors that define the NURBS surface. For simplicity, uniform weighting is used, at a value of 1 . First, the process for curve fitting will be explained, to give the foundation for surface fitting.

## Curve Fitting

In solving the system of equations, there are two more unknowns to add to the control points, the parameter vectors, $\bar{s}, \bar{t}$, and the knot vectors, $\bar{u}, \bar{v}$, which are required to compute the basis functions in Eqn. 3.8. The parameter vectors and knot vectors need to be determined first, based on the given data.

For a curve, the discrete set of parameters, $\bar{s}=\left[s_{0}, s_{1}, \ldots, s_{m_{s}-1}\right]$, for $s_{k} \in[0,1]$ are determined to represent the parametric coordinates for each of the $m_{s}$ data points, $\mathbf{D}_{k}$, that are being fit. This parameter set could be computed to be uniform, with equal spacing, yet by computing the spacing with respect to the cumulative chord length of the data, the parameters can give a more accurate representation. Setting first $s_{0}=0$ and $s_{m_{s}-1}=1$, each internal parameter is computed with:

$$
\begin{equation*}
s_{k}=s_{k-1}+\frac{\left|\mathbf{D}_{k}-\mathbf{D}_{k-1}\right|}{d} \quad k=1, \cdots, m_{s}-2 \tag{3.12}
\end{equation*}
$$

where $d$ is the total chord length:

$$
\begin{equation*}
d=\sum_{k=1}^{m_{s}-1}\left|\mathbf{D}_{k}-\mathbf{D}_{k-1}\right| \tag{3.13}
\end{equation*}
$$

The next step is to compute the knot vector, $\bar{u}$, with the desire to have every knot span (the parametric span between adjacent knot values) to include at least one parametric value, to ensure the curve fitting problem is not singular. The number of control points desired, $n_{s}$, and the degree of curves, $p$, need to be selected here, as there are $n_{s}+p+1$ values in the knot vector. First, values of 0 and 1 are repeated $p+1$ times at the starts and ends, respectively.

$$
\begin{gather*}
u_{0}=u_{1}=\cdots=u_{p}=0  \tag{3.14}\\
u_{n_{s}-p-1}=u_{n_{s}-p}=\cdots=u_{n_{s}-1}=1 \tag{3.15}
\end{gather*}
$$

A knot repeated $p+1$ times is referred to as having multiplicity $p+1$. Having $p+1$ multiplicity at the start and the ends forces the curve ends to match the end control points. The internal control points, from $p+1$ to $n_{s}-p-2$ are defined with consideration of $\bar{s}$ :

$$
\begin{align*}
\hat{d} & =\frac{m_{s}}{n_{s}-p}  \tag{3.16}\\
i & =\text { floor }(j \hat{d})  \tag{3.17}\\
\alpha & =j \hat{d}-i  \tag{3.18}\\
u_{p+j} & =(1-\alpha) s_{i-1}+\alpha s_{i} \quad j=1, \cdots, n_{s}-p-2 \tag{3.19}
\end{align*}
$$

For solving the least squares fit, the two end control points are first made equal to the end data points, ensuring the fitted curve will start and end at those points, $\boldsymbol{\rho}_{0}=\mathbf{D}_{0}, \boldsymbol{\rho}_{n}=\mathbf{D}_{n}$. For the remaining internal control and data points, Eq. 3.8 is rearranged into matrix form:

$$
\begin{equation*}
\left(\boldsymbol{B}^{T} \boldsymbol{B}\right) \boldsymbol{\rho}=\boldsymbol{Y} \tag{3.20}
\end{equation*}
$$

where $\boldsymbol{B}$ is a matrix of the NURBS basis functions, evaluated at the different parametric values:

$$
\boldsymbol{B}=\left[\begin{array}{ccc}
B_{1, p}\left(s_{1}\right) & \cdots & B_{n_{s}-2, p}\left(s_{1}\right)  \tag{3.21}\\
\vdots & \ddots & \vdots \\
B_{1, p}\left(s_{m_{s}-2}\right) & \cdots & B_{n_{s}-2, p}\left(s_{m_{s}-2}\right)
\end{array}\right]
$$

The matrix has dimensions $\left(m_{s}-2\right) \times\left(n_{s}-2\right)$, i.e. the length of the internal data in the first dimensions and the internal control points in the second dimension. The matrix $\boldsymbol{Y}$ is a grouping of the weighted data points, each adjusted by the end points:

$$
\begin{gather*}
\boldsymbol{Y}_{k}=\mathbf{D}_{k}-B_{0, p}\left(s_{k}\right) \mathbf{D}_{0}-B_{n_{s}-1, p}\left(s_{k}\right) \mathbf{D}_{m_{s}-1} \quad k=1, \ldots, m_{s}-2  \tag{3.22}\\
\boldsymbol{Y}=\left[\begin{array}{c}
B_{1, p}\left(s_{1}\right) \boldsymbol{Y}_{1}+\cdots+B_{1, p}\left(s_{m_{s}-2}\right) \boldsymbol{Y}_{m_{s}-2} \\
\vdots \\
B_{n_{s}-2, p}\left(s_{1}\right) \boldsymbol{Y}_{1}+\cdots+B_{n_{s}-2, p}\left(s_{m_{s}-2}\right) \boldsymbol{Y}_{m_{s}-2}
\end{array}\right] \tag{3.23}
\end{gather*}
$$

The matrix $\rho$ groups the internal control points and is the result of solving the system of equations presented in Eqn. 3.20. Given that $\boldsymbol{B}^{T} \boldsymbol{B}$ is square, this can be solved by taking the inverse of that matrix (although more advanced solution methods, taking advantage of the sparsity of the matrix could be used):

$$
\begin{equation*}
\boldsymbol{\rho}=\left(\boldsymbol{B}^{T} \boldsymbol{B}\right)^{-1} \boldsymbol{Y} \tag{3.24}
\end{equation*}
$$



FIgURE 3.12. NURBS curve fitting examples. A NURBS curve is fit to data points with $p=3$ and $n_{s}=5$. The control polygon is defined by straight lines between control points.

With the control points and knot vectors, the curve is then fully defined. Some example curve fittings are shown in Fig. 3.12.

## Surface Fitting

For surfaces, the challenge is in extending the curve fitting problem into a second parametric direction. For this, the data points, $\mathbf{D}_{k l}$, need to be ordered in a 2D grid (see Fig. 3.11), so when an index, $k$ or $l$, is increased, there is a continuous and logical progression along the surface. The result of such an organisation is that a mesh plotted for $\mathbf{D}_{k l}$ would represent a grid pattern, such as depicted in Fig. 3.13.

A parameter vector and a knot vector is computed for each parametric direction, $s$ and $t$, corresponding to the index directions $k$ and $l$, respectively. There are $m_{s}$ data points in the $s$ direction and $m_{t}$ data points in the $t$ direction. The number of control points needs to be selected for each dimension as well: $n_{s}$ and $n_{t}$, and the polynomial degrees, $p$.

The parameterisation for surfaces works in a similar manner to that of curves, using Eqns. 3.12 and 3.13. Each column of data (a line along $s$ with constant $t$ ), is treated as a curve to compute the parameterisation, then the average taken across all columns to give the $\bar{s}$ vector, of length $m_{s}$. The same is then done for the rows of the data (a line along $t$ with constant $s$ ) to give the second dimension of parameterisation, $\bar{t}$, of length $m_{t}$.

With the parameter vectors computed, the knot vectors are then determined in exactly the same way as for curves, using Eqn. 3.19 for each each parametric direction. In the $s$ direction, $\bar{u}$, of length $n_{s}+p+1$ is computed using $\bar{s}$ and in the $t$ direction, $\bar{v}$, of length $n_{t}+p+1$ is computed using $\bar{t}$. The implication in this process is that the parametric spacing and knot spacing is the same for every column $(\bar{s}, \bar{u})$ and likewise the same for every row $(\bar{t}, \bar{v})$.


Figure 3.13. Mesh requirements for NURBS surface fitting. (a) The 3D surface to be measured, which could be sampled in any way. (b) Observations of the surface that are organised in a structured mesh, which is what is needed for NURBS surface fitting.

When performing the fit of the surface, the process is followed as outlined below:

1. For each $i$ th row of the $m_{s}$ rows of data, fit a curve through $\mathbf{D}_{i l}, l=0, \ldots, m_{t}-1$ to get temporary control points $\mathbf{T}_{i, j}, j=0, \ldots, n_{t}-1$
2. For each $j$ th column of the $n_{t}$ columns of temporary control points, fit a curve through
$\mathbf{T}_{k, j}, \quad k=0, \ldots, m_{s}-1$ to get $\boldsymbol{\rho}_{i, j}, \quad i=1, \ldots, n_{s}-1$

Each of the curve fitting steps is the same process as outlined for the curves above. For the row curve fits, $\bar{v}$ and $\bar{t}$ are used, and for the column fits, $\bar{u}$ and $\bar{s}$ are used. The resulting matrix of $n_{s} \times n_{t}$ control points, $\boldsymbol{\rho}$, has each of the corner points matched to the corners of the data points. In the implementation of the process outlined above, the computation of the $\boldsymbol{B}$ matrices and in particular, $\left(\boldsymbol{B}^{T} \boldsymbol{B}\right)^{-1}$ only needs to be performed once for the rows, and once for the columns, as it is a function of only the knot vectors and parameter vectors. This fact allows the surface fitting procedure to be rapidly processed. An example of surface fitting is shown in Fig. 3.14). The number of control points and polynomial degree are the parameters that can be tuned to balance the accuracy and speed of computation for the surface fitting.

### 3.1.3.3 NURBS Surface Manipulation

There is a range of methods to manipulate surfaces once they have been generated. One that can be of particular use is knot insertion. This process involves adding extra knots (and hence control points) without changing the shape of the surface. Having extra control points can be of use to be able to manipulate a surface by moving the control point. Refer to [181] for details on knot insertion.


Figure 3.14. NURBS surface fitting example. (a) Data points of an observation of a surface. (b) The data points organised into a structured mesh. (c) A NURBS surface fit to the data, plotted with the data. (d) The NURBS surface in isolation showing contours at evenly spaced parametric values.

### 3.1.3.4 Assessment - NURBS for SLAM and Trajectory Planning

NURBS surfaces present the middle-ground between the quick, simple but inaccurate ellipsoids and the slow, complex and accurate GPIS. NURBS surfaces can be fitted to the data quickly, and the resulting representation is compact: only the control points and knot vectors. The accuracy depends on the number of control points used, enabling both high and low detail, depending on the needs of the representation. Higher detail comes with higher computation time, but the flexibility enables the balance between accuracy and computation time to be tuned. Another benefit for NURBS is the ability to sample the surface at multiple resolutions: a strength when using the representation for both SLAM and trajectory planning.

### 3.1.4 Selected 3D Object Representation

A summary of the pros and cons of the three different 3D modelling algorithms is presented in Table 3.2. From the review, NURBS is selected for further investigation, providing the best trade-off between computational speed and accuracy, with the flexibility to adapt the resolution for the given application. NURBS surfaces have not been previously applied to SLAM or trajectory planning, only with NURBS curves [225], or other forms of spline curves [137, 221]. The next section describes how NURBS are used for SLAM and trajectory planning.

TABLE 3.2. Summary of strengths and limitations of 3D modelling algorithms for SLAM and trajectory planning.

| Method | Strengths | Limitations |
| :--- | :--- | :--- |
| Ellipsoids | - Fast computation | - Inaccurate representation |
|  | - Ideal obstacle representation | - Inaccuracy for localization |
|  | - Simple representation for | and update |
|  | storage |  |
| GPIS | - Accurate representation | - Very computationally expensive |
|  | - Models into unknown space | - Need a different representation |
|  | - Probabilistic representation | for storage and for |
|  |  | trajectory planning |
| NURBS | - Fast computation | - Limited accuracy |
|  | - Strong update capability | in representation |
|  | - Relatively compact | - Slower computations for |
|  | representation for storage | trajectory planning |

### 3.2 NURBSLAM: Using NURBS Surfaces for Localisation, Mapping and Trajectory Planning

For use in autonomous navigation, a NURBS object representation needs to be useful in the following tasks:

## 1. Data Association:

- To have a sufficient number of descriptors to match an observation to the appropriate object.

2. Mapping:

- Generating a surface from point cloud observations.
- Updating and extending a surface with new observations.

3. Localisation:

- Matching observations to a previously mapped object to give information on the robot's pose.

4. Trajectory Planning:

- Providing an efficient way to determine the minimum distance from a point to the object and the gradient of that distance for obstacle representation in a trajectory planning algorithm.

This section outlines how NURBS are applied to each of these tasks and is a presentation of NURBSLAM (Non-Uniform Rational B-Spline Localisation And Mapping), a new algorithm that is a contribution of this thesis.

### 3.2.1 Data Association

The task for data association is to match an observation of an object (a point cloud) to the correct existing NURBS object stored in the map. A NURBS object is defined as a NURBS surface that has been generated from observations and is stored in a global reference frame. The map is defined by a set of NURBS objects, to which new observations are matched. One of the advantages of using 3D objects as features for SLAM is that the data association challenge becomes simpler because there are fewer features that are more spread out. These characteristics mean that using only a centroid of the NURBS object gives an effective data association. The centroid of the data can be computed by taking the average of a down-sampled set of the observed points. Additionally, a representative centroid of the NURBS object can be computed by taking an average of the control points. Because observations are only ever on the surface, it is unlikely there will be an exact alignment of centroids; hence high thresholds are used to select candidate matches and the closest object selected as the correct match.

If the number of objects does become large, when operating in a large environment, then there is the potential to use more information to describe an object, such as the colour, a visual or 3D appearance descriptor, or surface texture metrics from the field of Geographic Information Systems, such as in [109].

### 3.2.2 Mapping - Object Generation

Mapping is the process of using point cloud observations to generate 3D models of the environment. This process assumes that the input is a segment from a point cloud observation that is either flagged
as a new object, or a new observation of an existing object, from the data association step. The steps required for mapping are:

1. Initial generation of an object.
2. Extension and update of an existing object.

The generation of an object follows the NURBS surface fitting steps outlined in Section 3.1.3. One crucial requirement for surface fitting is that the observation data needs to be organised in a structured mesh. This requirement is non-trivial and requires careful consideration. Although a point cloud observation from an RGBD camera or stereo camera pair is organised thanks to the grid of the image, the segmentation of that point cloud for a given object is unlikely to give an equal number of points along each row or column.

The process taken to provide observations in a structured mesh is to utilise the structure from the initial observation: the grid of the image. First, the assumption is that a segmented point cloud for a given object is provided, with all pixels that are not part of the object set with values as NaN (Not A Number). An approach is then taken to iteratively remove rows and columns of a scan until a rectangular mesh is left that contains a large percentage of points from the object (e.g. $90 \%$ ). An average of neighbouring points replaces the remaining NaN points. The process of removing rows and averaging NaN points is outlined in Algorithm 1. The first loop involves removing rows, and then columns, from the mesh that have more NaN values than a threshold. That threshold changes with the size of the mesh to be equal to the largest number of NaNs in any row or column minus a user-set buffer. Once there are few enough NaNs , or a lower limit on the mesh size, the first loop exits. The second loop averages all the remaining NaN values by taking the mean of the points in adjacent parametric coordinates. This averaging process is in a loop in case all the neighbours of a NaN point are also NaN . In Algorithm 1, the parameter buffer controls the update of the threshold thresh to determine which rows and columns to remove in each iteration. A higher buffer will more aggressively remove rows and columns, leading to fewer iterations, but a potentially higher loss of good data.

Following the removal of NaNs from the mesh, the number of rows and columns are reduced, if required, to give the desired mesh dimensions, $m_{s} \times m_{t}$. The mesh is reduced by taking $m_{s}$ evenly distributed rows and $m_{t}$ evenly distributed columns of the desired number. The output is an organised mesh containing entirely valid 3D points. An example of the mesh generation from a point cloud is presented in Fig. 3.15.

One limitation of this mesh reduction approach is the need to do averaging for the NaN points that remain after the row and column removal. This averaging introduces artefacts into the data that are not present in the observations, which can lead to irregularities in the surface generation. If the percentage of points to average is low, these issues have minimal impact on performance.

```
Algorithm 1 Mesh Processing
    mesh \(\leftarrow\) segmented scan in structured mesh
    \(n\) Rows \(\leftarrow\) NumberOfRows(mesh)
    \(n\) Cols \(\leftarrow\) NumberOfColumns \((m e s h)\)
    \(m_{s} \leftarrow\) desired number of rows
    \(m_{t} \leftarrow\) desired number of columns
    buffer \(\leftarrow\) setting for how aggresively to remove rows and columns
    \(N a N_{l i m} \leftarrow\) maximum allowable NaNs
    procedure PROcESSMESH
        mesh \(\leftarrow\) RemoveRowsAndColumns(mesh)
        \(m e s h \leftarrow\) AverageRemainingNaNs \((\) mesh \()\)
    end procedure
    procedure REMOVEROWSANDCOLUMNS
        while \(n R o w s>m_{s}\) and \(n C o l s>m_{t}\) and \(n N a N>N a N_{l i m}\) do
            thresh \(\leftarrow\) UpdateThresh (mesh,doRow)
            for row in mesh do
                        if NumberOfNaNs \((\) row \()>\) thresh then
                                mesh \(\leftarrow\) RemoveRowFromMesh(row,mesh)
                    end if
            end for
            thresh \(\leftarrow\) UpdateThresh \((\) mesh, doCol \()\)
            for col in mesh do
                    if NumberOfNaNs \((\) col \()>\) thresh then
                mesh \(\leftarrow\) RemoveColFromMesh(col,mesh)
                    end if
            end for
            \(n N a N \leftarrow\) NumberOfNans (mesh)
            \(n\) Rows \(\leftarrow\) NumberOfRows(mesh)
            \(n\) Cols \(\leftarrow\) NumberOfCols (mesh)
        end while
        function UpdateThresh (mesh, rowOrCol)
            if RowOrCol is doRow then
                thresh \(\leftarrow\) MaxNanInRows(mesh) - 1-buffer
            else if RowOrCol is doCol then
                thresh \(\leftarrow\) MaxNanInCols(mesh) - 1-buffer
            end if
        end function
    end procedure
    procedure AVERAGEREMAININGNANS
        while \(n N a N>0\) do
            for \(N a N_{p}\) point in mesh do
                mesh \(\left(N a N \_p o i n t\right) \leftarrow\) AverageFromNeighbours \(\left(N a N \_p o i n t\right)\)
            end for
            \(n N a N \leftarrow\) NumberOfNans \((\) mesh \()\)
        end while
    end procedure
```



Figure 3.15. Mesh generation example. A dense point cloud observation of an object is shown in red, that is not organised in a rectangular mesh. Using the mesh processing steps, this data is reduced to the rectangular mesh in black.

### 3.2.3 Mapping - Object Update

When a new observation is made of an existing surface, the surface is extended to grow the surface to include newly observed parts of the object. The methods for doing this will first be explained for NURBS curves, as it forms the basis for the explanation of NURBS surface extension. Curve extension consists of 1) determining where to split the observations between overlapping and new data, and 2) joining the new data to the existing curve.

### 3.2.3.1 Determining the Split Between Overlapping and New Data

Given an existing NURBS curve and new observations that overlap with this curve (Fig. 3.16.a), the goal is to update the existing curve to take into account the observation data. The first step here is determining the split of observation data between new points and overlapping points. This split is determined by finding the closest point on the observation data to each of the ends of the existing curve. The curve-end that has the smallest distance to the observation data is then selected as the end to be extended. The closest observation data point to this curve-end is selected as the split point. The overlapping data extends from the end data point that is closest to the curve, to the split point. The remaining observation points are the new data. These steps are depicted in Fig. 3.16.b.


Figure 3.16. Steps to use new data to extend a NURBS curve. (a) Existing curve and observation data. (b) Identificaiton of overlapping and new data, as well as the split point (last overlapping data point). (c) Fitting a curve to the new data. (d) Control points on the existing and the new curve, with knot insertion adding control points on the new curve. (e) Combined control points and resulting curve. (f) Final curve with original curve and observation data.

### 3.2.3.2 Joining New Data to a NURBS Curve

A NURBS curve is fit to the new data, with the number of control points being computed as the default number of control points multiplied by the percentage of the observation data classified as new (Fig. 3.16.c).

The next step is to combine the two curves, which involves joining control points and knot vectors. When joining the knots vectors, it is desired to maintain a multiplicity of one for each internal knot, i.e. there are no repeated knots except at the ends. Managing multiplicity in this way allows continuity to be maintained (refer to [181] for more discussion on knot multiplicity and continuity). Because each curve ends with $p+1$ repeated knots, there needs to be a removal of $2 p+1$ knots. This removal of knots also requires removal of $p-1$ control points. The priority is given to the existing curve in this knot
removal; hence knots and control points are removed from the new curve. Knot insertion precedes this knot removal to minimise the loss of the end part of the new curve. Knots are inserted at the end of the new curve that will be joined, with an additional $p$ knots and $p-1$ control points. An example of knot insertion is shown in Fig. 3.16.d.

When the existing curve (with knot vector $\bar{u}$ ), is extending from the $s=1$ parametric end, and the new curve knot vector $\bar{\mu}$ starts from the $s=0$ parametric end, the knot vector components taken are:

$$
\begin{align*}
\bar{u}_{\text {part }} & =\left[u_{0}, u_{1}, \cdots, u_{n-p-1}\right]\left(\frac{n_{s, 1}}{n_{s, 1}+n_{s, 2}}\right)  \tag{3.25}\\
\bar{\mu}_{\text {part }} & =\left[\mu_{p+1}, \cdots, \mu_{n-1}\right]\left(\frac{n_{s, 2}}{n_{s, 1}+n_{s, 2}}\right)+u_{n-p-1}\left(\frac{n_{s, 1}}{n_{s, 1}+n_{s, 2}}\right)  \tag{3.26}\\
\bar{u}_{\text {combined }} & =\left[\bar{u}_{\text {part }}, \bar{\mu}_{\text {part }}\right] \tag{3.27}
\end{align*}
$$

where $n_{s, 1}$ and $n_{s, 2}$ are the number of control points for the existing and new curve respectively. The right-most term in Eqn. 3.26 sets $\bar{\mu}_{\text {part }}$ to start with values higher than the last value in $\bar{u}_{\text {part }}$. The fraction that multiplies the vectors (the fraction with $n_{s, 1}$ and $n_{s, 2}$ ) scales the knot values so the combined vector, $\bar{u}_{\text {combined }}$ is monotonically increasing from 0 to 1 and so there is more parametric range given to the curve with the greater number of control points. The control points are combined in a similar manner (using $\zeta$ to denote the control points on the new curve):

$$
\begin{align*}
\boldsymbol{\rho}_{\text {part }} & =\left[\boldsymbol{\rho}_{0}, \cdots, \boldsymbol{\rho}_{n_{s, 1}-1}\right]  \tag{3.28}\\
\boldsymbol{\zeta}_{\text {part }} & =\left[\boldsymbol{\zeta}_{p-1}, \cdots, \zeta_{n_{s, 2}-1}\right]  \tag{3.29}\\
\boldsymbol{\rho}_{\text {combined }} & =\left[\boldsymbol{\rho}_{\text {part }}, \boldsymbol{\zeta}_{\text {part }}\right] \tag{3.30}
\end{align*}
$$

The example outlined above is one of four scenarios for combining the existing curve with the new curve. The existing curve can extend from either the $s=1$ parametric end, or the $s=0$ parametric end. These two variations affect which curve comes first in the combined knot vector and control point vector. Additionally, the new curve can be joined at either the $s=0$ or $s=1$ parametric end. These two variations, combined with the direction that the existing curve is extending, determine whether the parametric direction on the new curve needs to be flipped, $\bar{\mu}=1-\bar{\mu}$, or not. The steps in determining the split of data provide the information to select from the four permutations. Regardless of variation, $p$ knot vectors are removed from the existing curve at the end that is being extended. Then $p+1$ knots and $p-1$ control points are removed from the end of the new curve that is being joined. Finally, the steps to join the knot vectors are adjusted to ensure a monotonic increase from 0 to 1 . The combined control points and knot vectors define the combined curve (Fig. 3.16.e).

### 3.2.3.3 Steps for Surface Extension

The principles for extending and updating a surface are the same as for curves, but with the added complication of the extra parametric dimension. The process can be summarised as:

1. Determining the direction to extend the surface: left, right, up or down.
2. Determining the split of data between new points and overlapping points. This step also determines the direction the new data will join the surface.
3. Fitting a surface to the new data, with an appropriate number of control points.
4. Extending the existing surface row-curve by row-curve or column-curve by column-curve to the new surface (which could be regarded as sets of columns, or sets of rows) using the curve combination method described above.

### 3.2.3.4 Surface Split Determination

Given successful completion of steps 1 and 2 for surface-extension, steps 3 and 4 are simple extensions from the method with NURBS curves. Achieving steps 1 and 2 is challenging though, with difficulty in determining the appropriate directions to extend the surface and split the data over the range of all possible cases. In contrast to a curve, there are a large number of possible combinations to join two surfaces, with 16 different combinations for extension directions of the existing surface, and joining directions for the new data.

The steps to determine the split of data and extension direction for surfaces are:

1. Generate surface points from the stored NURBS object.
2. Classify observation points as new or overlapping.
3. Determine the join-edge for the observation data.
4. Determine the join-edge for the surface.
5. Compute the indices of new data to take from the observation data.

These steps are elaborated below. The first step is to generate a set of points from the stored surface (Eqn. 3.11). The second step is to classify observation points as either overlapping or new. Normalshooting correspondence estimation [200] is used to classify the points by matching every observation point to the closest map surface point. The normal-shooting correspondence returns the perpendicular distance of each observation point to the normal from the matching surface point. These distances are used with a threshold to classify observations points as either overlapping points, if the distance is below the threshold, or new points, if not. The threshold is the average of the distance between adjacent observation points, so when a distance exceeds this, it is likely to be past the edge of a surface. The output from this step is a boolean array, boolNewArray, indicating whether the point in each mesh coordinate is overlapping (false) or not (true).

The next step uses the boolean array to determine an initial split of the observation data. This split is denoted by a join-edge and a far-edge of the observation data that are both either rows or columns. The method to extract the far-edge on the observation data using boolNewArray is outlined in Algorithm 2, with steps depicted in Fig. 3.17.a-c. The ID of the far-edge denotes which of the four edges are the far-edge. The join-edge is the last row or column from the far-edge that is classified to be new.

```
Algorithm 2 Determining the far-edge of the observation data
    boolNewArray \(\leftarrow\) boolean array indicating new (true) or overlapping (false) observation points
    newCountVector \(\leftarrow\) vector storing the number of new rows or columns from each edge
    newDataFarEdgeID \(\leftarrow\) index of observation-data edge that is opposite the join-edge
    \(m_{s} \leftarrow\) number of rows in observation data
    \(m_{t} \leftarrow\) number of columns in observation data
    newCountBuffer \(\leftarrow\) number of overlapping points allowed to still be classified as a new row/col.
    procedure DeterminenewDataJoinEdge
        for edge in observationData do
            newRowColCount \(\leftarrow 0\)
            if EdgeIsRow(edge) then
                lim \(=m_{t}-\) newCountBuffer
            else
                lim \(=m_{s}-\) newCountBuffer
            end if
            while not exitFlag do
                boolNewVector \(\leftarrow\) GetRowOrCol(of boolNewArray, at (Index(edge) - newRowColCount))
                \(n N e w \leftarrow\) CountTrue(boolNewVector)
                if \(n N e w>l i m\) then
                    newRowColCount \(\leftarrow\) newRowColCount +1
                else
                    exitFlag \(\leftarrow\) true
                end if
            end while
            newCountVector \(\leftarrow\) Append newRowColCount to newCountVector
        end for
        newDataFarEdgeID \(\leftarrow\) IndexOfMaximum(newCountVector)
    end procedure
```

The join-edge of the surface is then determined, using the far-edge of the observation data. The mid-point of each edge of the surface is compared to the mid-point of the observation far-edge, as outlined in Algorithm 3. The comparison checks the angle, $\theta$, between a surface tangent to the surface edge mid-point, $v_{s e}$ and the vector from the surface mid-point to the observation far-edge mid-point, $v_{e d}$, as depicted in Fig. 3.17.d. The angle, $\theta$, needs to be less than $90^{\circ}\left(v_{e d} \cdot v_{s e}>0\right)$ to pass, indicating an appropriate extension direction. The closest edge that passes the criterion is the surface join-edge.

With the surface join-edge determined, the next step is to determine which observation data to use for extending the surface. A simple approach is to use the rectangle of observation data from the join-edge to the far-edge. This approach works well for rectangular surfaces, but when there are curves in edges, such as in Fig. 3.18.a, there will be a large gap with no data. Instead, the surface join-edge is used to compute a set of indices on the observation data, to be one boundary of the new data, with the observation far-edge being the opposite boundary. This process is outlined in Algorithm 4 for the case when surface rows are being extended into observation data rows. The algorithm changes slightly with the other three variations of taking surface rows or columns, with observation data rows or columns. Regardless of variation, the principle remains the same. The result is a rectangular mesh of data that occupies the new region of the observation data.

```
Algorithm 3 Determining the surface join-edge from which to extend the surface
    surface \(\leftarrow\) the existing surface to be extended
    farEdgeMid \(\leftarrow\) midpoint of new data edge that is opposite the joining edge
    \(v_{s e} \leftarrow\) vector along the surface to the midpoint of a surface edge
    \(v_{e d} \leftarrow\) vector from the midpoint of a surface edge to farEdgeMid
    edgeDistances \(\leftarrow\) vector to store distances from each surface edge
    surfaceJoinEdgeID \(\leftarrow\) index of the edge on the surface to be extended
    procedure DetermineSurfaceExtensionDirection
        for edge in surface do
            edgeMidPoint \(\leftarrow\) EdgeMidPointFromNURBS(surface,edge)
            \(v_{s e} \leftarrow\) VectorToEdgeAlongSurface(surface,edge)
            \(v_{\text {ed }} \leftarrow\) farEdgeMid - edgeMidPoint
            if \(v_{s e} \cdot v_{e d}<0\) then
                edgeDistances \(\leftarrow\) Append Inf to edgeDistances
            else
                edgeDistances \(\leftarrow\) Append \(\left|v_{e d}\right|\) to edgeDistances
            end if
        end for
        surfaceJoinEdgeID \(\leftarrow\) IndexOfMinimum (edgeDistances)
    end procedure
```

```
Algorithm 4 Extracting new data (for the case when rows are extending to rows)
    surface \(\leftarrow\) existing surface
    observationData \(\leftarrow\) mesh of observations
    \(m_{s} \leftarrow\) number of points to sample along the surface join edge
    newData \(\leftarrow\) rectangular mesh of new data for surface extension
    procedure ExtractNewDataRowsFromSurfaceEdge
        surfaceEdgePoints \(\leftarrow\) NPointsAlongNURBSEdge (surface, surfaceJoinEdgeID, \(m_{s}\) )
        dataJoinPoints \(\leftarrow\) NormalShooting(match surfaceEdgePoints to observationData)
        joinStartIndices \(\leftarrow\) GetIndicesFromPoints(dataJoinPoints)
        farEdgeIndices \(\leftarrow\) GetIndicesFromEdgeID(newDataFarEdgeID)
        \(n\) Points \(\leftarrow\) MinimumNumberOfIndicesBetween(joinStartIndices,farEdgeIndices)
        for \(i=0: m_{s}-1\) do
            tempRow \(\leftarrow\) observationData \((\) row \((i))\) from joinStartIndices to farEdgeIndices
            newData \((\) row \((i)) \leftarrow n\) Points from tempRow
        end for
    end procedure
```



Figure 3.17. Depiction of surface splitting steps. (a) Existing surface in blue and new observation points in black. (b) Identificaiton of one new row above, and three new columns to the right. (c) Extaction the new data join-edge in red, and the far-edge in blue. (d) Vector from one surface edge to the new data far-edge ( $v_{e d}$ ) and along the surface to the edge ( $v_{s e}$ ), used in the calculations to determine the surface join-edge.

### 3.2.3.5 Joining New Data to an Existing Surface

A surface is then fit to this new data, with the same number of control points along the parametric direction of the join-edge as for the existing surface join-edge. The number of control points along the other parametric direction is computed based on the number of data points and the default number of control points for a new object. For example, when extending surface rows to rows of new data, the number of control points $n_{t, 2}$ along the rows of the new surface are:

$$
\begin{equation*}
n_{t, 2}=\left(\frac{m_{t, 2}}{m_{t, d}}\right) n_{t, d} \tag{3.31}
\end{equation*}
$$

where the subscript 2 is for the new surface, and $d$ is for the default values for a newly observed object. When columns are used for the surface or data, then the $t$ subscript changes to the $s$ subscript for the corresponding terms. An example of fitting a surface to the extracted new data is shown in Fig. 3.18.b. There is new observation data that is excluded from the surface extension, but this exclusion enables the new surface to be efficiently joined to the existing surface.

The surfaces are joined by repeatedly performing the curve joining procedure to rows or columns of control points, which, with the knot vectors, are treated as NURBS curves. When rows are extending to rows, for instance, each row of the surface control points are treated as a curve that is joined to the corresponding row of control points on the new surface. An example result is shown in Fig. 3.18.c.

This procedure to extend surfaces has 16 variations, with two binary parameters added to the two from joining curves. These four parameters are:

1. Columns or rows to be added from the new data.
2. Columns or rows to be extended from the existing surface.
3. New data before old data (parametrically).
4. New data to reverse parameterisation or not.

The process to compute the observation data join-edge gives the first parameter. This parameter sets whether rows or columns are used for extracting the new data and joining the new surface. The second parameter is determined by the selection of the surface join-edge and sets all remaining steps with the existing surface to use either rows or columns. The surface join-edge also determines the third
$\qquad$
parameter: if the selected edge is towards the 0 end parametrically for the direction to be extended, then the new surface is placed before the existing surface; otherwise, the new surface is after the existing surface. The final parameter comes from a combination of the surface join-edge and the observation data join-edge, with 8 out of the 16 combinations requiring parameters to be reversed. The third and fourth parameters are used in the curve joining steps.


Figure 3.18. Example Surface extension. (a) Existing surface in blue and observation data as black circles. (b) Surface fit to new data in pink. (c) The result from joining the existing surface and the new surface.

### 3.2.4 Localisation

Using NURBS surfaces as the features for localisation provides more information than single 3D points. This extra information is used by matching observations to surfaces. The surfaces are matched by generating sampling points on the existing object and using methods similar to ICP to align the new observations with the map object and compute the resulting transformation. For this process, the suite of tools available in the Point Cloud Library (PCL) [200] is utilised. The alignment of surfaces for each object returns the associated transformation, including both linear and angular components. The set of transformations from the observed objects are used as observations to include in an Extended Kalman Filter (EKF) that fuses observations to update the state estimate. The localisation process assumes that data association has already been performed to match the observations to the appropriate map object.

### 3.2.4.1 3D Feature Descriptors for Point Correspondences

It is desired to use the shape of the object for aligning surfaces. Therefore, surface normals and 3D features descriptors are extracted and used for finding correspondences and performing alignment. 3D feature descriptors are similar in concept to visual feature descriptors, as described in Section 2.1.4, and often the same algorithms are used. The difference is that 3 D features focus on capturing unique shape information, so are based on 3D locations and surface normals, rather than pixel intensities. There are a large number of feature types implemented in PCL, including SHOT [217], NARF [211], Spin Images [96], Point Feature Histograms (PFH) [199] and Fast Point Feature Histogram (FPFH) [198, 201]. Following a review from Guo et al. [84], FPFH is selected, as it was shown to give superior performance to other descriptors, with lesser computational speed. FPFH captures information on the relationship between normals around a sample point to give a view-point-independent curvature metric.

The computation of a PFH, which forms the basis of FPFH, starts by comparing all the points within a set radius around the test point by computing the angles between their normal vectors. Three angles are computed, and the value of these three angles for every pair of points are used to fill a multi-dimensional histogram. There is one dimension for each angle and subdivisions into bins along each dimension. A given point pair adds one count to the bin that matches the range of values for the three angles. The result is a count for each bin that represents a feature descriptor. For example, with five divisions there would be $5^{3}=125$ parameters in the descriptor. This histogram is the Point Feature Histogram, that is a rotational invariant descriptor. FPFH is a modification to the process described above to improve the speed of the computation by approximating some of the computations and produces a set of 33 parameters [198, 201].

### 3.2.4.2 Correspondences and Alignment of Observations

With the features for every point on the map and surface computed, the next step is to iteratively determine the correspondences between observation points and map points as well as the transformation to align corresponding points. A pre-rejective RANSAC algorithm is used to perform this step [27], and was found to give better performance than Initial Alignment RANSAC [198] and ICP [228], all implemented in PCL. Pre-rejective RANSAC proceeds as follows:

1. Randomly select a set of three observation points (and corresponding FPFH descriptors) and match them to the closest FPFH features on the map object.
2. Reject the alignment set if the difference between the edge lengths of the polygon between map points and the polygon between their corresponding observation points is above a threshold (see [27] for details). If rejected, return to step 1. This is the pre-rejective step.
3. Compute the transformation to minimise the spatial distance between corresponding points using a Singular Value Decomposition (SVD) algorithm [53] (refer to Appendix A for details).
4. Apply the transformation to all observation points and perform a nearest neighbour search to determine the minimum distance from each observation point to a point on the map object.
5. Use the distances between corresponding points to determine the fraction of inliers (with a distance below a given threshold). If the fraction is below a threshold, reject the alignment and return to step 1.
6. Compute the mean nearest neighbour distance for all inlier points. Exit if this metric is below a threshold.
7. Apply the transformation to all observation points and return to step 1 for another iteration if the mean inlier distance is not below the threshold.
8. Repeat until the iteration limit or exit criteria are met.

The pre-rejection phase allows a quicker rejection of bad correspondences to improve the speed of the overall algorithm. If the iteration limit is reached without a successful convergence, then the object match is rejected, and the observation is deemed a new object. Examples of the alignment process are shown in Fig. 3.19. The approach can work well for partial observation of an object (Fig. 3.19.a), for localisation to a pre-mapped object, as well as for observations that extend beyond the current map of the object (Fig. 3.19.b), as is relevant for SLAM.


Figure 3.19. Demonstration of alignment. The blue dots represent sample points on a premapped surface. The observations are the orange dots, and the aligned observations are the black dots. (a) Alignment with a partial observation of a larger object. (b) Alignment with the observation extending beyond the edge of the existing object.

### 3.2.4.3 Estimation and Filtering

The alignment of the observed data to a mapped object is used as an observation in an Extended Kalman Filter (EKF) to estimate the state of the robot. An overview of an EKF can be found in [130]. The main steps are summarised here to provide context for the discussion. An EKF is an online estimator that produces an estimated state, $\boldsymbol{x}$ and covariance $\Sigma$. The filtering consists of a prediction step, which propagates forward the dynamics and grows the uncertainty, and an update step, which fuses observations to reduce the uncertainty. A Kalman filter provides a least-squares optimal way to update the estimation through these steps, given an accurate representation of the process noise, $\mathbf{Q}_{n}$, and the measurement uncertainty $\mathbf{R}_{n}$. The optimality assumes a linear system though, but it can be applied to non-linear systems by linearising about each subsequent estimate, in what is the EKF.

## EKF Background Theory

The process step updates the state estimate, adding noise:

$$
\begin{equation*}
\boldsymbol{x}_{k}^{-}=f\left(\boldsymbol{x}_{k-1}^{+}, \mu\right) \tag{3.32}
\end{equation*}
$$

There is an assumed, zero mean Gaussian noise added, in the model, $\mu$. The superscript - and + indicate the state estimate before and after the observation update step, respectively. $k$ indexes the steps of the filter. The noise does not effect the computation of Eqn. 3.32, but rather represents the process noise which causes the covariance to grow in the process step, $\mathbf{Q}_{n}$. The covariance of the state estimate, $\Sigma$, is updated using $\mathbf{Q}_{n}$ and the Jacobian of the process function $\mathbf{J}=\partial f / \partial \mathbf{x}$ :

$$
\begin{equation*}
\Sigma_{k}^{-}=\mathbf{J} \Sigma_{k-1}^{+} \mathbf{J}^{T}+\mathbf{Q}_{n} \tag{3.33}
\end{equation*}
$$

The update step compares observations, $\mathbf{z}_{k}$ with the predicted observations, $\hat{\mathbf{z}}_{k}=h\left(\mathbf{x}_{k}^{-}\right)+v$, and uses the error to update the state. The observations have assumed zero mean Gaussian noise, $v$, which is captured in the covariance matrix $\mathbf{R}_{n}$. This matrix, along with the Jacobian of the observation model to the state, $\mathbf{J}_{h}=\partial h / \partial \mathbf{x}$, is used to compute the Kalman gain, $\mathbf{K}$ :

$$
\begin{equation*}
\mathbf{K}=\Sigma_{k}^{-} \mathbf{J}_{h}^{T}\left(\mathbf{J}_{h} \Sigma_{k}^{-} \mathbf{J}_{h}^{T}+\mathbf{R}_{n}\right)^{-1} \tag{3.34}
\end{equation*}
$$

The Kalman gain is used for updating the state and state covariance estimates:

$$
\begin{align*}
\mathbf{x}_{\text {update }} & =K\left(\mathbf{z}_{k}-h\left(\mathbf{x}_{k}^{-}\right)\right)  \tag{3.35}\\
\mathbf{x}_{k}^{+} & =\mathbf{x}_{k}^{-}+\mathbf{x}_{\text {update }}  \tag{3.36}\\
\Sigma_{k}^{+} & =\left(\mathbf{I}-\mathbf{K} \mathbf{J}_{h}\right) \Sigma_{k}^{-} \tag{3.37}
\end{align*}
$$

## Handling Attitude Dynamics

Attitude dynamics are non-linear, hence Eqn. 3.36 is not valid for the attitude. Additionally, different attitude parameterisations can have issues in maintaining an unbiased covariance through the Kalman updates. Therefore a Multiplicative EKF (MEKF) [130] is used to address these issues to update the attitude estimate. This variant of the Kalman filter is selected due to the common use for attitude estimation. In an MEKF, the attitude state in the filter is a three parameter attitude error, represented by Rodrigues parameters, $\eta \mathbf{e}$, where $\mathbf{e}$ is the axis of rotation, and $\eta$ is the tangent of half the angle of rotation, $\phi: \eta=\tan (\phi / 2)$. The attitude error is defined as a rotation in the current body frame, using a quaternion representation:

$$
\begin{equation*}
\mathbf{q}_{\text {true }}=\Delta \mathbf{q}(\eta \mathbf{e}) \otimes \hat{\mathbf{q}} \tag{3.38}
\end{equation*}
$$

where $\hat{\mathbf{q}}$ is the current attitude estimate. The quaternion product is represented with $\otimes$. Refer to Appendix $C$ for details on quaternion maths. This representation is chosen to have a suitable threeparameter attitude representation to include in the state and covariance matrices, as described in [130]. Rodrigues parameters have a singularity at $180^{\circ}$ rotation, but as they are being used to represent
error, the angle remains far from that singularity. The attitude error is not updated in Eqn. 3.36, but is instead used to update a separately stored quaternion attitude state with Eqn. 3.38. After this update, the attitude error is reset to zero in $\mathbf{x}_{k}^{+}$. See [130] for more details on estimating attitude with the MEKF.

## NURBSLAM Process Model

The full state vector used here includes the linear position, velocity and acceleration as well as the attitude error, $\eta \mathbf{e}$, giving: $\mathbf{x}=[\boldsymbol{x}, \dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}}, \eta \mathbf{e}]$. A constant acceleration motion model is used in the process step to make the prediction for the linear states:

$$
\begin{align*}
& \boldsymbol{x}_{k+1}^{-}=\boldsymbol{x}_{k}^{+}+\dot{\boldsymbol{x}}_{k}^{+} \Delta t+0.5 \ddot{\boldsymbol{x}}_{k}^{+} \Delta t^{2}  \tag{3.39}\\
& \dot{\boldsymbol{x}}_{k+1}^{-}=\dot{\boldsymbol{x}}_{k}^{+}+\ddot{\boldsymbol{x}}_{k}^{+} \Delta t  \tag{3.40}\\
& \ddot{\boldsymbol{x}}_{k+1}^{-}=\ddot{\boldsymbol{x}}_{k}^{+} \tag{3.41}
\end{align*}
$$

The attitude error is assumed to stay constant at zero through the process step. ${ }^{2}$ The process model Jacobian is a function of the timestep between observations, $\Delta t$ :

$$
\mathbf{J}=\left[\begin{array}{rrrr}
\mathbf{I}_{3 \times 3} & \Delta t \mathbf{I}_{3 \times 3} & 0.5 \Delta t^{2} \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3}  \tag{3.42}\\
\mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3} & \Delta t \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3}
\end{array}\right]
$$

## NURBSLAM Update Step

The matching and alignment of each observed object to a map object produces a linear and angular transformation to the state. These transformations are taken as direct measurements of error from the current state estimate, represented as a translational error and a rotational error, as Rodrigues parameters. The observation of a single object is then:

$$
\mathbf{z}_{k}=\left[\begin{array}{c}
\boldsymbol{x}_{\mathrm{err}}  \tag{3.43}\\
\eta \mathbf{e}_{\mathrm{err}}
\end{array}\right]
$$

The predicted measurements are for zero error, i.e. $h(\mathbf{x})=\mathbf{0}$, and the Jacobian is:

$$
\mathbf{J}_{h}=\left[\begin{array}{llll}
\mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3}  \tag{3.44}\\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} & \mathbf{I}_{3 \times 3}
\end{array}\right]
$$

Using the alignment transformations as direct observations makes the update step very simple ${ }^{3}$ but it becomes less clear how to represent the observation uncertainty, $\mathbf{R}_{n}$. Nonetheless, $\mathbf{R}_{n}$ can be

[^6]used to adjust how much to trust a given observation, based on metrics from the alignment. The inlier fraction (the fraction of observed points within the inlier threshold), number of observed points and the magnitude of the resulting transformation are used to scale the magnitude of $\mathbf{R}_{n}$. These metrics adjust an approximate standard deviation on each observation parameter, which is used to compute a covariance in the $\mathbf{R}_{n}$ matrix:
\[

$$
\begin{align*}
\sigma_{\text {linear }} & =\frac{1}{3}\left(\sigma_{\text {base }}+w_{0} \frac{n_{\text {inlier }}}{n_{\text {points }}}+w_{1} n_{\text {points }}+w_{2}\left|\boldsymbol{x}_{\text {err }}\right|^{4}\right)  \tag{3.45}\\
\sigma_{\text {angular }} & =\frac{1}{3}\left(\sigma_{\text {base }}+w_{3} \frac{n_{\text {inlier }}}{n_{\text {points }}}+w_{4} n_{\text {points }}+w_{5} \phi^{4}\right) \tag{3.46}
\end{align*}
$$
\]

The weightings $w_{0}-w_{5}$ are parameters that can be tuned to adjust how sensitive $\mathbf{R}_{n}$ is to the different metrics. In the results presented here, it was found that the magnitude of the transformation was the best indicator of a low quality alignment, hence $w_{2}$ and $w_{5}$ were the most important to tune. The error values are raised to the power of 4 to have small penalty for low errors and rapidly increasing penalties when the errors become large. The $\mathbf{R}_{n}$ matrix is then constructed with the $\sigma$ values to give:

$$
\mathbf{R}_{n}=\left[\begin{array}{rrr}
\sigma_{\text {linear }}^{2} \mathbf{I}_{3 \times 3} & & \mathbf{0}_{3 \times 3}  \tag{3.47}\\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \mathbf{0}_{3 \times 3} \\
\mathbf{0}_{3 \times 3} & \sigma_{\text {angular }}^{2} & \mathbf{I}_{3 \times 3}
\end{array}\right]
$$

With the $\mathbf{R}_{n}$ matrix, the Kalman gain, and EKF state, the Kalman update can then be computed.
Multiple observations are fused into the filter by iteratively applying the update step for each observation. In each iteration, the observed alignment transformation from a single NURBS object, $\mathbf{q}_{\text {obs }}, \mathbf{t}_{\text {obs }}$ is adjusted to be defined from the latest state estimate, $\hat{\mathbf{q}}, \hat{\boldsymbol{x}}$.

$$
\begin{align*}
& \mathbf{q}_{\mathrm{obs}}=\mathbf{q}_{\mathrm{obs}} \otimes \hat{\mathbf{q}}  \tag{3.48}\\
& \mathbf{t}_{\mathrm{obs}}=\hat{\boldsymbol{x}}-\hat{\mathbf{q}} \otimes \mathbf{t}_{\mathrm{obs}} \otimes \hat{\mathbf{q}}^{c} \tag{3.49}
\end{align*}
$$

The map objects are not included in the filter and are instead updated as described in Section 3.2.3, using the estimated state after the update step to project the observations into the global coordinate frame. Future developments could look to include NURBS parameters of the observed features in the filter state. Additionally, future work could look to integrate IMU measurements. The filter described currently does not integrate an IMU but it is formulated such that this can easily be done, by including IMU measurements and noise characteristics in the process step.

### 3.2.4.4 Localisation Discussion

While the localisation approach is point-based, it draws on the advantage of an underlying continuous surface representation in being able to take point samples at varying resolutions, depending on the trade-off between accuracy and computational speed desired. Additionally, the surface representation can be efficiently updated, and points re-sampled, rather than storing a growing set of points.

The number of points to sample on a surface is just one of the tuning parameters available for the localisation steps. Other parameters include the number of points to take from the scan, the radius to use for computing the normals, the radius to use for computing the features, the number of RANSAC iterations, and the inlier threshold. The parameters do need to be tuned to suit a given data-set but can also provide an ability to adjust the trade-off between computational speed and accuracy, as analysed in Section 3.3.3.1.

### 3.2.5 SLAM

SLAM combines the mapping and localisation steps described above and is outlined in Fig. 3.20. For the theory described here, it is assumed that there is no IMU information. The initial observation of an object is used to create a new NURBS object, that is stored in the map. Subsequent observations then go through the data association step. If there is no match, a new NURBS object is created; otherwise, the match is used to perform localisation. The EKF in the localisation step is used to update the state estimate and covariance. The new state estimate is used to transform the current observations into the global frame, which are then used to update the NURBS objects to which they have been matched. If an alignment is rejected, then a new NURBS object is created. Any new objects are created after updating the state estimate, to use the latest, best estimate for transforming the observations into the global frame.

The current implementation does not include the map state in the EKF; hence the combined joint optimisation common in SLAM is not present. This approach is taken to reduce the computational complexity of storing many parameters of a NURBS object in an EKF state.


Figure 3.20. Flow diagram for the SLAM algorithm. Solid lines indicate the steps in the process, with a description of the criteria for that step. Dashed lines indicate both the transfer of data and the steps in the process. Each box indicates one step of the SLAM algorithm.

### 3.2.6 Trajectory Optimisation

To represent an obstacle for trajectory planning, a NURBS object needs to take a set of query points (the points along a trajectory), and return a collision cost, as well as a gradient of that cost with respect to $x, y$ and $z$, i.e. it should represent a cost function, $g\left(\boldsymbol{x}_{i}\right)$, where $\boldsymbol{x}_{i}$ is the $i$ th 3D position sample on a trajectory. The cost should be positive when in a collision and negative outside of a collision. For informing a gradient descent trajectory optimisation algorithm, the gradient should be returned as:

$$
g^{\prime}\left(\boldsymbol{x}_{i}\right)=\left[\begin{array}{l}
\frac{\partial g}{\partial x_{i}}  \tag{3.50}\\
\frac{\partial g}{\partial y_{i}} \\
\frac{\partial g}{\partial z_{i}}
\end{array}\right]
$$

For more details on requirements of obstacle representations for trajectory optimisation, see Section 4.3.3. For a NURBS object, the cost function uses the negative of the signed distance from a point to the surface. The signed distance is positive when away from the surface and negative inside it. The distance to the surface is found by using correspondence tools in PCL to find matches between the set of points along a trajectory and the closest points from a set of sample points on the surface. The correspondence algorithm returns the distance between matched points to provide the signed distance magnitude. To determine the sign, the vector from the matched surface point to the trajectory point, $\mathbf{v}_{\text {st }}$
is compared with the outward-facing surface normal at the matched surface point $\mathbf{n}_{\mathrm{st}}$, which can be directly sampled from the NURBS object. If the dot product is negative, the angle is greater than $90^{\circ}$, hence the point is inside the surface:

$$
\mathbf{s i g n}=\left\{\begin{array}{l}
+ \text { if } \mathbf{v}_{\mathrm{st}} \cdot \mathbf{n}_{\mathrm{st}} \geq 0  \tag{3.51}\\
- \text { if } \mathbf{v}_{\mathrm{st}} \cdot \mathbf{n}_{\mathrm{st}}<0
\end{array}\right.
$$

The vector, $\mathbf{v}_{s} t$, can be used as an approximate cost gradient when multiplied by the computed sign in Eqn. 3.51. Using this vector requires no extra computation, in contrast to using numerical differentiation as can be required for other obstacle representations, such as ESDFs.

In the case where a surface does not fully enclose an object, a cap is placed on the maximum negative distances that are returned, so that there is not a restriction on all space behind the surface. Trajectory points that are past that maximum distance are given zero distance and gradient so that they do not contribute to the overall cost and gradient that is used by the trajectory optimisation algorithm.

The advantage of using NURBS objects to represent obstacles over occupancy grids is that there is an underlying continuous surface representation. This representation means that the surface can be sampled at varying resolutions. Additionally, low sampling resolutions are possible because surface normals can be used to detect when a point is in a collision.

### 3.3 NURBSLAM Demonstration, Testing and Analysis

Tests are performed with NURBSLAM on a simulated dataset generated with Blensor [82], a 3D sensor simulation tool. The implementation of NURBSLAM utilises the NURBS++ library [117] and PCL [200]. A simplified test case is developed to analyse the performance of NURBSLAM. This scenario involves an RGBD camera following a circular path around one central object. The camera's gaze is constantly fixed on this object, and 100 observations are made in one complete rotation. Three configurations are tested to isolate each component of the NURBSLAM algorithm:

Mapping: Generating and updating an object with perfect knowledge of the camera state.
Localisation: Starting with a known map and estimating the state of the camera.
SLAM: Generating and updating a map of objects while using those objects to localise.

For a majority of the simulations, the central object is a highly distorted spheroid. The true camera state comes from the simulation, and the only observation inputs are point clouds (there is no IMU information included). There are no other objects in view so that the performance of NURBSLAM can be isolated from segmentation algorithms. To further simplify the problem, and demonstrate what is possible with NURBS, the point cloud observations have no noise applied. Presented below are the results and analysis from tests with each configuration, followed by an example of trajectory planning with the mapped NURBS objects.

### 3.3.1 Mapping

The sequence of map generation and update are depicted in Fig. 3.21, with three updates to the NURBS object. The mesh reduction steps mean that the top and the bottom of the object are not mapped but the resulting surface provides an accurate representation of a majority of the object, as shown in Fig. 3.22.

Fig. 3.23 shows the results from mapping a sphere and a cube, showing that continuous curvature, as well as sharp corners, can be handled. Table 3.3 presents the Root Mean Square Error (RMSE) for each of these mapped objects, plus a scenario of going vertically over the cube. The RMSE is computed by using the minimum distance from a set of points on the NURBS surface to the true object.

TABLE 3.3. RSME errors for mapping tests

| Test Case | RMSE (mm) |
| :---: | :---: |
| Distorted spheroid | 4.9 |
| Cube | 4.1 |
| Cube Vertical | 2.9 |
| Sphere | 4.3 |

Each of the mapping examples has a gap in the surface when completing the loop. This gap is because the method of selecting the new data to use for extending the surface returns no extension when all edges of the new data are overlapping the existing surface. A complete map can be generated, though, by creating a new NURBS object when there have been too many consecutive observations with no extension of the surface. The results from using multiple NURBS objects are shown in Fig. 3.24.


Figure 3.21. Mapping sequence for the distorted spheroid object. The blue surface is the true object and the black mesh is the NURBS object. (a) Initial surface. (b) First extension. (c) Second extension. (d) Third extension.


Figure 3.22. Final mapping result from a single orbit of the distorted spheroid object. The blue surface is the true object and the black mesh is the NURBS object. (a)-(d) are different view angles of the same object and mesh.


Figure 3.23. Mapping results for different objects. The blue surface is the true object and the black mesh is the NURBS object. (a) Sphere object. (b) Cube object.


Figure 3.24. Mapping result for the distorted spheroid object when using multiple NURBS surfaces. The blue surface is the true object and the black mesh is the NURBS object. (a) Result with a single NURBS object. (b) Result with multiple NURBS objects.

### 3.3.2 Localisation

The localisation tests use the map of NUBRS objects that results from a multi-surface mapping of the distorted spheroid object. The state is initialised with the true state and a large covariance. To compute the tracking error for the attitude, an angular error is computed using the difference between the true rotation matrix, $\mathbf{R}_{t}$ and the estimated rotation matrix, $\mathbf{R}_{e}$ :

$$
\begin{equation*}
\phi \mathbf{e}_{a t t}=\frac{1}{2}\left(\mathbf{R}_{e}^{T} \mathbf{R}_{t}-\mathbf{R}_{t}^{T} \mathbf{R}_{e}\right)^{V} \tag{3.52}
\end{equation*}
$$

The vee operator, ${ }^{V}$, is the inverse of a hat operator: it maps a skew symmetric matrix to a vector:

$$
G^{V}=\left[\begin{array}{ccc}
0 & -g_{3} & g_{2}  \tag{3.53}\\
g_{3} & 0 & -g_{1} \\
-g_{2} & g_{1} & 0
\end{array}\right]^{V}=\left[\begin{array}{l}
g_{1} \\
g_{2} \\
g_{3}
\end{array}\right]
$$

This operator can be used because $\left(\mathbf{R}_{e}^{T} \mathbf{R}_{t}-\mathbf{R}_{t}^{T} \mathbf{R}_{e}\right)$ is skew-symmetric. The angular error vector, $\phi \mathbf{e}_{a t t}$ is a rotation vector; hence the vector components represent errors about the $x, y$ and $z$ axes. The magnitude of the vector, $\phi$, represents the single angular error to rotate from the estimated attitude to the true attitude.

The tracking results from an orbit of localisation are shown in Fig. 3.25, with the linear and angular error along the trajectory shown in Fig. 3.26. There are several large spikes of error visible in Fig. 3.26, yet the localisation quickly recovers after these errors. The source of the errors is due to bad alignments in specific observations, which are filtered by the EKF but still have a negative impact on tracking performance.


Figure 3.25. 3D track of localisation testing. The true path is in orange and the SLAM tracked path is in blue. Camera frame axes are plotted at even intervals along each trajectory. The trajectory starts near $x=0, y=-4$ and travels anti-clockwise.

An analysis of the odometry error can give further insight into the fact that the main errors are from a small set of bad alignments. Odometry error is computed by taking small segments of the tracked trajectory, aligning the start with the true state, and then computing the linear and angular error at


Figure 3.26. Error plots for localisation test. (a) Linear error plots. (b) Angular error plots.
the end of the segment. By taking segments starting at each point along the trajectory, the variation in odometry error can be analysed. A depiction of a small set of the odometry error segments is shown in Fig. 3.27.b. The localisation test is analysed with a segment size of 10 steps, producing the result in Fig. 3.27.a.


Figure 3.27. Localisation odometry error. (a) Linear and angular odometry error from localisation tests. (b) Depiction of how odometry error is computed. The blue trajectory is the truth, each other line is a segment of the localisation tracking that has had the starting pose aligned with the truth. The odometry error for a given step is the error of the pose at the end of the aligned segment.

The odometry error results confirm that the tracking is generally low in error, with a small set of spikes in error, after which tracking quickly recovers to a low odometry error. Fig. 3.27.a also shows that the position and attitude errors are coupled, with errors due to inaccurate transformations, including both position and orientation.

The RSME for the localisation track is 0.138 m for position and $3.6^{\circ}$ for attitude (using $\phi$ as the error for each step). These low errors are despite only using a single object. Additionally, the trajectory is challenging for a constant acceleration model, as the acceleration is never constant.

### 3.3.3 SLAM

For the SLAM tests, the initial pose is given, and then the only information provided is the point cloud observations. The resulting trajectory and map are shown in Fig. 3.28. The trajectory starts to lose track towards the last quarter of the circle, and as a result, generates two new NURBS objects that are not near the true object. The plots of the linear and angular error in Fig. 3.29 give more insight into the tracking performance, where there are relatively small errors until after observation 70, where there is a large jump in orientation error. Such a jump can occur when there is a succession of failed alignments, followed by an inaccurate, but accepted, alignment. The failed alignments increase the size of the covariance, to result in a large Kalman gain for the angular update, even though the observation uncertainty may be large. The velocity builds up over the failed alignments and leads to drifting of the trajectory away from the circle. This drift leads to the need to generate new NURBS objects. The bad alignments occur because the observations are in a difficult region for localisation on the true object, with a largely spherical region being observed. This geometric ambiguity makes the alignment susceptible to incorrect results.


Figure 3.28. SLAM Tracking and Mapping. The true trajectory is orange and the SLAM tracked trajectory is blue. Camera frame axes are plotted at even intervals along each trajectory. The true object is blue and the NURBS objects are black. The trajectory starts near $x=0, y=-4$ and travels anti-clockwise.

Other than the one region of lost tracking, the performance is strong throughout the trajectory, as shown in the odometry error analysis in Fig. 3.30. The odometry error is consistently low, with a set of spikes to higher error, after which odometry tracking returns to being low. Fig. 3.30.b provides a


FIGURE 3.29. Error plots for NURBSLAM tracking. (a) Linear error. (b) Angular error.
visualisation for where the larger tracking errors occur, and how there are only a small set of locations where the aligned segments of the trajectory diverge from the true path. While the new objects that are generated are far from the true surface, they allow the drift to be limited, and to continue tracking the odometry accurately.


Figure 3.30. SLAM odometry error analysis. (a) Linear and angular odometry errors across the trajectory. (b) 3D plot of aligned segments that are used to compute the odometry error. A circular trajectory starting near $x=0, y=-4$ and travelling anti-clockwise is the truth.

If the results are extracted up until observation 70, then the tracking is accurate throughout, and the three objects that are generated are close to the true object (Fig. 3.31). The RMSE for tracking and mapping for the full loop, and up to observation 70 are shown in Table 3.4, which highlight how
accurate the mapping is before the trajectory drifts away. Despite the momentary loss of tracking, the RSME for the full loop is within $2 \%$ of the total path length, in a challenging trajectory with only a single object to observe.


Figure 3.31. Mapped object from SLAM test case for $70 \%$ of the trajectory. The blue surface is the true object and the black meshes are the NURBS objects. Three NURBS objects are generated. (a)-(d) are different view angles of the same object and meshes.

TABLE 3.4. Tracking and mapping errors for SLAM test case

| RMSE | Full loop | 70\% of loop |
| :---: | :---: | :---: |
| Position (m) | 0.57 | 0.33 |
| Angular (deg) | 9.77 | 6.07 |
| Mapping (m) | 0.47 | 0.02 |

### 3.3.3.1 Parameter Sensitivity and Timing Analysis

An analysis is performed to assess the sensitivity of the computation time and accuracy of NURBSLAM to varying parameters. This analysis also provides insight into the trade-off between computation time and accuracy by exploring different configurations.
The parameters varied are:
Number of RANSAC Iterations: The maximum number of iterations in the pre-rejective RANSAC algorithm. The baseline is 5000 .

Number of Rows and Columns for Mesh Generation: Controls the size of the observation data that is then used for localisation and update. The baseline is 95 .

Surface Points Multiplier: Sets the number of surface points to generate for localisation (multiplies the number of control points). The baseline is 5.0
Number of Control Points: The default number of control points in each parametric direction that is used when generating a new NURBS surface. The baseline is 17 .

The same test scenario as presented for the SLAM result above is repeated multiple times, each with a variation in a single parameter. The average computation time per observation is recorded for each test, and the errors computed. The tests are run on an Intel i7-7820 2.90 GHz 8 core processor with 16GB RAM on 64 bit Ubuntu 16.04.

A line is fit through the results for single parameter variations, and the gradient used to assess the sensitivity of NURBSLAM to the parameter. The sensitivities are weighted by multiplying the gradient by the baseline parameter setting to enable a comparison between parameters. The resulting scaled sensitivities are presented in Table. 3.5. A combined error metric is defined to have a single parameter of comparison. This combined error is computed by first normalising the RMSE for position, angular and mapping errors by dividing by the maximum value across the test cases. Then the RMS of the normalised values is computed to give the combined error metric.

TABLE 3.5. Scaled sensitivity of errors and computation time to NURBSLAM parameters

| Sensitivity of | RANSAC Iter. | \# Rows \& Cols | Surf. Pnt Multiplier | \# Ctrl Pnts |
| :---: | :---: | :---: | :---: | :---: |
| Position RMSE | -0.14 | 0.14 | 1.26 | $\mathbf{- 1 . 9 9}$ |
| Angular RMSE | -0.05 | -0.09 | 0.30 | $\mathbf{- 0 . 2 9}$ |
| Mapping RMSE | 0.25 | -0.22 | 0.49 | $\mathbf{- 0 . 7 1}$ |
| Total RMSE | 0.04 | -0.19 | 0.72 | $\mathbf{- 1 . 0 0}$ |
| Mean Comp. Time (s) | 1.90 | 1.70 | 2.16 | $\mathbf{- 0 . 2 8}$ |

The most sensitive parameter with a negative gradient is in bold. The most sensitive parameters with a positive gradient is in italics.

The variation in the default number of control points has the most significant impact on the results, with more control points leading to a significant reduction in both tracking and mapping error. Interestingly, the changes also have minimal impact on computation time.

Increasing RANSAC iterations, the number of rows and columns, and the surface points multiplier increases computation time, as expected. The primary source of the computation time increase is in the RANSAC alignment, where increasing the number of points increases the time to check for inliers and outliers. Increasing the RANSAC iterations, surprisingly, has a minimal beneficial impact on
performance. Similarly, changing the number of rows and columns does not have a significant impact on the tracking or mapping error.

The surface points multiplier has a negative impact on tracking and mapping performance when increased. This result may seem counter-intuitive but is because there is an increased possibility of having false positives in the RANSAC alignment.

Plotting the RSME against the computation time for all the tests, as in Fig. 3.32, provides an insight into the trade-offs that are possible. While the baseline case has the lowest tracking error, a similar amount of error, at a much-reduced computation time can be obtained by reducing the number of RANSAC iterations. Fewer iterations are less robust though, due to the random nature of the sampling approach. The trends for the mapping error are less clear, as the errors can vary substantially depending on when new objects are created and what the tracking error is at the time. There is some randomness in the alignment process, from RANSAC; hence a large set of tests would be beneficial in future work to characterise trends in more detail.


Figure 3.32. Accuracy/computation time trade-off analysis. Colored dots represent test cases, with coloring indicating variations in different NURBSLAM parameters. (a) Position error. (b) Angular error. (c) Mapping error. (d) Combined error.

With the current implementation, there is, in general, a long computation time per scan, at an average of 2.7 s from the tests in Fig. 3.32. A majority of this time is spent in the alignment step, as outlined in the timing analysis in Table 3.6. The mesh processing is very quick, taking at most $0.5 \%$ of the computation time for a scan. The map update can take up to $54.6 \%$ of the computation time for a given step, but only when there is an extension to the surface, which happens infrequently. Data association is of negligible computation load due to the small number of features considered.

TABLE 3.6. Percentage of computation time for a single scan for steps of the NURBSLAM process.

| Step | Mean Time (\% of total) | Max \% of total |
| :---: | :---: | :---: |
| Mesh Processing | 0.3 | 0.5 |
| Alignment | 82.4 | 95.2 |
| Map Update | 17.2 | 54.6 |
| Other | 0.1 | 0.1 |

### 3.3.4 Trajectory Optimisation

A trajectory is planned using the mapped NURBS objects as obstacles in the Admissible Subspace TRajectory Optimizer (ASTRO), an algorithm that is described in the next chapter. The trajectory is planned with a buffer for the vehicle size of 0.3 m . Fig. 3.33.a shows the successfully planned trajectory around the object. The initial plan is a straight line between the start and end locations that passes through the middle of the object. The signed distance for 500 samples along the initial plan, plotted in Fig. 3.33.b, shows that the distance and distance gradient are appropriate measures to push a trajectory out of a collision: there is a peak negative distance at the most violating point, and monotonically increasing distances on either side. While a sharp peak is not desired for a cost function, the signed distance is summed across the whole trajectory, and then squared, smoothing the cost function for trajectory planning (see Section 4.3.3 for more details on the desired traits of cost functions).


FIGURE 3.33. Trajectory planning with NURBS objects. (a) Trajectory planned from the red circle to the black circle with a 0.3 m buffer. The true obstacle is blue and the NURBS objects are the black meshes. (b) Signed distance along a straight-line trajectory from the start to the end, passing through the middle of the obstacle. Negative distances are inside the object, and a 0.3 m buffer is used. There are 500 samples in time along the trajectory.

### 3.4 Conclusion

This chapter has demonstrated the concept of using 3D objects as the features for SLAM and as obstacles for trajectory planning. An analysis of different methods for modelling 3D objects was presented, including a full implementation and tests of Ellipsoid-SLAM: using ellipsoids as the 3D object representation. These tests showed ellipsoids to be inadequate for mapping and localisation. Gaussian Process Implicit Surfaces were also analysed and shown to have restrictively high computation times. Non-Uniform Rational B-Splines (NURBS) were identified as providing the best trade-off between computational speed and accuracy.

The theory for NURBS-based Localisation and Mapping (NURBSLAM), was developed, including surface-fitting to data, extending surfaces from multiple observations, performing surface alignment with 3D feature descriptors for localisation information, and using a Multiplicative Extended Kalman Filter for state estimation. NURBS are used for obstacle representation by utilising surface normals to help computation of a signed distance from a trajectory sample point to the surface.

Tests of NURBSLAM on simulated data successfully demonstrate the concept of using NURBS surfaces for mapping, localisation, SLAM, and trajectory planning. Compared to other approaches to using a 3D representation for SLAM and trajectory planning, NURBS provides the benefits of:

- A continuous 3D representation that can be sampled at multiple resolutions.
- The ability to have an accurate surface representation of an object.
- Quick generation and update of surfaces.
- Rich information for localisation by aligning surfaces.
- A method to compute a signed distance for use as an obstacle in trajectory planning.

The limitations of NURBS compared to other approaches are:

- Slow computation for alignment.
- Susceptibility to bad alignments, especially for uniform shapes.

Using NURBS objects as the features for SLAM means that the SLAM map is immediately useful for trajectory planning. Additionally, with 3D objects modelled in the map, there is object shape information that could be used to represent dynamic objects, for object interaction tasks and to aid object classification in future work.

Further work is needed to develop the concept of NURBSLAM that has been demonstrated here. Computation time could be improved with the use of keypoints in the alignment step. Additionally, the mapping approaches could include an update of the existing surface with overlapping observations, to improve the surface estimate and handle noisy observations. The tests demonstrated here are for simulated, noiseless data with a single object. The method of estimation in SLAM could also be further developed by looking into pose-graph optimisation approaches (e.g. using GTSAM [47]), in contrast to Kalman filters, to obtain more accurate results by optimising of a whole history of poses.

Further testing and development should look to apply NURBSLAM to real data and to test the algorithm with multiple objects.

Nonetheless, the capability of NURBSLAM presented here is sufficient to assess the benefit that comes from having a single 3D representation for SLAM and trajectory planning. The full integration of

NURBSLAM into the autonomous navigation stack is presented in Chapter 7, including a comparison to the standard, heterogeneous approach to SLAM and mapping. The next layers in the autonomous navigation stack are investigated next, in Chapters 4-6, to build up to the full system demonstration.

## TRAJECTORY OPTIMISATION



## Attributions:

The core theory behind the algorithm that will be presented comes from Chamitoff et al. [35]. The author of the work presented here contributed to [35] in: a) clarifying and developing the theory from an earlier publication [34]; b) analysing results from on-orbit experiments and extracting lessons learned, in particular, the dynamic obstacle cases; and c) expanded and enhanced simulated test cases. These contributions from [35] are included in this chapter. The section on on-orbit testing is adapted from [35], and is presented to give context to a presentation of the analysis of results. The theory presented here adapts and expands from what is presented in [35].

A number of the results with static obstacles, dynamic obstacles, and computation time analysis have been presented in [35, 151, 155]. All of these results, unless otherwise stated, are work of the author of this thesis. One set of results on randomised perturbations, and work on fitting trajectories for replanning come from a collaboration with Rigter [195] as is noted in the text. Citations and further statements clarify these contributions throughout the chapter.
he planning layer of the autonomous navigation stack is addressed in this chapter, with goals to enhance the capability to produce dynamically-optimal trajectories with static and dynamic
obstacles. These goals are realised through the Admissible Subspace TRajectory Optimizer (ASTRO), first introduced by Chamitoff et al. [33]. ASTRO solves a trajectory optimisation problem between boundary conditions with obstacle constraints, restrictions on the volume of operation, and limitations on performance. The optimisation objective can be to minimise path-length, velocity, acceleration, and snap (4th derivative with respect to time). The key components of ASTRO are a parameterisation of the trajectory with Legendre polynomials, a subspace-projection to enforce boundary conditions in every step of a gradient-descent optimisation, and a consideration of a broad range of constraints. Compared to the current state-of-the-art, the key differentiating factors of ASTRO are:

1. The flexibility of constraint descriptions, with the ability to include obstacles, performance limitations and free-space restrictions all in the same framework. This flexibility of representation allows the algorithm to be applied to a range of different scenarios, such as representing a small set of discrete obstacles or restricting free-space to navigate through a large cluttered building.
2. The method of considering dynamic obstacle constraints, to predict their position and adjust their size given the uncertainty in position.
3. Optimisation techniques to enable the generation of feasible solutions to complex, non-convex problems with many constraints. These techniques make it feasible to include many constraints together in the one optimisation.
4. The algorithm has been demonstrated on a robotic free-floating satellite in orbit.

This work expands and enhances an early version of ASTRO, as will be described in detail in this chapter. The main contributions in this work will first be presented, before a thorough description of the theory behind the current version of the algorithm. The theory includes: the core formulation; constraints to represent static obstacles, dynamic obstacles and performance limitations; optimisation methods; replanning and multiple robot considerations; and multi-segment optimisation. A set of simulated test cases then demonstrate the capability of ASTRO and analyse the performance of the algorithm. Finally, the application of ASTRO to free-flying satellites is described, and analysis presented of the on-orbit testing of an early version of the algorithm.

### 4.1 Contributions

The work of this author expanded from an early version of ASTRO [34], starting with an analysis of results from on-orbit testing. The key contributions made in this work are:

## Core algorithm:

1. The generalisation of ASTRO to have a trajectory cost operating on any derivative, including changes to the subspace-projection.
2. Modification of how the trajectory is parameterised to take full advantage of the cost formulation: the Legendre polynomials represent the derivative that is used by the cost function.
3. Application of the subspace-projection step to full coefficient sets: a capability useful for replanning applications.
4. A detailed analysis of the convexity of the problems solved by the algorithm, plus clear descriptions of when problems are non-convex.

## Constraint Formulations:

1. Proofs of convexity of constraint formulations, and explanations of non-convexity where applicable.
2. Expansion of constraint formulations to include an approximate path-integral cost method.
3. Addition of new constraint types: rectangular prisms and Euclidean Signed Distance Fields.
4. Introduction of a class of dynamic obstacles that have consideration of predicted location and orientation, enabling more efficient trajectories than previous approaches.
5. Development of an approach to grow the size of an obstacle based on the uncertainty in position.

## Optimisation Techniques:

1. Formulation of a quadratic line-search on projected gradient steps for convex problems, to enable rapid optimisation.
2. Introduction of iterative optimisation on simplified sub-problems, to increase computational speed and enable solutions to problems with many constraints.
3. Adapting constraint weights based on initial trajectory costs and constraint costs, to automatically compute an appropriate weight for a range of scenarios.
4. Inflation of obstacle constraint sizes to have a stronger forcing of solutions into feasible space. The coupling of this inflation with a criterion to exit when the trajectory is feasible (collision-free), enables rapid generation of feasible trajectories.
5. Extension of work by Marc Rigter [195] to apply randomised initial seeding of trajectories to help the algorithm find better local minima in non-convex scenarios. This concept is extended to formulate random perturbations of a solution when it is detected to have converged in an infeasible local minimum. The perturbations serve to jump the trajectory out of the infeasible local minima.
6. Expanding ASTRO to multi-segment optimisation, with an application of principles from [26], including an outer-loop optimisation of time. This formulation expands the possible applications of ASTRO to large, and more complex environments.

## Multiple Robot Applications:

1. Specification of a replanning framework to take into account limited knowledge of the environment and computation delays.
2. Demonstration of multi-robot applications of ASTRO, including adversarial and cooperative approaches.

## Analysis:

1. Extracting lessons learned from on-orbit testing to inform future development, including analysis of the algorithm for online planning with real dynamic obstacles.
2. Simulated test cases to demonstrate the performance of the algorithm on a wide range of scenarios.
3. Computation time analysis, including discussion on the potential for the algorithm for use in real-time applications.


Figure 4.1. The ASTRO algorithm optimises the path between boundary conditions $\boldsymbol{x}\left(t_{0}\right)$ and $\boldsymbol{x}\left(t_{f}\right)$, with constraints along the path, such as obstacles.

### 4.2 Preliminaries

For clarity in the descriptions in this chapter, some definitions and notation conventions are presented here.

The term path is used to describe a purely physical sequence of positions, e.g. a set of $x, y, z$ coordinates. A trajectory is a time-dependent path, where each state along the path has an associated time. A trajectory, therefore, has time derivatives and can be described by any of these trajectories, such as a velocity trajectory.

The use of bold font represents vector variables: $\boldsymbol{x}$ is a vector, whereas $x$ is a scalar.
Trajectory optimisation is the process of planning a trajectory while minimising a cost function. Trajectories can be planned without being optimised, but in this chapter the terms plan and optimise are used interchangeably because ASTRO is always planning trajectories that are optimising an objective.

Other terms are defined throughout the document.

### 4.3 Algorithm Description

The core of ASTRO is designed to optimise a polynomial trajectory between two boundary conditions, i.e. a starting state to an end state, in a fixed time, as illustrated in Fig. 4.1. The optimisation minimises an integral of a property of the trajectory, such as acceleration or snap, to have a dynamically-smooth trajectory. The boundary conditions can include position, velocity, and higher derivatives, as well as orientation. Between the boundary conditions are constraints on the trajectory, including obstacles, performance constraints and limits to the volume of operation. Each dimension of the trajectory is represented by a sum of basis polynomials multiplied by coefficients. The coefficients are the optimisation parameters that are adjusted to minimise a cost function that includes a trajectory cost and constraint cost.

The trajectory cost is the integral of a state derivative squared, such as the integral of velocity squared, integral of acceleration squared, or the integral of snap, the fourth derivative with respect to time, squared. The selection provides different qualities of optimised trajectories. The integral of
velocity squared, for instance, is a combination of minimising the path length and minimising the variations in velocity along the path. The integral of snap squared can help to minimise control input for quadrotors, as described in Chapter 5. The trajectory cost is expressed, for $d$ dimensions, operating on the $\xi$ th derivative, as:

$$
\begin{equation*}
f_{s}=\int_{t_{0}}^{t_{f}}\left(\sum_{i=1}^{d} x_{i}^{(\xi)}(t)^{2}\right) \mathrm{d} t \tag{4.1}
\end{equation*}
$$

Time is fixed to be from $t_{0}$ to $t_{f}$. The trajectory of the $\xi$ th derivative for the $i$ th dimension is given by $x_{i}^{(\xi)}(t)$, where the zeroth derivative, $x_{i}^{(0)}(t)$ is simply $x_{i}(t)$. Each $x_{i}^{(\xi)}(t)$ is represented by a sum of Legendre polynomials $P_{k}$, up to order $N_{i}-1$, multiplied by coefficients $C_{i k}$ for each dimension $i$ :

$$
\begin{equation*}
x_{i}^{(\xi)}\left(t^{\prime}\right)=\sum_{k=0}^{N_{i}} C_{i k} P_{k}\left(t^{\prime}\right) \tag{4.2}
\end{equation*}
$$

The coefficients, $C_{i k}$, are the free parameters that are modified to optimise the trajectory, and comply with constraints. Time is normalised to be between -1 and 1 :

$$
\begin{equation*}
t^{\prime}=2\left[\frac{t-t_{0}}{t_{f}-t_{0}}\right]-1 \tag{4.3}
\end{equation*}
$$

Because the $\xi$ th derivative is represented with the Legendre polynomials, lower derivatives are attained through integration, and higher derivatives through differentiation. In general, an $r$ th derivative, if $r<\xi$ is given by:

$$
\begin{align*}
x_{i}^{(r)}\left(t^{\prime}\right) & =\frac{1}{a^{\xi-r}} \sum_{k=0}^{N_{i}} C_{i k}\left[\underline{\int \cdots \int_{(\xi-r)}} P_{k}\left(t^{\prime}\right) \underline{\mathrm{d} t \cdots \mathrm{~d} t_{(\xi-r)}}\right] \quad \text { if: } r<\xi  \tag{4.4}\\
& =\frac{1}{a^{\xi-r}} \sum_{k=0}^{N_{i}} C_{i k}\left[P_{k}^{\int_{(\xi-r)}\left(t^{\prime}\right)}\right]
\end{align*}
$$

 The term $a$ is a weighting to account for the normalised time scaling:

$$
\begin{equation*}
a=\frac{2}{t_{f}-t_{0}} \tag{4.5}
\end{equation*}
$$

 derivative where $r \geq \xi$ is given by:

$$
\begin{equation*}
x_{i}^{(r)}\left(t^{\prime}\right)=a^{r-\xi} \sum_{k=0}^{N_{i}} C_{i k} P_{k}^{(r-\xi)}\left(t^{\prime}\right) \quad \text { if: } r \geq \xi \tag{4.6}
\end{equation*}
$$

There are a total of $d \times N_{i}$ optimisation coefficients, $C_{i k}, N_{i}$ per dimension. The derivative chosen for the cost function is directly parameterised by Legendre polynomials (Eq. 4.2) to take advantage of the orthogonality property of Legendre polynomials, that is defined as:

$$
\begin{equation*}
\int_{-1}^{1} P_{i}(t) P_{j}(t) \mathrm{d} t=0 \quad \text { if } i \neq j \tag{4.7}
\end{equation*}
$$

Rescaling time from -1 to 1 allows the orthogonality property to be used to simplify the cost function, Eq. 4.1, to only consider the squared terms:

$$
\begin{equation*}
\left[f_{s}\right]_{i}=\sum_{k=0}^{N_{i}}\left\{C_{i k}^{2} \int_{-1}^{1}\left[P_{k}\left(t^{\prime}\right)\right]^{2} \mathrm{~d} t^{\prime}\right\} \tag{4.8}
\end{equation*}
$$

The $P_{k}$ integral components are standard Legendre polynomials and hence can be computed off-line, reducing the cost function calculation to a simple matrix multiplication for a given dimension:

$$
\begin{equation*}
\left[f_{s}\right]_{i}=\mathbf{C}_{i}^{T} \boldsymbol{P}_{\mathrm{int}} \mathbf{C}_{i} \tag{4.9}
\end{equation*}
$$

where the coefficients are stacked in a vector $\boldsymbol{C}_{i}$ for a the $i$ th dimension:

$$
\boldsymbol{C}_{i}=\left[\begin{array}{c}
C_{i 1}  \tag{4.10}\\
\vdots \\
C_{i N_{i}}
\end{array}\right]
$$

All the basis polynomial integrals can be pre-computed and are placed in a diagonal matrix $\boldsymbol{P}_{\text {int }}$.

$$
\boldsymbol{P}_{\text {int }}=\left[\begin{array}{cccc}
\int_{-1}^{1}\left[P_{1} \mathrm{~d} t^{\prime}\right]^{2} & 0 & \cdots & 0  \tag{4.11}\\
0 & \int_{-1}^{1}\left[P_{2} \mathrm{~d} t^{\prime}\right]^{2} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \int_{-1}^{1}\left[P_{N_{i}} \mathrm{~d} t^{\prime}\right]^{2}
\end{array}\right]
$$

The costs are added together for each dimension to give the total trajectory cost: $f_{s}=\sum_{i=1}^{d}\left[f_{s}\right]_{i}$. This trajectory cost is then added to weighted constraint costs in an augmented cost function ${ }^{1}$.

$$
\begin{equation*}
J=\sum_{i=1}^{d}\left[f_{s}\right]_{i}+\sum_{j=1}^{n_{o}} K_{j} f_{c_{j}} \tag{4.12}
\end{equation*}
$$

The coefficients $K_{j}$ represent the relative weights for each constraint function, $f_{c_{j}}$. There is flexibility in the nature of the constraint functions, as will be discussed in Section 4.3.3, nonetheless their general form is an inequality on the derivatives of a trajectory:

$$
\begin{equation*}
f_{c_{j}}\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \ddot{\boldsymbol{x}}(t), \cdots, \boldsymbol{x}^{(\xi)}(t)\right)=f_{c_{j}}(\overline{\boldsymbol{x}}(t)) \leq 0, \quad \forall t \in\left[t_{0}, t_{f}\right] \tag{4.13}
\end{equation*}
$$

where $\boldsymbol{x}$ represents the a vector of all $d$ dimensions, and $\overline{\boldsymbol{x}}$ groups all derivatives for each dimension, for ease of notation. The dimensions are grouped together for constraints because some constraints, such as obstacles and acceleration limits, will mix dimensions. For standard constraints, there is only a cost when the inequality is violated, as captured in the coefficients $K_{j}$ :

[^7]\[

K_{j}=\left\{$$
\begin{array}{l}
0, \text { if } f_{c_{j}} \leq 0  \tag{4.14}\\
W_{j}, \text { if } f_{c_{j}}>0
\end{array}
$$\right.
\]

where $W_{j}$ is the weighting for each constraint. The trajectory and all constraints can be expressed as functions of $\boldsymbol{x}(t)$ and higher derivatives (Eqs. 4.8, 4.13), that can be represented by the coefficients $C_{i k}$ (Eqs. 4.24 .4 and 4.6); therefore, the full cost function in Eq. 4.12 can be written as:

$$
\begin{align*}
J & =\sum_{j=1}^{n_{o}+1}\left[f_{j}\left(\boldsymbol{C}_{1}, \boldsymbol{C}_{2}, \cdots, \boldsymbol{C}_{d}\right)\right]  \tag{4.15}\\
& =\sum_{j=1}^{n_{o}+1}\left[f_{j}(\overline{\boldsymbol{C}})\right]
\end{align*}
$$

The $f_{j}$ represent the cost for each of the $n_{o}$ constraint functions. The $n_{o}+1$ term represents the sum of the path length cost terms $f_{n_{o}+1}=\sum_{i=1}^{d}\left[f_{s}\right]_{i}$. The vector $\overline{\boldsymbol{C}}$ is a combination of the coefficients for each dimension.

### 4.3.1 Convexity of the Cost Function

For problems without any obstacle constraints, the formulation provides a convex search space, allowing for rapid optimisation in a gradient-descent approach. This convexity is demonstrated here. For the cost function to be convex, the following must hold:

$$
\begin{equation*}
\frac{\partial^{2} J}{\partial \overline{\boldsymbol{C}}^{2}}=\sum_{j=1}^{n_{o}+1} \frac{\partial^{2} f_{j}}{\partial \overline{\boldsymbol{C}}^{2}} \geq 0 \tag{4.16}
\end{equation*}
$$

Therefore each $f_{j}$ must be convex to ensure the total cost function is convex, i.e. the Hessian has to be positive semi-definite:

$$
\begin{equation*}
\frac{\partial^{2} f_{j}}{\partial \overline{\boldsymbol{C}}^{2}} \geq 0 \tag{4.17}
\end{equation*}
$$

The first derivative of the trajectory cost with respect to a given $C_{i k}$ is given by:

$$
\begin{equation*}
\frac{\partial\left[f_{s}\right]_{i}}{\partial C_{i k}}=2 C_{i k} \int_{-1}^{1}\left[P_{k}\left(t^{\prime}\right)\right]^{2} \mathrm{~d} t^{\prime} \tag{4.18}
\end{equation*}
$$

This derivative is independent of all other $C_{i k}$ terms, hence all the off-diagonals in the Hessian for the trajectory cost are all zero. The diagonals of the Hessian are given by the second derivative with respect to each $C_{i k}$. These second derivatives are integrals of the basis polynomials squared; hence they are all positive, giving a positive definite Hessian:

$$
\begin{equation*}
\frac{\partial^{2}\left[f_{s}\right]_{i}}{\partial C_{i k}^{2}}=2 \int_{-1}^{1}\left[P_{k}\left(t^{\prime}\right)\right]^{2} \mathrm{~d} t^{\prime} \geq 0 \tag{4.19}
\end{equation*}
$$

Only performance constraints and keep-in constraints are convex, as will be described in Section 4.3.3. If the problem is defined with purely convex constraints, then a gradient-descent method will converge on the global optimal, $C_{i k}^{*}$. More specifically, for any $C_{i k} \neq C_{i k}^{*}$, $J$ can be reduced by a discrete step in $C_{i k}$ :

$$
\begin{equation*}
\left[C_{i k}\right]_{n e w}=\left[C_{i k}\right]_{o l d}+\delta C_{i k} \tag{4.20}
\end{equation*}
$$

where

$$
\begin{equation*}
\delta C_{i k}=-\alpha\left[\frac{\partial J}{\partial C_{i k}}\right] \tag{4.21}
\end{equation*}
$$

with step length $\alpha>0$. In a convex search space, the gradient-descent approach will converge to $C_{i k}^{*}$. If the search space is not convex, such as if the problem includes obstacles, then convergence to a global optimal is not assured, but local optima can be found. Techniques for optimising in non-convex search spaces will be explained in Section 4.3.6.2. ASTRO uses gradient-descent optimisation method, the details of which are elaborated in Section 4.3.6.

### 4.3.2 Boundary Conditions

For an optimal and feasible solution, it is required that the boundary conditions are complied with. The boundary conditions are equality constraints on a subset of the end positions and their derivatives for each dimension:

$$
\begin{align*}
{\left[f_{B C_{1}}\left(\gamma_{i, 0} x_{i}\left(t_{0}\right), \gamma_{i, 1} \dot{x}_{i}\left(t_{0}\right), \gamma_{i, 2} \ddot{x}_{i}\left(t_{0}\right), \cdots, \gamma_{i, q} x_{i}^{q}\left(t_{0}\right)\right)\right]_{i} } & =0  \tag{4.22}\\
{\left[f_{B C_{2}}\left(\lambda_{i, 0} x_{i}\left(t_{f}\right), \lambda_{i, 1} \dot{x}_{i}\left(t_{f}\right), \lambda_{i, 2} \ddot{x}_{i}\left(t_{f}\right), \cdots, \lambda_{i, q} x_{i}^{q}\left(t_{f}\right)\right)\right]_{i} } & =0
\end{align*}
$$

The parameters $\gamma_{i, r}$ and $\lambda_{i, r}$ are binary flags to make the constraints active or inactive, for dimensions $i=[1,2, \cdots, d]$, and derivatives $r=[0,1,2, \cdots q]$, with $q(q \geq \xi)$ being the highest derivative considered in the boundary conditions. This formulation gives the flexibility to specify a range of configurations, such as: fixing position, velocity and acceleration for all dimensions at the start and the end; leaving the $z$ dimension free; or leaving the final state free. The particular configuration depends on the application.

From the trajectory parameterization in Eq. 4.2, 4.4 and 4.6, the boundary conditions, for dimension $i$ can be written as follows

$$
\left[\boldsymbol{X}_{B C}\right]_{i}=\left[\boldsymbol{P}_{B C}\right]_{i} \boldsymbol{C}_{i}=\left[\begin{array}{c}
\boldsymbol{P}_{L}(-1)  \tag{4.23}\\
\boldsymbol{P}_{L}(1)
\end{array}\right]_{i} \boldsymbol{C}_{i}
$$

The active boundary conditions, i.e. the states that need to be matched at the start and end, are in the vector $\left[\mathbf{X}_{B C}\right]_{i}$. The matrix $\left[\boldsymbol{P}_{B C}\right]_{i}$ stores the basis Legendre polynomials corresponding to the given boundary condition, that is directly computable (with integrals for all derivatives below $\xi$ and derivatives for all above $\xi$ ). This matrix can be constructed with the $\boldsymbol{P}_{L}$ matrix, that represents the basis polynomials across each derivative for a given normalised time $t^{\prime}$. The start and end times are -1 and 1 respectively for the normalised time. The $\boldsymbol{P}_{L}$ matrices are given by:

$$
\boldsymbol{P}_{L}\left(t^{\prime}\right)=\left[\begin{array}{cccc}
\frac{1}{a^{\xi}} P_{1}^{\int_{\xi}}\left(t^{\prime}\right) & \frac{1}{a^{\xi}} P_{2}^{\int_{\xi}}\left(t^{\prime}\right) & \cdots & \frac{1}{a^{\xi}} P_{N_{i}}^{\int_{\xi}}\left(t^{\prime}\right)  \tag{4.24}\\
\vdots & \vdots & \ddots & \vdots \\
\frac{1}{a} \int P_{1}\left(t^{\prime}\right) \mathrm{d} t & \frac{1}{a} \int P_{2}\left(t^{\prime}\right) \mathrm{d} t & \cdots & \frac{1}{a} \int P_{N_{i}}\left(t^{\prime}\right) \mathrm{d} t \\
P_{1}\left(t^{\prime}\right) & P_{2}\left(t^{\prime}\right) & \cdots & P_{N_{i}}\left(t^{\prime}\right) \\
& & & \\
a P_{1}^{(1)}\left(t^{\prime}\right) & a P_{2}^{(1)}\left(t^{\prime}\right) & \cdots & a P_{N_{i}}^{(1)}\left(t^{\prime}\right) \\
\vdots & \vdots & \ddots & \vdots \\
a^{q-p} P_{1}^{(q-\xi)}\left(t^{\prime}\right) & a^{q-p} P_{2}^{(q-\xi)}\left(t^{\prime}\right) & \cdots & a^{q-p} P_{N_{i}}^{(q-\xi)}\left(t^{\prime}\right)
\end{array}\right]
$$

The first row corresponds to position and the last row the highest derivative. When constraints are not active (i.e. $\gamma_{i, r}=0$ or $\lambda_{i, r}=0$ ), then the corresponding rows in $\boldsymbol{P}_{L}(-1), \boldsymbol{P}_{L}(1)$ and $\boldsymbol{X}_{B C}$ are removed in Eq. 4.23. For example, when the cost function is the integral of velocity squared $(\xi=1)$ and all boundary conditions up to acceleration are considered ( $q=2$ ), the boundary conditions equation is:

$$
\left[\begin{array}{c}
x_{i}\left(t_{0}\right)  \tag{4.25}\\
\dot{x}_{i}\left(t_{0}\right) \\
\ddot{x}_{i}\left(t_{0}\right) \\
\\
x_{i}\left(t_{f}\right) \\
\dot{x}_{i}\left(t_{f}\right) \\
\ddot{x}_{i}\left(t_{f}\right)
\end{array}\right]=\left[\begin{array}{cccc}
\frac{1}{a} \int P_{1}(-1) \mathrm{d} t & \frac{1}{a} \int P_{2}(-1) \mathrm{d} t & \cdots & \frac{1}{a} \int P_{N_{i}}(-1) \mathrm{d} t \\
P_{1}(-1) & P_{2}(-1) & \cdots & P_{N_{i}}(-1) \\
a P_{1}^{(1)}(-1) & a P_{2}^{(1)}(-1) & \cdots & a P_{N_{i}}^{(1)}(-1) \\
& & & \\
\frac{1}{a} \int P_{1}(1) \mathrm{d} t & \frac{1}{a} \int P_{2}(1) \mathrm{d} t & \cdots & \frac{1}{a} \int P_{N_{i}}(1) \mathrm{d} t \\
P_{1}(1) & P_{2}(1) & \cdots & P_{N_{i}}(1) \\
a P_{1}^{(1)}(1) & a P_{2}^{(1)}(1) & \cdots & a P_{N_{i}}^{(1)}(1)
\end{array}\right] \boldsymbol{C}_{i}
$$

The dimensions of $\left[\boldsymbol{P}_{B C}\right]_{i}$ are $n c_{i} \times N_{i}$, where $n c_{i}$ is the number of active constraints for dimension $i$, and hence the number of equations in 4.23. $N_{i}$ is the number of unknowns for each dimension (the number of coefficients) and is also equal to the order of the polynomials plus 1 . If we set $N_{i}=n c_{i}$, then $\left[\boldsymbol{P}_{B C}\right]_{i}$ becomes square and we can uniquely determine $\boldsymbol{C}_{i}$ to satisfy the boundary conditions. More coefficients are needed to have the degrees of freedom to optimise, and comply with constraints, but this lower order solution provides a strong initial guess, which can be used to set the lower order terms of a polynomial, leaving the higher order coefficients to zero; that is:

$$
\begin{equation*}
\boldsymbol{C}_{\text {start }_{i}}=\left[\boldsymbol{P}_{\text {BC }}\right]_{\text {start }_{i}}^{-1}\left[\boldsymbol{X}_{B C}\right]_{i} \tag{4.26}
\end{equation*}
$$

From this initial guess, it is possible to assure that all future optimisation steps maintain the boundary conditions by projecting the gradient used for the steps in such a way that it does not span the space of the $n c_{i}$ degrees of freedom already determined by the boundary conditions. The gradient projection involves partitioning the $\boldsymbol{C}_{i}$ vector into components as $\boldsymbol{C}_{i}=\boldsymbol{C}_{\perp i}+\boldsymbol{C}_{\| i}$, that are defined such that

$$
\begin{equation*}
\left[\boldsymbol{P}_{B C}\right]_{i} \boldsymbol{C}_{\perp i}=0 \tag{4.27}
\end{equation*}
$$

In other words, $\boldsymbol{C}_{\perp i}$ has no influence on the result of Eq. 4.23 , which is instead all controlled by $\boldsymbol{C}_{\| i}$. Given an arbitrary gradient step in the coefficient vector $\delta \boldsymbol{C}_{i}=\delta \boldsymbol{C}_{\| i}+\delta \boldsymbol{C}_{\perp i}$, the desired component
of that step is $\delta \boldsymbol{C}_{\perp i}$, since it won't change the boundary conditions. This step can be derived from the boundary condition matrix and the step in the coefficient vector:

$$
\begin{align*}
\delta \boldsymbol{C}_{\perp i} & =\left[\boldsymbol{I}-\left[\boldsymbol{P}_{B C}\right]_{i}^{+}\left[\boldsymbol{P}_{B C}\right]_{i}\right] \delta \boldsymbol{C}_{i}  \tag{4.28}\\
& =\left[\boldsymbol{I}-\left[\boldsymbol{P}_{B C}\right]_{i}^{T}\left(\left[\boldsymbol{P}_{B C}\right]_{i}\left[\boldsymbol{P}_{B C}\right]_{i}^{T}\right)^{-1}\left[\boldsymbol{P}_{B C}\right]_{i}\right] \delta \boldsymbol{C}_{i}
\end{align*}
$$

$\left[\boldsymbol{P}_{B C}\right]_{i}^{+}$is the Moore-Penrose pseudo-inverse, which, for the optimisation, can be pre-computed for computational efficiency. A full derivation of this projection is presented in Appendix B. Eq. 4.28 gives a method to project an arbitrary gradient onto the subspace that assures the boundary conditions are met. In a gradient-descent optimisation this projection can be used on the computed step, $\partial \boldsymbol{C}_{i}$, to give $\partial \boldsymbol{C}_{\perp i}$. Each iteration of the optimisation uses $\delta \boldsymbol{C}_{\perp i}$ and hence complies with the boundary conditions. Using this subspace-projection method the parameter adjustments for dimension $i$ are now limited to the $\left(N_{i}-n c_{i}\right)$ degrees of freedom remaining.

In selecting the boundary conditions and the order of the polynomials for a given problem, the degrees of freedom left available to optimise should be considered. There needs to be at at least one more coefficient than the number of boundary conditions to have any freedom to optimise ( $N_{i}>n c_{i}$ ). For example, if all constraints are active at the start and the end of a trajectory, up to acceleration, then $n c_{i}=6$, meaning 9 th order polynomials ( $N_{i}=10$ ) will give four degrees of freedom to optimise. Higher order polynomials could be used, yet there is a trade-off between having degrees of freedom to help find an optimal solution and the complexity of the search space when working with many coefficients.

The same projection method can be applied to a set of coefficients to project them onto the subspace of feasible solutions. Similarly to the projection for gradient steps, this uses the concept of splitting $\boldsymbol{C}$ into parts: $\boldsymbol{C}=\boldsymbol{C}_{\perp}+\boldsymbol{C}_{\| \mid}$, where $\left[\boldsymbol{P}_{B C}\right]_{i} \boldsymbol{C}_{\perp i}=\left[\boldsymbol{X}_{B C}\right]_{i}$, i.e. $\boldsymbol{C}_{\perp i}$ is the component of $\boldsymbol{C}_{i}$ that complies with the boundary conditions. This projected component can be computed with:

$$
\begin{equation*}
\boldsymbol{C}_{\perp i}=\boldsymbol{C}_{i}-\left[\boldsymbol{P}_{B C}\right]_{i}^{+}\left(\left[\boldsymbol{P}_{B C}\right]_{i} \boldsymbol{C}_{i}-\left[\boldsymbol{X}_{B C}\right]_{i}\right) \tag{4.29}
\end{equation*}
$$

A derivation for the coefficient projection is presented in Appendix B. Projecting the coefficient set, rather than the gradient, comes into use when replanning trajectories. When there are changes to the start or end states, or the trajectory time is updated, an existing solution can be modified to comply with the new boundary conditions, giving a feasible initial seed for the replanning that is close to the current trajectory. The projection is also useful when perturbing a solution to explore a different homotopy: a random coefficient set can have the boundary conditions enforced. See Section 4.3.7 for more details on replanning and Section 4.3.6.2 for details on randomised perturbations.

By parameterising the trajectory with Legendre polynomials, starting the optimisation with an initial guess that meets the boundary conditions, and enforcing those conditions by a projected gradient, the ASTRO algorithm is capable of quickly finding solutions that optimise the trajectory cost and meet all boundary conditions. The algorithm can also quickly find solutions that satisfy constraints, the nature of which will be discussed in the next section.

### 4.3.3 Obstacles and Performance Constraints

The core requirements for the constraint cost functions, $f_{c_{j}}$, are to:

- Provide a cost for a given trajectory, $\boldsymbol{x}(t)$ and its derivatives.
- Provide a cost gradient for the trajectory states.
- Have a convex cost profile, i.e. the maximum cost is with the greatest violation, and the cost decreases all around to the boundary of the constraint.

Ideally, the gradient can be computed analytically for fast computation. It is also useful if the curvature can be computed analytically, to help inform the gradient step (discussed in Section 4.3.6.1). As outlined in Eq. 4.13, constraints can operate on any or all of the derivatives, allowing geometric constraints, such as obstacles, or corridors, as well as performance constraints, such as limits on velocity and acceleration, to be specified within the same framework.

There are two classes of constraints: keep-in and keep-out. Keep-in constraints, such as cylindrical, physical corridors, or spherical bounds on maximum acceleration, will push the trajectory to remain inside the constraint. Keep-out constraints, primarily, are obstacles and push the trajectory outside of the constraint. The formulations of these two classes differ by a change in the sign of the cost and gradient.

The gradient of a constraint function with respect to the optimisation coefficients, $C_{i k}$, is computed with the chain rule:

$$
\begin{equation*}
\frac{\partial f_{c_{j}}}{\partial C_{i k}}=\frac{\partial f_{c_{j}}}{\partial x_{i}^{(r)}} \frac{\partial x_{i}^{(r)}}{\partial C_{i k}} \tag{4.30}
\end{equation*}
$$

The derivative of the state with respect to the coefficients comes from the parameterization of the trajectory with Legendre polynomials (Eqs. 4.2, 4.4 and 4.6), and is given by:

$$
\frac{\partial x_{i}^{(r)}}{\partial C_{i k}}= \begin{cases}\frac{1}{a^{(\zeta-r)}} P_{k}^{\int_{(\xi-r)}}\left(t^{\prime}\right) & \text { if: } r<\xi  \tag{4.31}\\ a^{r-1} P_{k}^{(r-1)}\left(t^{\prime}\right) & \text { if: } r \geq \xi\end{cases}
$$

where $P^{(0)}$ is the zeroth derivative, simply $P$. The other term in Eq. 4.30, $\frac{\partial f_{c_{j}}}{\partial x_{s}^{(r)}}$, is the cost function gradient with respect to a given derivative of the trajectory. This gradient needs to be computed, ideally analytically, from the given constraint cost function. These cost functions can operate separately on individual dimensions or can mix dimensions, such as for a spherical obstacle constraint, and a spherical bound on acceleration. Examples will be presented for a range of constraints in Section 4.3.4.

### 4.3.3.1 Convexity of Constraint Functions

Convex constraint functions require $\frac{\partial^{2} f_{c_{j}}}{\partial \overline{\boldsymbol{C}}^{2}} \geq 0$, which, expanding with the chain rule to consider derivatives with respect to the trajectory, gives:

$$
\begin{align*}
\frac{\partial^{2} f_{c_{j}}}{\partial \overline{\boldsymbol{C}}^{2}} & =\left(\frac{\partial}{\partial \overline{\boldsymbol{C}}} \frac{\partial f_{c_{j}}}{\partial \boldsymbol{x}^{(r)}}\right)^{T} \frac{\partial \boldsymbol{x}^{(r)}}{\partial \overline{\boldsymbol{C}}}+\left(\frac{\partial f_{c_{j}}}{\partial \boldsymbol{x}^{(r)}}\right)^{T} \frac{\partial^{2} \boldsymbol{x}^{(r)}}{\partial \overline{\boldsymbol{C}}^{2}}  \tag{4.32}\\
& =\frac{\partial^{2} f_{c_{j}}}{\partial \boldsymbol{x}^{(r)^{2}}}\left(\frac{\partial \boldsymbol{x}^{(r)}}{\partial \overline{\boldsymbol{C}}}\right)^{2} \tag{4.33}
\end{align*}
$$

where the simplification to get Eqn. 4.33 uses the fact that $\frac{\partial^{2} \boldsymbol{x}^{(r)}}{\partial \overline{\boldsymbol{C}}^{2}}=0$, which can be seen by observing that Eq. 4.31 is not a function of $C_{i k}$. Given that the right term in Eq. 4.33 is squared (hence $\geq 0$ ), the only remaining requirement is that $\frac{\partial^{2} f_{c_{j}}}{\partial \boldsymbol{x}^{(r)^{2}}} \geq 0$ : that the constraint function is convex with respect to the trajectory.

Satisfying the convexity requirements will be explored with some examples here. A simple keep-out constraint is a spherical obstacle, which can be represented by:

$$
\begin{equation*}
f_{c}\left(\boldsymbol{x}\left(t^{\prime}\right)\right)=1-\frac{\left\|\boldsymbol{x}\left(t^{\prime}\right)-\boldsymbol{x}_{\text {sphere }}\right\|^{2}}{r_{\text {sphere }}^{2}} \tag{4.34}
\end{equation*}
$$

The sphere is defined by radius $r_{\text {sphere }}$ and centre $\boldsymbol{x}_{\text {sphere }}$. The second derivative of this function for each dimension is:

$$
\begin{equation*}
\frac{\partial^{2} f_{c}}{\partial x_{i}\left(t^{\prime}\right)^{2}}=-\frac{2}{r_{\text {sphere }}^{2}}<0 \tag{4.35}
\end{equation*}
$$

with all off-diagonals in the Hessian being zero: $\frac{\partial^{2} f_{c}}{\partial x_{i} \partial x_{j}}=0, i \neq j$. Therefore, with a diagonal matrix for a Hessian, and all diagonal terms negative; hence the constraint is non-convex. This Hessian conditioning is a similar situation for all keep-out constraints (obstacles): the cost functions are non-convex and will cause the search space to be non-convex. Nonetheless, the optimisation can still be run effectively to quickly find local minima, as described in Section 4.3.6.2.

In contrast, keep-in constraints can be convex, such as a limit on maximum acceleration:

$$
\begin{align*}
f_{c}\left(\ddot{\boldsymbol{x}}\left(t^{\prime}\right)\right) & =\frac{\left\|\ddot{\boldsymbol{x}}\left(t^{\prime}\right)\right\|^{2}}{r_{\text {sphere }}^{2}}-1  \tag{4.36}\\
\frac{\partial^{2} f_{c}}{\partial \ddot{x}_{i}\left(t^{\prime}\right)^{2}} & =\frac{2}{r_{\text {sphere }}^{2}} \geq 0 \tag{4.37}
\end{align*}
$$

The off-diagonals in the Hessian are again zero, giving a diagonal matrix with all diagonal terms positive: a positive-definite Hessian. Therefore the cost function is convex.

For problems with only keep-in constraints, additional techniques can be used to speed-up optimisation, as explored in Section 4.3.6.1.

### 4.3.3.2 Constraint Function Methods

To assess the cost of violation of a constraint, the trajectory is discretised in time, $t_{i}^{\prime} \in[-1,1]$, and the constraint function evaluated at each sample. There are $n_{\text {samp }}$ samples evenly from -1 to 1 , that is, using MATLAB notation, $t_{i}^{\prime}=\operatorname{linspace}\left(-1,1, n_{s a m p}\right)$. Two different methods can be used to extract the final cost from these samples: maximum violation, or approximate-path-integral. The maximum violation method uses only the sample point with the largest cost for both the cost and the gradient, i.e.:

$$
\begin{equation*}
\left[f_{c_{j}}\left(\overline{\boldsymbol{x}}\left(t^{\prime}\right)\right)\right]_{\max }=\max _{t_{i}^{\prime} \in[-1,1]} f_{c_{j}}\left(\overline{\boldsymbol{x}}\left(t_{i}^{\prime}\right)\right) \tag{4.38}
\end{equation*}
$$

$\overline{\boldsymbol{x}}(t)$ is used again here to represent all derivatives of the trajectory. The second method is to use an approximate path-integral by adding together cost and gradient components from each sample point, giving:

$$
\begin{equation*}
\left[f_{c_{j}}\left(\overline{\boldsymbol{x}}\left(t^{\prime}\right)\right)\right]_{\text {path }}=\sum_{t_{i}^{\prime}=-1}^{1} f_{c_{j}}\left(\overline{\boldsymbol{x}}\left(t_{i}^{\prime}\right)\right) \tag{4.39}
\end{equation*}
$$

with the gradient also requiring a summation:

$$
\begin{equation*}
\frac{\partial f_{c_{j}}}{\partial C_{i k}}=\sum_{t_{i}^{\prime}=-1}^{1}\left[\frac{\partial f_{c_{j}}}{\partial x_{i}^{(r)}\left(t_{i}^{\prime}\right)} \frac{\partial x_{i}^{(r)}\left(t_{i}^{\prime}\right)}{\partial C_{i k}}\right] \tag{4.40}
\end{equation*}
$$

The maximum violation method is quick, simple and effective, but does not consider constraint influence on the entire trajectory, which the approximate-path-integral does. The better method depends on the problem. For problems with simple keep-out constraints, the maximum violation method is preferred. For more complex scenarios with many local minima, and where more of the trajectory could be in violation, the approximate-path-integral method is preferred. Results comparing the two methods are presented in Section 4.4.4.4.

### 4.3.3.3 Constraint Cost Weighting

Each of the constraint functions for a given problem has an associated weighting (Eq. 4.14). The weighting on the obstacle constraint has a large influence on the optimisation performance, with too large a weighting causing unstable large jumps in the solution and too small a weighting causing a slow convergence.

For problems with simple obstacle constraints, using the maximum violation cost method, a fixed weighting at a high value (e.g. $10^{5}$ ) works well to encourage rapid progression to a feasible (violation free) solution. This approach works because a large jump is permissible, to push the trajectory out of any violations. For problems with many constraints, the approximate-path-integral cost method, such an approach is not suitable. Instead, a low weight (e.g. $10^{-5}$ ), or more controlled tuning, is required. Techniques to automatically tune the weighting are presented in Section 4.3.6.4.

As outlined in Eq. 4.14, the weighting, and hence the constraint cost are zero when there is no violation. This behaviour can be modified to have a different performance with the cost functions. For keep-in constraints, a negative cost could be allowed to continue to push a trajectory towards the middle of the constraint, e.g. the middle of a corridor, or to zero acceleration. The benefit of allowing negative
costs is that the cost functions remain continuous throughout, rather than having discontinuous transitions to zero cost. A downside is that it is possible for the cost to be negative when there is still a violation, for example, if one part of the trajectory is in a slight violation, but the rest of the trajectory is feasible. In this case, the sign of the cost cannot be used to indicate the feasibility, and a separate check is required.

For tests cases using a maximum violation cost method, the approach is taken to switch to zero cost when there is no violation. When using the approximate-path-integral cost method, negative costs are included, with separate feasibility checks.

### 4.3.4 Example Constraint Cost Functions

Four example constraint functions are presented here. These are the constraint functions that will be used throughout the work presented here, yet ASTRO is not limited to just these functions. Each of the constraints presented is designed to work on a mix of $x, y$ and $z$ dimensions.

### 4.3.4.1 Ellipsoid Constraints

The first constraint function is an ellipsoid, which can be use as a general obstacle, or as a keep-in constraint on velocity and acceleration. The ellipsoid is defined by centre, $\boldsymbol{x}_{c}$ and a shape matrix, $\mathbf{A}$, that incorporates information on the size of each of the three axes and the orientation of the ellipsoid. The shape matrix can be intuitively defined from a rotation matrix, $\mathbf{R}$, giving the orientation, and a diagonal matrix of inverse square of the axes, $r_{1}, r_{2}, r_{3}$ :

$$
\mathbf{A}=\mathbf{R}^{T}\left[\begin{array}{ccc}
\frac{1}{r_{1}^{2}} & 0 & 0  \tag{4.41}\\
0 & \frac{1}{r_{2}^{2}} & 0 \\
0 & 0 & \frac{1}{r_{3}^{2}}
\end{array}\right] \mathbf{R}
$$

Note here that $\mathbf{A} \geq 0$, as it is a rotation of a diagonal matrix of squared terms. A sphere would have $r_{1}=r_{2}=r_{3}$, and $\mathbf{R}$ as an identity matrix, making $\mathbf{A}$ a uniform diagonal matrix. Using the shape matrix, an ellipsoid, centred around the origin is defined as:

$$
\begin{equation*}
\boldsymbol{x}^{T} \mathbf{A x}=1 \tag{4.42}
\end{equation*}
$$

An example ellipsoid is in Fig. 4.2. To evaluate the cost for an ellipsoid constraint, the trajectory point being evaluated, $\boldsymbol{x}^{(r)}\left(t_{i}\right)$, is adjusted to be relative to the centre of the ellipsoid, to give $\tilde{\boldsymbol{x}}^{(r)}\left(t_{i}\right)=$ $\boldsymbol{x}^{(r)}\left(t_{i}\right)-\boldsymbol{x}_{c}$. The cost is then computed with:

$$
\begin{align*}
{\left[f_{c}\right]_{\text {ellip }} } & =w_{i o}\left(\tilde{\boldsymbol{x}}^{(r)}\left(t_{i}\right)^{T} \mathbf{A} \tilde{\boldsymbol{x}}^{(r)}\left(t_{i}\right)-1\right)  \tag{4.43}\\
w_{i o} & =\left\{\begin{array}{l}
1, \text { for keep-in constraints } \\
-1, \text { for keep-out constraints }
\end{array}\right. \tag{4.44}
\end{align*}
$$

where $w_{i o}$ sets the constraint class as keep-in or keep-out, so that a positive $f_{c}$ denotes a violation. This cost function needs to be evaluated for each sample point. The gradient for the ellipsoid is then:


Figure 4.2. An example general ellipsoid with centroid, axes lengths, and a rotation from the coordinate axes

$$
\begin{equation*}
\left[\frac{\partial f_{c}}{\partial \boldsymbol{x}\left(t_{i}\right)}\right]_{\mathrm{ellip}}=2 w_{i o} \mathbf{A} \tilde{\boldsymbol{x}}^{(r)}\left(t_{i}\right) \tag{4.45}
\end{equation*}
$$

and the curvature:

$$
\begin{equation*}
\left[\frac{\partial^{2} f_{c}}{\partial \boldsymbol{x}\left(t_{i}\right)^{2}}\right]_{\text {ellip }}=2 w_{i o} \mathbf{A} \tag{4.46}
\end{equation*}
$$

Eq. 4.46 shows that the ellipsoid constraint function is convex for keep in constraints, when $w_{i o}=1$, and non-convex for keep-out constraints, when $w_{i o}=-1$.

In addition to being used as obstacles and performance constraints, ellipsoid constraints could also be specified for a specific point on the trajectory, such as a keep-in constraint on the final position. Such a capability is useful when the final position is free from boundary constraints.

### 4.3.4.2 Cylindrical Constraints

Cylindrical constraints are used to represent corridors of free-space, and as an obstacle representation for long slender objects, such as poles, trees and components of a space station. A cylinder is defined with three different sections: a cylindrical body and two end-caps. The end-caps are defined as half-ellipsoids, fitting the circular end of the cylinder, and extending out a user-defined distance: $l_{\text {cyl }}$. The cylinder is defined by its radius $r_{\text {cyl }}$ and the location of the end-points of the central axis, $\boldsymbol{x}_{1}$ and $\boldsymbol{x}_{2}$.

For a given trajectory point, $\boldsymbol{x}^{(\xi)}\left(t_{i}\right)$ (we will now drop the time argument and possible derivatives here for simplicity), the first step is to determine which segment it is in. Referring to Fig. 4.3, this is done by taking the dot product of the vector from $\boldsymbol{x}$ to the end-points: $\boldsymbol{x}_{1 x}=\boldsymbol{x}-\boldsymbol{x}_{1}$, and $\boldsymbol{x}_{1 x}=\boldsymbol{x}-\boldsymbol{x}_{1}$, with the vector between the end-points: $\boldsymbol{x}_{12}=\boldsymbol{x}_{2}-\boldsymbol{x}_{1}$ :

$$
\begin{align*}
\boldsymbol{x}_{12} \cdot \boldsymbol{x}_{1 x} & =\left|\boldsymbol{x}_{12}\right|\left|\boldsymbol{x}_{1 x}\right| \cos \left(\theta_{1}\right)  \tag{4.47}\\
-\boldsymbol{x}_{12} \cdot \boldsymbol{x}_{2 x} & =\left|\boldsymbol{x}_{12}\right|\left|\boldsymbol{x}_{2 x}\right| \cos \left(\theta_{2}\right) \tag{4.48}
\end{align*}
$$



Figure 4.3. Cylinder constraint in blue and elements used to assess the cost for point $\boldsymbol{x}$, including determining the distance $d_{\text {cyl }} . \boldsymbol{x}_{1}$ and $\boldsymbol{x}_{2}$ define the end-points of the cylinder, and $\boldsymbol{x}_{1 x}$ and $\boldsymbol{x}_{2 x}$ are the vectors from the end-points to the point of interest. $\boldsymbol{x}_{12}$ is the vector between the end-points. The radius of the cylinder is $r_{\text {cyl }}$.

If the sign of the dot product is less than zero then the corresponding angle, $\theta_{1}$ or $\theta_{2}$ is greater than $90^{\circ}$, which can be used to determine which region the point is in:

$$
\text { region }= \begin{cases}\text { end ellipsoid } 1, & \text { if : } \theta_{1}<90 \circ  \tag{4.49}\\ \text { end ellipsoid } 2 & \text { if: } \theta_{2}>90 \circ \\ \text { cylinder } & \text { otherwise }\end{cases}
$$

If the point is in the region for the end-caps then the cost is computed as described above for ellipsoids, with the shape matrix specified as:

$$
\mathbf{A}_{c a p}=\mathbf{R}^{T}\left[\begin{array}{ccc}
\frac{1}{r_{\mathrm{cyl}}^{2}} & 0 & 0  \tag{4.50}\\
0 & \frac{1}{r_{\mathrm{cyl}}^{2}} & 0 \\
0 & 0 & \frac{1}{l_{\mathrm{cyl}}^{2}}
\end{array}\right] \mathbf{R}
$$

$\mathbf{R}$ is the rotation of the cylinder from being aligned with the global $z$ axis and is extracted from the locations $\boldsymbol{x}_{1}$ and $\boldsymbol{x}_{2}$ using Rodrigues' rotation formula [15]:

$$
\begin{equation*}
\mathbf{R}=\boldsymbol{I}-\sin (\phi) \hat{\mathbf{e}}+(1-\cos (\phi)) \hat{\mathbf{e}} \hat{\mathbf{e}} \tag{4.51}
\end{equation*}
$$

where $\hat{\mathbf{e}}$ is a cross product matrix of the unit vector $\mathbf{e}$ about which $\boldsymbol{x}_{12}$ is rotated by angle $\phi$. $\hat{\mathbf{e}}$ is defined by the hat operator:

$$
\mathbf{a} \times \mathbf{b}=\hat{\mathbf{a}} \mathbf{b}=\left[\begin{array}{ccc}
0 & -a_{3} & a_{2}  \tag{4.52}\\
a_{3} & 0 & -a_{1} \\
-a_{2} & a_{1} & 0
\end{array}\right] \mathbf{b}
$$

This vector of rotation, $\mathbf{e}$ and the rotation angle $\phi$ are given by:

$$
\begin{align*}
\mathbf{e} & =\frac{\boldsymbol{x}_{12} \times \boldsymbol{z}_{g}}{\left|\boldsymbol{x}_{12} \times \boldsymbol{z}_{g}\right|}  \tag{4.53}\\
\phi=\arctan \left(\frac{\left|\boldsymbol{x}_{12} \times \boldsymbol{z}_{g}\right|}{\boldsymbol{x}_{12}}\right) & =\arctan \left(\frac{\left|\boldsymbol{x}_{12}\right| \sin (\phi)}{\left|\boldsymbol{x}_{12}\right| \cos (\phi)}\right) \tag{4.54}
\end{align*}
$$

For the cylindrical section, the cost function is based on the perpendicular distance from the axis of the cylinder, $d_{\text {cyl }}$ :

$$
\begin{equation*}
\left[f_{c}(\boldsymbol{x})\right]_{\mathrm{cyl}}=w_{i o}\left(d_{\mathrm{cyl}}^{2}-r_{\mathrm{cyl}}^{2}\right) \tag{4.55}
\end{equation*}
$$

with $w_{i o}$ the same as defined for ellipsoids in Eq. 4.44. Referring again to Fig. 4.3, $d_{\text {cyl }}=\left|\boldsymbol{x}_{1 x}\right| \sin \left(\theta_{1}\right)$. To compute $d_{\mathrm{cyl}}^{2}$, the magnitude of the cross product $\left|\boldsymbol{x}_{12} \times \boldsymbol{x}_{1 x}\right|=\left|\boldsymbol{x}_{12}\right|\left|\boldsymbol{x}_{1 x}\right| \sin \left(\theta_{1}\right)$ is used:

$$
\begin{equation*}
d_{\mathrm{cyl}}^{2}=\left|\boldsymbol{x}_{1 x}\right|^{2} \sin ^{2}\left(\theta_{1}\right)=\frac{\left|\boldsymbol{x}_{12} \times \boldsymbol{x}_{1 x}\right|^{2}}{\left|\boldsymbol{x}_{12}\right|^{2}} \tag{4.56}
\end{equation*}
$$

To give a more compact expression, the cross product can be represented by the hat operator, giving:

$$
\begin{equation*}
\left|\boldsymbol{x}_{12} \times \boldsymbol{x}_{1 x}\right|^{2}=\left(\hat{\boldsymbol{x}}_{12} \boldsymbol{x}_{1 x}\right)^{T}\left(\hat{\boldsymbol{x}}_{12} \boldsymbol{x}_{1 x}\right)=\boldsymbol{x}_{1 x}^{T} \hat{\boldsymbol{x}}_{12}^{T} \hat{\boldsymbol{x}}_{12} \boldsymbol{x}_{1 x}=\boldsymbol{x}_{1 x}^{T} \mathbf{A}_{\mathrm{cy} 1} \boldsymbol{x}_{1 x} \tag{4.57}
\end{equation*}
$$

where the term $\mathbf{A}_{\mathrm{cyl}}=\hat{\boldsymbol{x}}_{12}^{T} \hat{\boldsymbol{x}}_{12}$ is introduced here. The cost function is then given by:

$$
\begin{equation*}
\left[f_{c}(\boldsymbol{x})\right]_{\mathrm{cyl}}=w_{i o}\left(\frac{\boldsymbol{x}_{1 x}^{T} \mathbf{A}_{\mathrm{cyl}} \boldsymbol{x}_{1 x}}{\left|\boldsymbol{x}_{12}\right|^{2}}-r_{\mathrm{cyl}}^{2}\right) \tag{4.58}
\end{equation*}
$$

The gradient is:

$$
\begin{equation*}
\left[\frac{\partial f_{c}}{\partial \boldsymbol{x}\left(t_{i}\right)}\right]_{\mathrm{cyl}}=\left(\frac{2 w_{i o}}{\left|\boldsymbol{x}_{12}\right|^{2}}\right) \mathbf{A}_{\mathrm{cy} 1} \boldsymbol{x}_{1 x} \tag{4.59}
\end{equation*}
$$

and the curvature is:

$$
\begin{equation*}
\left[\frac{\partial^{2} f_{c}}{\partial \boldsymbol{x}\left(t_{i}\right)^{2}}\right]_{\mathrm{cyl}}=\frac{2 w_{i o}}{\left|\boldsymbol{x}_{12}\right|^{2}} \mathbf{A}_{\mathrm{cyl}} \tag{4.60}
\end{equation*}
$$

For a positive $w_{i o}$ (keep-in constraints), the curvature of the cylinder constraint satisfies the requirements for a convex function, with the matrix $A_{\text {cyl }}$ being positive semi-definite, coming from the transpose self-multiplication of a real valued matrix: $\hat{\boldsymbol{x}}_{12}^{T} \hat{\boldsymbol{x}}_{12}$ [178].

Cylinders, as keep-in corridors, provide a convex method of restricting a trajectory to free-space. For complex environments, there can be much benefit in representing free-space with these convex constraints, rather than with obstacles, which make the problem non-convex.

### 4.3.4.3 Rectangular Prism Constraints

Rectangular prism constraints are defined by a centroid, $\boldsymbol{x}_{c}$ and the length of each side, $r_{i}$. In the simplest form this can be independent inequalities on each dimension, each with separate cost, gradient and curvature:

$$
\begin{align*}
& {\left[f_{c}\left(x_{i}\right)\right]_{\text {rect }}=w_{i o}\left(\frac{\left(x_{i}-x_{c_{i}}\right)^{2}}{r_{i}^{2}}-1\right)}  \tag{4.61}\\
& {\left[\frac{\partial f_{c}}{\partial x_{i}}\right]_{\text {rect }}=2 w_{i o}\left(\frac{\left(x_{i}-x_{c_{i}}\right)}{r_{i}^{2}}\right)}  \tag{4.62}\\
& {\left[\frac{\partial^{2} f_{c}}{\partial x_{i}^{2}}\right]_{\text {rect }}=\frac{2 w_{i o}}{r_{i}^{2}}} \tag{4.63}
\end{align*}
$$

The cost is only active if all dimensions are in violation. This formulation gives a convex function for keep-in constraints and can be used to define bounds on the region of operation, to define walls, or as obstacles from an occupancy grid.


Figure 4.4. An example rectangular prism constraint, with centroid, $\boldsymbol{x}_{c}$, side lengths, $r_{i}$ and a general rotation from the coordinate axes.

If the rectangular prism is able to rotate, defined by rotation matrix $\mathbf{R}$, then the dimensions become mixed (for example, see Fig. 4.4). In this case, when evaluating the cost for a point on the trajectory, that point is first rotated from a global frame, $g$, into the body frame, $b$, of the prism: $\boldsymbol{x}_{b}=\mathbf{R} \boldsymbol{x}_{g}$. Working in the body frame, each dimension is independently checked for violation, with all required to be in violation for the constraint to be active. If there are violations, then the cost is computed as a product of each individual costs:

$$
\begin{equation*}
\left[f_{c}\left(x_{i}\right)\right]_{\mathrm{rect}}=w_{i o} \prod_{i=1}^{3}\left(\frac{\left(x_{i}-x_{c i}\right)^{2}}{r_{i}^{2}}-1\right)=w_{i o}(\breve{x} \breve{y} \breve{z}) \tag{4.64}
\end{equation*}
$$

The notation $\breve{x}$ is used here to represent the cost component for dimension $x$. The function is designed so that, from anywhere in the prism, the cost smoothly decreases (for a keep-out constraint) to zero cost at the boundary. The gradient of this cost function needs to take into account the rotation, hence it is computed in the body frame then transformed back into the global frame:

$$
\left[\frac{\partial f_{c}}{\partial \boldsymbol{x}}\right]_{\text {rect }}=2 w_{i o} \mathbf{R}^{T}\left[\begin{array}{c}
\left(x-x_{c}\right) \breve{y} \breve{z}  \tag{4.65}\\
\left(y-y_{c}\right) \breve{x} \breve{z} \\
\left(z-z_{c}\right) \breve{x} \check{y}
\end{array}\right]
$$

The curvature does not provide a convex constraint in general, but is presented here for reference. Further developments could look to find a convex formulation for a general rectangular prism constraint.

$$
\left[\frac{\partial^{2} f_{c}}{\partial \boldsymbol{x}^{2}}\right]_{\text {rect }}=2 w_{i o} \mathbf{R}^{T}\left[\begin{array}{ccc}
\breve{y} \breve{z} & 2\left(x-x_{c}\right)\left(y-y_{c}\right) \breve{z} & 2\left(x-x_{c}\right)\left(z-z_{c}\right) \breve{y}  \tag{4.66}\\
2\left(x-x_{c}\right)\left(y-y_{c}\right) \breve{z} & \breve{x} \breve{z} & 2\left(y-y_{c}\right)\left(z-z_{c}\right) \breve{x} \\
2\left(x-x_{c}\right)\left(z-z_{c}\right) \breve{y} & 2\left(y-y_{c}\right)\left(z-z_{c}\right) \breve{x} & \breve{x} \breve{y}
\end{array}\right]
$$

A rectangular prism obstacle could be used to represent a physical obstacle more accurately and could be combined with cylinders and ellipsoids to make a composite shape to represent a physical obstruction.

### 4.3.4.4 Euclidean Signed Distance Field Constraints

Each of the constraints mentioned above, if used to represent obstacles, are for a distinct object, with multiple such objects needed to represent a complete environment. If instead it is desired to have a single representation, then a Euclidean Signed Distance Field (ESDF) can be used. An ESDF is a voxel grid representation of an environment where each voxel in the grid gives the Euclidean signed distance to the nearest obstacle (see Fig. 4.5). The distance is negative inside an obstacle, zero on the surface and positive in free-space. By using the negative of the signed distance as the cost, the ESDF gives a natural gradient of cost to push a trajectory away from collisions. By taking distance samples in the vicinity of a query location, the cost gradient can be evaluated numerically.


Figure 4.5. Example of a slice of an ESDF (coloured cells) in with a 3D mesh (grey). Red blocks are inside obstacles, purple blocks are near the surface in free-space and blue blocks are further into free-space.

An advantage of the ESDF representation is that the maximum cost for a point on the trajectory can be sampled directly, rather than having to search through a large set of objects to find the object that is being violated.

An ESDF represents obstacles, and hence is inherently non-convex, can have many corners and many local minima. Costs are allowed to go to negative values, and the approximate-path-integral cost method is used, to help push a trajectory into free-space.

For efficient ESDF representation and querying, the open source Voxblox library [171] is used. When working with ESDF constraints, the spacing of the trajectory samples needs to be small enough to ensure that all voxels along a trajectory are queried when checking the constraint.

### 4.3.4.5 Discussion - Constraint Formulations

A benefit of ASTRO is the ability to mix the range of constraints discussed above, e.g. to have an ESDF to represent the obstacle presented by the environment, with the addition of ellipsoids, cylinders or prisms to represent newly observed or dynamic obstacles, and spherical ellipsoids to limit acceleration and velocity. Adding many constraints, though, makes the optimisation problem more difficult to solve efficiently. Techniques to aid in optimisation with multiple constraints are discussed in Section 4.3.6.

### 4.3.5 Dynamic Obstacles

Attribution: Theory presented in this paper is adapted from that presented in [155] by the author of the work presented here.

Dynamic obstacles can be represented in the same constraint formulation by taking advantage of the fact that the algorithm checks constraints at discrete values of time along the trajectory. To make a constraint dynamic, the parameters that describe the constraint can be represented as functions of time, changing where the constraint lies for different points in time along the trajectory. The use of dynamic constraints is most applicable to keep-out, obstacle constraints. For example, the centre of an ellipsoid, $\boldsymbol{x}_{c}$ would become a function of time, $\boldsymbol{x}_{c}(t)$. The function could be a pre-defined trajectory, in the case where the path of an obstacle is known. Alternatively, a motion model could define the function, such as a constant velocity model, where observations of a moving obstacle define the model parameters. An example, constant velocity model is:

$$
\begin{equation*}
\boldsymbol{x}_{c}(t)=\boldsymbol{x}_{c}\left(t_{o b s}\right)+\boldsymbol{v}_{c}\left(t-t_{o b s}\right) \tag{4.67}
\end{equation*}
$$

where $\boldsymbol{v}_{c}$ is an observed velocity at time $t_{o b s}$. When computing the cost for a constraint at time $t$, Eq. 4.67 is used to compute centre, to then be used in the cost function, such as Eq. 4.43 for ellipsoids. This motion model can then be updated when replanning, for instance after the robot makes a new observation of the obstacle.

The orientation of the obstacles can also be a function of time, by defining an attitude trajectory with quaternions, $\mathbf{q}_{c}(t)$. The quaternion at the given time is used to generate the rotation matrix $\mathbf{R}$, to be used in defining the $\mathbf{A}$ matrix for ellipsoids (Eq. 4.41), and in the cost functions for prisms (Eq. 4.64). See Appendix C. 10 for details on methods to propagate forward quaternion attitude dynamics if an initial observation of angular rates was made.

For cylinders, the end-points need to be transformed with the rotation as well as the translation. To have a dynamic cylinder, a centroid $\left(\boldsymbol{x}_{c}=\left(\boldsymbol{x}_{1}+\boldsymbol{x}_{2}\right) / 2\right)$ and rotation matrix are needed, and made to be functions of time as above. The end-points are then computed from these values, by rotating the vector from the centre to the end points:

$$
\begin{equation*}
\boldsymbol{x}_{1}(t)=\boldsymbol{x}_{c}(t)+\mathbf{R}\left(\boldsymbol{x}_{1}\left(t_{0}\right)-\boldsymbol{x}_{c}\left(t_{0}\right)\right) \tag{4.68}
\end{equation*}
$$

and similarly for $\boldsymbol{x}_{2}$. The cylinder cost can then be evaluated as in Eq. 4.58. For the end-caps, the shape matrix, $\mathbf{A}$, also needs to be transformed, in the same manner as for rotating ellipsoids (Eq. 4.41).

## Dynamic Uncertainty

If observations are used to model the dynamics of an obstacle, then there is an advantage in being able to account for uncertainty in the predicted motion. The uncertainty can be modelled by having the radii or axes sizes, $r_{i}$, as increasing functions of time. The growth in the radii increases the volume of occupied space to encompass all likely positions, given the uncertainty in the motion model and observations (see Fig. 4.6).

With the use of a constant velocity model, the growth of the radius of an obstacle would be represented with $3 \sigma$ values of uncertainty on position, $x_{u}$, velocity, $v_{u}$, and acceleration, $a_{u}$ (as a deviation from the model predicted zero). A example of the radius growth with this uncertainty model for a sphere is:

$$
\begin{equation*}
r(t)=r_{o b s}+x_{u}+\left(t-t_{o b s}\right) v_{u}+0.5\left(t-t_{o b s}\right)^{2} a_{u} \tag{4.69}
\end{equation*}
$$

where $x_{u}, v_{u}$ and $a_{u}$ are all positive scalar values giving the maximum magnitude of uncertainty across dimensions, and $r_{o b s}$ is the observed radius.


FIGURE 4.6. Obstacle prediction given a constant velocity model, with velocity $v_{o b s}$ and radius growth with uncertainty in position $x_{u}$, velocity $v_{u}$ and acceleration $a_{u}$.

For a more general uncertainty representation, the position, velocity and acceleration uncertainties can be represented by their $3 x 3$ covariance matrices $\Sigma$, which can be combined to grow in magnitude over time (assuming the different sources of uncertainty are uncorrelated):

$$
\begin{equation*}
\Sigma=\Sigma_{0}+\left(t-t_{o b s}\right)^{2} \Sigma_{v}+\frac{1}{4}\left(t-t_{o b s}\right)^{4} \Sigma_{a} \tag{4.70}
\end{equation*}
$$

where $\Sigma, \Sigma_{v}$ and $\Sigma_{a}$ are the covariance matrices for position, velocity and acceleration respectively. The concept of the probability ellipsoid can then be used to grow a constraint to encompass the uncertain region. A probability ellipsoid is defined as:

$$
\begin{equation*}
\boldsymbol{x}^{T} \Sigma^{-1} \boldsymbol{x}=l^{2} \tag{4.71}
\end{equation*}
$$

The value of $l$ can be set to 1,2 , or 3 to set the ellipsoid to be at the $1 \sigma, 2 \sigma$ or $3 \sigma$ bounds respectively. The covariance matrix can be represented as a shape matrix: $\mathbf{A}_{p}=\frac{1}{l^{2}} \Sigma^{-1}$, or inversely, the shape matrix
of the physical ellipsoid obstacle could be represented as a covariance matrix: $\Sigma_{s}=\frac{1}{l^{2}} \mathbf{A}^{-1}$. In covariance matrix form the obstacle physical shape and the position uncertainty can be combined to get a final shape matrix that represents the region to be constrained:

$$
\begin{align*}
\Sigma_{c} & =\Sigma+\Sigma_{s}  \tag{4.72}\\
\mathbf{A} & =\frac{1}{l^{2}} \Sigma_{c}^{-1} \tag{4.73}
\end{align*}
$$

The shape matrix $\mathbf{A}$ is then used in the ellipsoid obstacle cost function, Eq. 4.43.
For highly conservative trajectory planning, the maximum assumed acceleration the obstacle is capable of could be used in place of the acceleration uncertainty. In the case where the obstacle is another vehicle, then this could be linked to knowledge of that vehicle's performance.

A consideration when growing the radius is that the obstacle can take up a significant amount of free-space when planning over a long time interval; therefore the method of growing the radius is designed to operate with regular observations of the obstacles, and subsequent replanning (see Section 4.3.7). The observations provide an update on the position of the obstacle, to collapse the radius down and provide more free-space in which to plan.

### 4.3.6 Optimisation Techniques

To take advantage of the convex nature of the path and keep-in constraints, ASTRO uses gradientdescent optimisation. The subspace-projection (Eq. 4.28) is a key part of the gradient-descent optimisation in ASTRO, where any step in the optimisation coefficients, $\delta \boldsymbol{C}$, is projected onto the subspace that always complies with the boundary conditions.

In each iteration of the optimisation, the gradient is computed for each cost component, and combined in the augmented cost function gradient:

$$
\begin{equation*}
\frac{\partial J}{\partial \overline{\boldsymbol{C}}}=\sum_{i=1}^{d} \frac{\partial\left[f_{s}\right]_{i}}{\partial \boldsymbol{C}_{i}}+\sum_{j=1}^{2} K_{j} \frac{\partial f_{c_{j}}}{\partial \overline{\boldsymbol{C}}} \tag{4.74}
\end{equation*}
$$

The different dimensions can not be optimised independently because constraints, such as ellipsoid obstacles, or spherical acceleration bounds, mix dimensions, hence the optimisation acts on the stacked vector of all coefficients for each dimension, $\overline{\boldsymbol{C}}$. A Quasi-Newton gradient-descent optimisation is used on these coefficients, where a local quadratic approximation is made: represented by an approximate Hessian $\boldsymbol{H}$. The inverse of the Hessian is used along with the gradient to get the optimisation step:

$$
\begin{equation*}
\delta \overline{\boldsymbol{C}}=-\boldsymbol{H}^{-1} \frac{\partial J}{\partial \overline{\boldsymbol{C}}} \tag{4.75}
\end{equation*}
$$

This step, $\delta \overline{\boldsymbol{C}}$, is then split into each dimension, $\delta \boldsymbol{C}_{i}$, for projection onto the subspace that ensures the boundary conditions (Eq. 4.28). The coefficients are updated with the projected step, $\delta \overline{\boldsymbol{C}}_{\perp}$, that is scaled by the step size: $\alpha$ :

$$
\begin{equation*}
\overline{\boldsymbol{C}}_{\text {new }}=\overline{\boldsymbol{C}}_{\text {old }}+\alpha \delta \overline{\boldsymbol{C}}_{\perp} \tag{4.76}
\end{equation*}
$$

The step size $\alpha$ is determined in a one dimensional optimisation called a line-search. A backtracking Armijo line-search is used: starting at 1 , $\alpha$ decreases by a constant factor $\beta$, ( $0 \leq \beta<1$ ), until the Armijo-Goldstein condition [9] is met:

$$
\begin{equation*}
J\left(\overline{\boldsymbol{C}}_{n e w}\right)-J\left(\overline{\boldsymbol{C}}_{o l d}\right)<\sigma_{1} \alpha \overline{\boldsymbol{C}}_{o l d} \frac{\partial J}{\partial \overline{\boldsymbol{C}}} \tag{4.77}
\end{equation*}
$$

The variable $\sigma_{1}$ is a user setting for convergence tolerance, normally chosen at $1 \times 10^{-8}$. A value of 0.85 for $\beta$ is commonly used. The line-search also exits if an iteration limit reached.

The Hessian is approximated with the BFGS method [24, 66, 80, 207], supplemented with the Sherman-Morrison-Woodbury formula to update and track $\boldsymbol{H}^{-1}$ through each optimisation [29]. $\boldsymbol{H}^{-1}$ is initialised as an identity matrix, and then updated at the end of each iteration, $k$, with:

$$
\begin{align*}
\boldsymbol{H}_{k+1}^{-1} & =\boldsymbol{H}_{k}^{-1}+\frac{\left(\mathbf{s}_{k}^{T} \boldsymbol{y}_{k}+\boldsymbol{y}_{k}^{T} \boldsymbol{H}_{k}^{-1} \boldsymbol{y}_{k}\right)\left(\mathbf{s}_{k} \mathbf{s}_{k}^{T}\right)}{\left(\mathbf{s}_{k}^{T} \boldsymbol{y}_{k}\right)^{2}}-\frac{\left(\boldsymbol{H}_{k}^{-1} \boldsymbol{y}_{k} \mathbf{s}_{k}^{T}+\mathbf{s}_{k} \boldsymbol{y}_{k}^{T} \boldsymbol{H}_{k}^{-1}\right)}{\left(\mathbf{s}_{k}^{T} \boldsymbol{y}_{k}\right)}  \tag{4.78}\\
\boldsymbol{y}_{k} & =\operatorname{Proj}\left(\left[\frac{\partial J}{\partial \overline{\boldsymbol{C}}}\right]_{\text {new }}-\left[\frac{\partial J}{\partial \overline{\boldsymbol{C}}}\right]_{\text {old }}\right)  \tag{4.79}\\
\mathbf{s}_{k} & =\alpha \delta \overline{\boldsymbol{C}}_{\perp} \tag{4.80}
\end{align*}
$$

The function Proj is projecting the change in gradient onto the subspace of feasible solutions: Eq. 4.28.

If a positive curvature is detected, with $\mathbf{s}_{k}^{T} \boldsymbol{y}_{k}<=0$, then the inverse Hessian is reset to an identity matrix.

The optimisation is deemed to have converged when one of several criteria have been met: a first order decrease condition, when the predicted step reduction in cost is sufficiently small:

$$
\begin{equation*}
\delta \overline{\boldsymbol{C}}_{\perp} \frac{\partial J}{\partial \overline{\boldsymbol{C}}}<\sigma_{2} \tag{4.81}
\end{equation*}
$$

or a sufficient decrease check, when the change in cost is sufficiently small:

$$
\begin{equation*}
\frac{\left|J_{k+1}-J_{k}\right|}{J_{k}}<\sigma_{3} \tag{4.82}
\end{equation*}
$$

The convergence tolerances $\sigma_{2}$ and $\sigma_{3}$ can be tuned. Values of $1 \times 10^{-10}$ and $1 \times 10^{-12}$, respectively, are generally used in this work.

Using this customised method for optimisation rather than existing optimisation tools enables the use of the subspace-projection method to comply with the boundary conditions on every iteration, and leads to much-improved computation times. An example comparison with MATLAB's fmincon function is presented in Section 4.4.7.

Several techniques are combined with the gradient-descent approach to assist optimisation in different scenarios: quadratic line-search for entirely convex problems; randomised initial seeding and perturbations for non-convex problems; and custom constraint weighting for complex constraint sets.

### 4.3.6.1 Convex, Quadratic Optimisation Steps and Line Search

For scenarios with only convex constraints, such as the trajectory cost, corridors and performance constraints, the convex nature of the resulting augmented cost function can be used to speed-up the optimisation. In addition to being convex, the cost functions described in Eqs. 4.8, 4.43 and 4.58 are quadratic, with constant second derivatives, enabling a computation of the optimal step size and direction using Newton's method:

$$
\begin{equation*}
\delta \overline{\boldsymbol{C}}^{*}=\left[\frac{\partial^{2} J}{\partial \overline{\boldsymbol{C}}^{2}}\right]^{-1}\left[\frac{\partial J}{\partial \overline{\boldsymbol{C}}}\right] \tag{4.83}
\end{equation*}
$$

Without any other factors, a single step will give the optimal solution, but the subspace-projection selects the component of the step that does not violate the boundary conditions; hence the full step is not taken, but instead a component of the step that will be sub-optimal and require multiple iterations to converge. Depending on the scenario, the projected step using the curvature may be worse than with the projected gradient step. The result of using convex, quadratic steps is analysed in Section 4.4.6.

While using Newtons method before projection is not immediately beneficial, the projection of a convex and quadratic search space is still convex and quadratic, allowing Newton's method to be used in the line-search. The one-dimensional optimisation of the step size can be directly solved by numerically approximating the gradient and curvature with respect to the step size at a starting value of 1 . The optimal step size, $\alpha^{*}$, can then be computed with:

$$
\begin{equation*}
\alpha^{*}=1-\left[\frac{\partial^{2} J}{\partial \alpha^{2}}\right]^{-1}\left[\frac{\partial J}{\partial \alpha}\right] \tag{4.84}
\end{equation*}
$$

The use of the quadratic line-search provides more optimal step sizes as well as taking away the need for multiple iterations in the line-search, as analysed in Section 4.4.6.

While there is some limitation in what problems are fully convex, a complex obstacle field a can be represented with free-space by a set of convex, keep-in corridor constraints. This approach is demonstrated in Section 4.4.1.

### 4.3.6.2 Randomised Initial Seeding and Perturbations

If the optimisation is not convex, such as when including obstacles, then the optimisation is susceptible to converging in local minima, especially for complex, obstacle-rich environments. Taking inspiration of sampling-based methods such as $R R T^{*}$, this issue can be addressed by generating a set of random initial seed trajectories ${ }^{2}$.

The seed trajectories are produced by randomised perturbations of the coefficients, $\overline{\boldsymbol{C}}$, from the initial, straight line estimate between boundary conditions, or from an existing trajectory when replanning. The coefficient perturbations are then projected onto the subspace that ensures the boundary conditions, using Eq. 4.29.

Each random seed is optimised for a set number of iterations, and the lowest cost solution is taken to continue the optimisation. The selection of the amount of perturbation, the number of initial seeds,

[^8]and the number of iterations to run, all affect the performance of the method, and provide an ability to tune. For instance, if running on a multi-core processor, it might be desired to run, in parallel, a number of samples equal to the number of cores. For demonstrations presented here, four initial seeds are run, with four iterations until the winning solution is selected. For more details and analysis of this method, see the work of Rigter [195].

An alternative approach is to randomly perturb the trajectory once it has been detected that the solution is converging in an infeasible local minimum. For many scenarios, such as with ESDF representations of real environments, only 3-4 iterations are required to converge; hence if the solution is still infeasible after four iterations, it is likely caught in an infeasible local minimum. At this stage the coefficients are randomly perturbed, then forced to comply with the boundary conditions (Eq. 4.29) before continuing the optimisation. Such an approach aims to jump the solution out of local minima to eventually converge on a feasible solution. Demonstrations with this approach are presented in Section 4.4.2.

### 4.3.6.3 Iterative Sub-problems

For problems with many constraints, the search-space becomes very complex, and the optimisation takes a long time to solve. A relaxed sub-problem can be solved first, with only the trajectory cost, to improve the performance in such scenarios. The solution to this sub-problem provides a dynamically smooth, or low distance trajectory (depending on the trajectory cost function). This initial trajectory is used as a seed for a second optimisation with the addition of constraints. The second optimisation effectively perturbs the trajectory to satisfy the constraints, e.g. adjusting sideways to go around an obstacle. If there are multiple constraints then this sub-problem approach can be performed iteratively, first adding obstacle constraints, then corridors, then performance constraints in multiple optimisations. This approach is similar to that described in [105].

The iterative sub-problem approach gives a high-quality seed trajectory to each successive optimisation as well as giving a simpler problem for the complex non-convex optimisation with obstacles. The result is a quicker overall optimisation. For example, when running the optimisation with obstacle constraints, early iterations of the trajectory would have strong violations of performance constraints, before settling on a smoother final trajectory. Therefore, if additional constraints are present for these iterations, they would give a large gradient step in the opposite direction. The result is that the trajectory bounces between infeasible solutions, and is slow to convergence. Additionally, the solution to the sub-problem with obstacles can often give a solution that already complies with the corridor and acceleration constraints, providing a suitable final solution with a simpler optimisation.

### 4.3.6.4 Customised Weighting

The weighting on the obstacle constraint has a large influence on the optimisation performance, with too large a weighting causing unstable large jumps in the solution and too small a weighting causing a slow convergence. To address this sensitivity, the weighting, $W_{j}$, for a constraint can be computed from the starting trajectory cost $f_{s}$ and starting constraint cost $\left[f_{c}\right]_{j}$ :

$$
\begin{align*}
W_{j} & =10^{\chi}  \tag{4.85}\\
\chi & =\operatorname{round}\left[v \log _{10}\left(f_{s}\right)\right]-\operatorname{round}\left[\mu \log _{10}\left(f_{c}\right)\right] \tag{4.86}
\end{align*}
$$

The parameters $v$ and $\mu$ are tuning coefficients that use the starting costs to scale the order of magnitude of the constraint cost. For example, tests in this paper set $v$ and $\mu$ near to 3.0 and 1.0 respectively, which effectively scales the obstacle cost to be approximately two orders of magnitude higher than the trajectory cost, to be of sufficient influence to force a feasible trajectory in early iterations.

### 4.3.6.5 Obstacle Inflation and Exiting When Feasible

For highly complex obstacle fields, such as an ESDF of a cluttered environment, it can take a long time to produce the optimal solution, but a feasible solution (collision-free) can be generated more quickly. Therefore a new exit criterion is used in the optimisation to exit when the solution is first feasible. While not fully optimal, by using the iterative sub-problem approach, the starting solution will be near optimal, and small adjustments to become feasible will retain a high quality (low trajectory cost function) trajectory. This new exit criterion is combined with an inflation of the obstacles, in addition to the inflation to account for the robot size. The inflation provides a larger and more uniform gradients to push the trajectory into free-space and is ignored when checking for feasibility. For keep-in constraints, this translates to a decrease in their size. Examples successfully demonstrating this approach are presented in Section 4.4.1, and an analysis in Section 4.4.4.

### 4.3.7 Replanning and Multiple Robots

For continuous operations in dynamic environments, replanning a trajectory becomes a key capability. ASTRO can be used for replanning in a similar manner to many other planners, by repeatedly optimising a trajectory from a predicted position shortly in the future to a goal position. The goal position may be static or could be moving in a receding horizon approach based on a long-term path or an exploration goal. The required rate of replanning depends on the speed of the algorithm, the computational resources and how dynamic the environment is.

The approach taken for ASTRO is to replan, if possible, when there is an observed change, such as a new observation of a dynamic obstacle that updates the motion estimate. With this approach, the replanning timeline is as depicted in Fig. 4.7. There is a delay from the time of an observation to having the processed result of the observation. Then there is a fixed time allocated for trajectory optimisation, with the updated trajectory acted on at the end of the fixed computation time. If the optimisation does not converge in time, a sub-optimal solution is returned, if feasible. If dynamic obstacles are considered,
their motion is predicted forward from the time of observation, $t_{o b s}$, and the trajectory is planned from the predicted robot position at the end of the computation time: $t_{\text {replan }}$. The robot position at $t_{\text {replan }}$ is predicted by assuming the robot is following a previously planned trajectory.

## Time Delay



Figure 4.7. Timeline of delays and fixed replanning computation time.

In a very restrictive scenario, there is the possibility that the path generated at the end of the time limit could violate constraints (if there was insufficient time to reach a feasible solution). Alternate strategies could be employed in this case, such as a stop command.

### 4.3.7.1 Seeding a Trajectory for Replanning

To speed-up replanning when the goal location remains constant, the remaining segment of the previously planned trajectory is used as a seed to the next optimisation problem ${ }^{3}$. The remaining trajectory is sampled to give the positions points $\boldsymbol{X}_{i}^{\prime}$, and is fit to solve for the coefficients $\boldsymbol{C}_{i}^{\prime}$ :

$$
\begin{equation*}
\boldsymbol{C}_{i}^{\prime}=\boldsymbol{P}_{i}^{+} \boldsymbol{X}_{i}^{\prime} \tag{4.87}
\end{equation*}
$$

The matrix $\boldsymbol{P}_{i}$ is an $n_{f} \times N_{i}$ matrix of the Legendre polynomial basis functions for position, evaluated at $n_{f}$ samples of normalised time, $t_{0}^{\prime}, \cdots t_{n_{f}-1}^{\prime}$ ranging from -1 to 1 :

$$
\boldsymbol{P}_{i}=\left[\begin{array}{cccc}
\frac{1}{a^{\xi}} P_{1}^{\delta_{\xi}}\left(t_{0}^{\prime}\right) & \frac{1}{a^{\xi}} P_{2}^{\delta_{\xi}}\left(t_{0}^{\prime}\right) & \cdots & \frac{1}{a^{\xi}} P^{\int_{\xi}}\left(t_{0}^{\prime}\right)  \tag{4.88}\\
\frac{1}{a^{\xi}} P_{1}^{\delta_{\xi}}\left(t_{1}^{\prime}\right) & \frac{1}{a^{\xi}} P_{2}^{\delta_{\xi}}\left(t_{1}^{\prime}\right) & \cdots & \frac{1}{a^{\xi}} P_{N_{i}}^{\delta_{\xi}}\left(t_{1}^{\prime}\right) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{1}{a^{\xi}} P_{1}^{\int_{\xi}}\left(t_{n_{f}-1}^{\prime}\right) & \frac{1}{a^{\xi}} \int_{2}^{\int_{\xi}}\left(t_{n_{f}-1}^{\prime}\right) & \cdots & \frac{1}{a^{\xi}} P_{N_{i}}^{\delta_{\xi}}\left(t_{n_{f}-1}^{\prime}\right)
\end{array}\right]
$$

Eq. 4.87 amounts to a least squares fit of the polynomial to the sample points. To perform the fit, time has to be has to be normalised again for the new time interval (Eq. 4.3), with the same final time, $t_{f}$, and an updated initial time, $t_{0}=t_{\text {replan }}$. The boundary conditions, $\boldsymbol{X}_{B C}$ are also updated, and the boundary conditions matrix $\left[\boldsymbol{P}_{B C}\right]_{i}$ rescaled for the new times, by adjusting the $a$ values (see Section 4.3.8.1 for details). After evaluating Eq. 4.87, the fitted coefficients $\boldsymbol{C}^{\prime}$ have the boundary conditions enforced with Eq. 4.29 to then be ready as the seed for the replanning optimisation.

[^9]When replanning, there are options to either keep the same number of sample points in each planning instance or reduce the number of samples as the robot gets closer to the goal. If the number of sample points is fixed, then $\boldsymbol{P}_{i}^{+}$can be pre-computed.

### 4.3.7.2 Multiple Robots

A challenging application of replanning with dynamic obstacles is to coordinate the motion of multiple robots. A decentralised process is used to individually solve the trajectories for each robot using ASTRO, with every other robot considered as a dynamic obstacle. Two levels of coordination are considered: one with detailed communication (cooperative) and one with complete independence (adversarial).

In the cooperative case, it is assumed that the current position and the entire planned trajectory is communicated between each robot. The adversarial case provides greater autonomous capability, obtaining information on the positions and velocities of the other robots as would be possible through onboard sensors, and treating the other robots purely as unknown dynamic obstacles.

### 4.3.8 Multi-Segment Optimisation

For planning trajectories over large spaces, the optimisation can be made more efficient by splitting the trajectory into multiple segments. In an approach frequently used [26, 32, 63, 121, 139, 170] these segments can be divided between waypoints that come from a global, sampling-based planner such as RRT* [102]. Unless all derivatives are fixed at these waypoints, it is not optimal to plan each segment individually, as the velocity and acceleration at the end of one segment will impact what trajectories are possible in the next. Therefore the set of segments are optimised in one batch, stacking the coefficients, $\boldsymbol{C}_{i}$, for each segment, and enforcing continuity between each segment. The stacked coefficients for the $i$ th dimension for $n_{\text {seg }}$ segments is given by:

$$
\underline{\boldsymbol{C}}_{i}=\left[\begin{array}{c}
\boldsymbol{C}_{i 1}  \tag{4.89}\\
\boldsymbol{C}_{i 2} \\
\vdots \\
\boldsymbol{C}_{i, n_{\mathrm{seg}}}
\end{array}\right]
$$

The cost function, Eq. 4.8 remains the same, but the matrix $\boldsymbol{P}_{\text {int }}$ (Eq. 4.11) is repeated $n_{\text {seg }}$ times in a block diagonal matrix:

$$
\left[f_{s}\right]_{m}=\sum_{i=1}^{d} \boldsymbol{C}_{i}^{T}\left[\begin{array}{cccc}
\boldsymbol{P}_{\mathrm{int}} & \mathbf{0} & \cdots & \mathbf{0}  \tag{4.90}\\
\mathbf{0} & \boldsymbol{P}_{\mathrm{int}} & \cdots & \mathbf{0} \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{P}_{\mathrm{int}}
\end{array}\right] \underline{\boldsymbol{C}}_{i}
$$

In addition to the boundary conditions, constraints need to be included to enforce continuity at waypoints. The first segment has no continuity conditions enforced and instead has boundary conditions for both the start and the end. All subsequent segments have continuity constraints on all derivatives at the start of the segment, which enforces both continuity and any fixed boundary conditions. The end
of the segments then have boundary conditions applied. This organisation of constraints means that any derivative that is not specified at a waypoint is left free, but with continuity enforced.

First, the $\boldsymbol{P}_{B C}$ matrices are formed for each segment with the active boundary conditions. The values in each $\boldsymbol{P}_{B C}$ will be different for each segment as the value of $a$ from Eq. 4.5 and 4.24 changes as a function of time. The combined boundary condition matrix and boundary condition vector becomes:

$$
\overline{\boldsymbol{P}}_{B C}=\left[\begin{array}{cccc}
{\left[\boldsymbol{P}_{B C}\right]_{1}} & \mathbf{0} & \cdots & \mathbf{0}  \tag{4.91}\\
\mathbf{0} & {\left[\boldsymbol{P}_{B C}\right]_{2}} & \cdots & \mathbf{0} \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{0} & \mathbf{0} & \cdots & {\left[\boldsymbol{P}_{B C}\right]_{n_{\text {seg }}}}
\end{array}\right] \quad\left[\overline{\boldsymbol{X}}_{B C}\right]_{i}=\left[\begin{array}{c}
{\left[\boldsymbol{X}_{B C}\right]_{i, 1}} \\
{\left[\boldsymbol{X}_{B C}\right]_{i, 2}} \\
\vdots \\
{\left[\boldsymbol{X}_{B C}\right]_{i, n_{\text {seg }}}}
\end{array}\right]
$$

For the continuity constraints, there are effectively a new set of boundary constraints is a similar form to Eq. 4.23, that is:

$$
\begin{equation*}
[\mathbf{0}]=\left[\boldsymbol{P}_{C C}\right]_{i} \underline{\boldsymbol{C}}_{i} \tag{4.92}
\end{equation*}
$$

The matrix $\boldsymbol{P}_{C C}$ subtracts the end state of segment $k$ from the starting state of segment $k+1$, with the constraints that the result is equal to zero ${ }^{4}$. An example of this matrix for three segments (using the definition for $\boldsymbol{P}_{L}$ in Eq. 4.24) is:

$$
\left[\boldsymbol{P}_{C C}\right]_{i}=\left[\begin{array}{ccc}
\boldsymbol{P}_{L}(1) & -\boldsymbol{P}_{L}(-1) & \mathbf{0}  \tag{4.93}\\
\mathbf{0} & \boldsymbol{P}_{L}(1) & -\boldsymbol{P}_{L}(-1)
\end{array}\right]_{i} \underline{\boldsymbol{C}}_{i}
$$

For the continuity constraints, all rows of $\boldsymbol{P}_{L}$ are active for derivatives where it is desired to maintain continuity between segments. If all derivatives up to acceleration are constrained, and there are three segments, then $\left[\boldsymbol{P}_{C C}\right]_{i}$ will be of dimensions $6 \times 3 N_{i}$. The boundary conditions and continuity conditions stack to give the overall, multi-segment boundary conditions for dimension $i$ :

$$
\left[\begin{array}{c}
\overline{\boldsymbol{X}}_{B C}  \tag{4.94}\\
\mathbf{0}
\end{array}\right]_{i}=\left[\begin{array}{c}
\overline{\boldsymbol{P}}_{B C} \\
\boldsymbol{P}_{C C}
\end{array}\right]_{i} \underline{\boldsymbol{C}}_{i}
$$

this expression is grouped together to write as:

$$
\begin{equation*}
\left[\underline{\boldsymbol{X}}_{C}\right]_{i}=\left[\underline{\boldsymbol{P}}_{C}\right]_{i} \underline{\boldsymbol{C}}_{i} \tag{4.95}
\end{equation*}
$$

The grouped boundary conditions vector, $\underline{\boldsymbol{X}}_{B C}$, and basis polynomial matrix, $\underline{\boldsymbol{P}}_{C}$, can then be used in exactly the same way as $\boldsymbol{X}_{B C}$ and $\boldsymbol{P}_{B C}$. First, an initial guess is generated with Eq. 4.26. Then, in the optimisation, the gradient steps are projected onto the subspace that ensures compliance with the boundary conditions by using Eq. 4.28 and Eq. 4.29.

The consideration of degrees of freedom is the same for the multi-segment case, as for the single segment case, noting that the continuity conditions constrain the starting degrees of freedom. The number of continuity constraints depends on the number of derivatives being considered. If constraints

[^10]are placed on all derivatives up to snap for the start and end of a segment, then 10 degrees of freedom are taken up for each dimension. If 9th order polynomials are used, there is a sufficient number of terms to provide a solution $\left(N_{i}=10\right)$, but then there are no degrees of freedom available to optimise. Generally, only the start and end have all derivatives fixed, and all internal waypoints only have position fixed. In this case, each segment has $q+1$ constrained degrees of freedom, where $q$ is the number of derivatives considered.

Constraints are handled separately for each segment, with the resulting costs and gradients added to the overall augmented cost function and augmented gradient:

$$
\begin{equation*}
J_{m}=\left[f_{s}\right]_{m}+\sum_{k=1}^{n_{\mathrm{seg}}} \sum_{j=1}^{n_{o}} K_{j} f_{c_{j}} \tag{4.96}
\end{equation*}
$$

### 4.3.8.1 Time Optimisation

The relative time allocated to each segment is another factor that affects the quality of a trajectory. For example, if segment 1 has a small allocated time, it will have high velocities coming into segment 2, which will limit how smooth that segment can be while still reaching its end waypoint. The relative segment times are therefore parameters to be optimised. An approach similar to that of Bry et al. [26] is taken: an outer-loop, gradient-descent optimisation is performed on the segments times with an augmented cost function of total time:

$$
\begin{equation*}
J_{t}=W_{t} \sum_{k=1}^{n_{\text {seg }}}\left[t_{f}\right]_{k}+J_{m} \tag{4.97}
\end{equation*}
$$

The snap cost, $J_{m}$ is also a function of the segment times, $\left[t_{f}\right]_{k}$; hence the weighting, $W_{t}$ controls the trade-off between quick trajectories (a high $W_{t}$ ) and smooth trajectories, with low $J_{m}$ (low $W_{t}$ ). An optimisation toolbox ${ }^{5}$ is used to perform the optimisation, with the inner-loop snap optimisation being run for each evaluation of $J_{m}$ with a different set of segment times.

When changing segment times, a rescaling can be performed to get an inner-loop initial solution close to the optimal as an initial seed to the optimisation. The time scaling parameter, $a$ from Eq. 4.5 is rescaled by the factor $\Gamma$, based on the change in time for a given segment:

$$
\begin{equation*}
a_{\text {new }}=a_{o l d} \Gamma=a_{o l d} \frac{\left[t_{f}\right]_{n e w}}{\left[t_{f}\right]_{o l d}} \tag{4.98}
\end{equation*}
$$

The rescaling is then used to modify the relevant rows in the existing $\underline{\boldsymbol{P}}_{C}$ matrices by multiplying the values in the row by an appropriate power of $\Gamma$. This can be broken down into multiplying the rows in the $\boldsymbol{P}_{L}$ matrices (Eq. 4.24), from top to bottom, by $\left[\Gamma^{-p}, \ldots, \Gamma^{-1}, 1, \Gamma, \ldots, \Gamma^{q-p}\right]$, for the active boundary conditions in a given segment. For the continuity constraints, the same process follows but noting that different rows of $\boldsymbol{P}_{L}$ are active.

The change in the boundary condition matrix, $\underline{\boldsymbol{P}}_{C}$ represents a change in the boundary conditions; hence these new conditions need to be enforced on the existing set of coefficients, $\underline{\boldsymbol{C}}$. The conditions are enforced through the subspace-projection method, Eq. 4.29.

[^11]After enforcing the boundary and continuity conditions, the coefficients provide an initial guess to run the inner-loop snap optimisation. Because of the considerable computation time when running the inner-loop optimisation many times, the time optimisation is generally performed without any performance or obstacle constraints. The time-optimal solution is then used as a seed for optimisation with the inclusion of constraints, similarly to what is described in Section 4.3.6.3.

The time-rescaling process can also be used to modify all segment times equally, to scale the entire trajectory without changing relative times. This capability provides a rapid method to change the overall trajectory time. The resulting solution will no longer be the optimal solution, though, for the given time penalty, $W_{t}$. Rescaling the whole trajectory could be used as an initial, one-dimensional search in the time optimisation, to get a total trajectory time near to the optimal, before changing time allocation between segments.

### 4.3.9 Summary of ASTRO

This section has presented a thorough overview of the ASTRO algorithm. The key differentiating elements from the state-of-the-art are:

- The flexibility to optimise for any derivative.
- The range of obstacle representations that can be included in the same optimisation, including:
- Performance constraints.
- Free-space bounds.
- Static obstacles and dynamic obstacles.
- The method of considering dynamic obstacles to allow more exploitation of free-space than many existing methods while accounting for uncertainty to ensure safe trajectories.

A set of techniques aid the projected gradient-descent optimisation to solve complex and non-convex problems involving many constraints.

The following section will demonstrate the capabilities of the algorithm in a set of simulated test cases.

### 4.4 Simulation Results

The different features of ASTRO are demonstrated in a set of simulated test cases in this section. Trajectories are being planned for a spherical robot with the freedom to move in any direction.

### 4.4.1 Static Demonstrations

Fig. 4.8 shows a simple example that represents the capability of ASTRO to plan trajectories with static obstacles (from [35]). Boundary conditions are active for the position and velocity, with the velocity constrained to zero at start and end. The cost function used is the integral of velocity squared. There is a fixed time of 100 seconds to complete the trajectory. A mix of ellipsoid and cylindrical obstacle constraints are set up to provide a challenging scenario, where there is an infeasible local minimum in between the central sphere obstacle and the outer torus shape made of the four cylinders.

As shown in Fig. 4.8, the initial guess is a straight line between two points and passes through both spherical constraints as well as the enclosed (but admissible) region between the four cylinders, hence is potentially susceptible to the infeasible local minima. The high weighting on the constraints causes a substantial jump in the early iterations though, allowing the trajectory to settle on the outside in one of the four global minima (given the symmetry of the problem).


Figure 4.8. A single trajectory planned by ASTRO from the black circle to the open circle. The grey shapes are obstacles. Successive iterations of the optimisation are shown from initial guess to the final solution. This result was initially presented in Chamitoff et al. [35].

Fig. 4.9.a presents a simple example with a single cubic rectangular prism constraint, showing how the cost function allows traversal close to the cube face. One possible application of rectangular prisms is to represent voxel-based occupancy, as demonstrated in the example in Fig. 4.9.b, with cubic constraints representing the occupied cells. While effective, the optimisation is slow, and a more efficient representation of the environment such as an ESDF is recommended.


Figure 4.9. ASTRO planning a single segment with cubic prism obstacle constraints. (a) A single cube obstacle with the trajectory going over the top face. (b) Multiple cube constraints from an octomap.

An example using an ESDF as an obstacle representation is shown in Fig. 4.10, where a multisegment trajectory is optimised between five waypoints in a large warehouse, and the cost function is the integral of snap squared. The use of the ESDF enables a large, complex environment to be efficiently represented, with the maximum violation for a given point immediately sampled, rather than searched for through a large set of obstacles. The impact of optimising multiple segments together can be seen in how the trajectory adjusts. The second segment from the left is the segment that needs to change to avoid any collisions, but in doing so, the acceleration into and out of the segment changes, affecting all other segments. While more complicated than planning a single segment, the combined optimisation provides a dynamically superior overall trajectory with these adjustments throughout the segments.

The ESDF can instead be used to define a cylindrical keep-in corridor constraint for each segment, which makes the problem convex. Fig. 4.11 presents an example of using these corridor constraints, with ten waypoints through the same large warehouse environment. An optimisation without constraints is first performed to be the seed for optimisation with constraints. The approximate-path-integral cost method is used when the constraints are added, in addition to constraint inflation and a flag to exit when feasible. The trajectory is successfully able to be pushed to remain inside the very conservative free-space bounds.


FIGURE 4.10. Multi-segment trajectory optimisation with ASTRO using an ESDF representation of the obstacles in a large warehouse environment. The trajectory is planned between waypoints represented by the small quadrotor figures, starting at the green quadrotor and ending at the red quadrotor. A slice of the ESDF is shown, where the colours represent the distance to obstacles: red indicates a collision, shades of purple are close to obstacles, and blue is clear in free-space. The grey mesh is the physical environment. (a) The initial planned trajectory without considering the ESDF: there is a collision near the middle of the trajectory. (b) The final trajectory, successfully avoiding all obstacles.


FIGURE 4.11. Example of trajectory optimisation with keep-in corridor constraints on each segment (red opaque cylinders). The grey mesh represents the physical environment. The seed trajectory in black is planned without any constraints and the green trajectory is the result from optimisation with the corridor constraints. The inset shows the seed path violating the cylindrical bounds and the solution correcting to stay within the bounds.

### 4.4.2 Randomised Seeding and Perturbations

The primary motivation of the randomised initial seeding is to reduce the chances of a trajectory getting caught in a local minimum, something that is especially important for applications with numerous obstacles, providing a large number of local minima. An example is presented here from Rigter [195] to demonstrate the concept. The effect of varying the strength of the randomised initial perturbation is shown in Fig. 4.12.a. With regular randomised seeding when replanning, the planned trajectory can escape from a local minimum to achieve a more optimal solution, as shown in Fig. 4.12.b, in a complex obstacle environment consisting of many ellipsoid obstacles. There is a balance though between performing randomised seeding and computation time, with more iterations of the optimisation required to process each random seed, before proceeding with the best option.


Figure 4.12. Demonstration of randomised initial seeding. (a) Perturbations of a seed path with different perturbation strengths. (b) Trajectory planning with a set of randomised ellipsoids. Strategy 1 is a single trajectory plan. Strategy 2 uses randomised initial seeding with replanning and achieves a more optimal local minima. Images are from [195].

An example of ASTRO with randomised perturbations is shown in Fig. 4.13 in an ESDF of an indoor environment with three waypoints. The solution is successfully able to jump out of a local minimum to converge on a feasible solution.


FIGURE 4.13. Effect of random perturbations to escape from infeasible local minima with an ESDF constraint. Grey objects represent the physical environment, and small icons represent the waypoints. (a) The solution converges in an infeasible local minima with the trajectory in a collision, as outlined with the red ellipse in the inset. (b) With a randomised perturbation, the solution can escape the local minima and converge on a feasible solution.

### 4.4.3 Dynamic Obstacles and Multi-Robot Planning

Attributions: The results in this section, other than the dynamic cylinder results, were presented in Morrell et al. [155], and are the work of this author.

Moving cylinders can be used to represent a variety of possible obstacles. One obstacle of particular interest is moving solar panels on satellites or the International Space Station. Fig. 4.14 presents an example using a rotating cylinder, where the trajectory successfully deviates to pass behind the moving obstacle. The algorithm was run with a single segment and a single planning instance. The integral of velocity squared is the trajectory cost function, which is also used for all dynamic obstacle test cases presented here.

Presented in Fig. 4.15 is a challenging test case where a dynamic obstacle is incorporated in addition to keep-in corridor constraints. The obstacle is adversarial and adjusts its trajectory to block the path of the robot. Therefore, replanning is performed multiple times to adapt to changes in the velocity of the obstacle. The constraints are added to the problem sequentially: first, only the dynamic obstacle is considered, then the corridor constraint is added. The trajectory consistently updates to move around the obstacle and give a safe trajectory to the destination.


FIGURE 4.14. Snapshots of a robot (red dot) moving along a planned trajectory from black circle to blue circle to avoid the moving grey cylindrical obstacle constraint.


FIGURE 4.15. Adversarial dynamic obstacle with keep-in corridor constraints. The dynamic obstacle (the light grey sphere) repeatedly changes direction to obstruct the path from the robot (black sphere) to the goal location. In the top three images, the solid line is the trajectory taken, and the dot-dash lines are interim planned trajectories before the trajectory is replanned. The black circles represent the location where the trajectories are replanned. The bottom three images show three stages of trajectory updates, where the dashed black line is the current, planned trajectory.

### 4.4.3.1 Multiple Robots

Problems with two spherical objects can be extended to the case where both objects are robots running ASTRO. The example presented in Fig. 4.16, has two independent robots, each running ASTRO, which are required to exchange positions around a corner of a junction of corridors in a fixed time of 100 s . In this problem, replanning is performed with simulated delays (as depicted in Fig. 4.7) and performance constraints on acceleration are added. The iterative sub-problem approach is used, starting with all dynamic obstacles, then adding the corridor constraints, and then the performance constraints. The robots replan with simulated time delays: 2 s for image processing and 4 s for trajectory planning. Each robot models the other as an unknown dynamic object, assuming a constant velocity model from the time of observation. The uncertainty in the obstacle position accounted for with obstacle growth, using an assumed maximum acceleration (Eq. 4.69).

Observing Fig. 4.16, there is a significant margin between the two robots due to the modelled uncertainty that grows the radius of the obstacle that ASTRO considers. The modelling of the dynamics of the other robot and the growth in uncertainty allows the trajectories to be completed despite the simulated planning delay.


Figure 4.16. Two robot dynamic replanning example. Two spherical robots using ASTRO to exchange positions around the corner of a corridor junction. Dashed lines are interim planned paths, and solid lines are the trajectories followed. (a) Initial paths are planned with no knowledge of the other spacecraft, leading to overlapping trajectories. (b) After 25 seconds it is modelled that the two spacecraft observe each other. The spacecraft represented by the dark sphere observes first, and diverts around the path of the other, with a large margin to account for uncertainty. (c) 5 seconds later the second spacecraft replans, observing the new dynamics of the first, and hence continues on the same path.

A consideration in coordinating the paths of multiple robots is the replanning dynamics. If multiple robots are replanning their paths at the same time, it is possible that they could update their paths to travel through the same location and hence collide, having no other information on the new path of the other robots. Such dynamics can lead to limit cycle behaviour, such as is observed in Fig. 4.17.a where the two robots repeatedly replan paths that obstruct each other, before generating what is
an infeasible trajectory to avoid a collision. In the cooperative case, priority assignment that sets an order of replanning can overcome the impasse issue (Fig. 4.17.b). The priority assignment can be arbitrarily chosen or could be linked to other coordination logic, such as the importance of robot goals. For the adversarial case, this issue cannot be avoided entirely but could be addressed with a dampening component in the replanning dynamics. For example, one robot could wait and observe instead of acting on a new path, so the replanning times of the two robots move out of phase. The sequencing of one robot to replan before the other enables convenient coordination of trajectories in the example depicted in Fig. 4.16.


FIgure 4.17. Cooperative planning impasse example. Two spherical robots are using ASTRO to plan trajectories to exchange position. Trajectories are replanned at the red circles in regular intervals of time. The plotted lines are the trajectories taken, combined over each replan. (a) When replanning is done at the same time the trajectories come to an impasse, requiring a dynamically infeasible trajectory. The final part of the trajectory is black. (b) When replanning is offset in time, the robots pass efficiently. Final paths in light blue and blue.

More robots can be introduced into the problem to present yet more of a challenge with dynamic obstacles. Fig. 4.18 demonstrates ASTRO successfully working in such a scenario, where six robots are required to exchange positions in pairs along each coordinate axis in a fixed time of 100 s. Performance constraints, constant velocity models and radius growth are the same as for the two robot case above. Simulated observations are made: each robot directly observes the position and velocity of all other robots, before replanning. This observe-and-replan process is done three times per robot, starting at 25 s ( $t_{f}=100 \mathrm{~s}$ ). Time delays of 2 s for image processing and 5 s for trajectory planning are implemented. Replanning is manually offset, to avoid the impasse scenario, using a gap of 5 s between each replan and 1 s between each robot. With similar dynamics to the two robot case, the six robots can successfully navigate through the junction, changing paths in all dimensions to avoid collisions (Fig. 4.18).


FIGURE 4.18. Six robot trajectory planning example, without cooperation. Robots are the spheres, each running ASTRO multiple times, with other robots considered as dynamic obstacles with radius growth to account for uncertainty. Pairs are required to exchange positions along one axis of the corridors in a fixed time of 100 s . Grey corridors are keep-in constraints, and the spheres represent the robots. Plotted lines represent only the trajectories that each of the robots follows, through three planning instances. The bold arrows indicate the direction of travel.

An alternative to using a constant velocity model for dynamic obstacles, and growing the size of the constraints, is to have a more accurate prediction for the path of the dynamic obstacle. If the multiple robots cooperate, then an entire planned trajectory for each robot could be shared. This approach allows for navigation under much tighter constraints, as shown in Fig. 4.19. Trajectories are still planned in a decentralised fashion, but with complete knowledge of the planned paths of the other robots.

Replanning is still required as a robot needs to have planned a trajectory to be able to communicate it to another robot. Therefore, if each robot initially plans without any communication, then plans need to be updated when they do receive information from the others.

Each of the robots successfully navigates through the crowded junction, with some of the spacecraft passing through earlier (the dark sphere with the triangle in the centre), while some wait and pass






Figure 4.19. Animation sequence for six robots navigating through a junction with cooperation in sharing planned trajectories. Robots are the spheres, each running ASTRO multiple times, with other robots considered as dynamic obstacles with radius growth to account for uncertainty. Pairs are required to exchange positions along a corridor in a fixed time of 100 s . Grey corridors are keep-in constraints, and the spheres represent the robots. Plotted lines represent only the trajectories that are followed through three planning instances per robot.

### 4.4.4 Analysis of Optimisation Techniques

A batch of 100 trajectory planning instances is run inside a map of a real warehouse environment to assess the impact of the different optimisation features employed. An ESDF of the environment represents the physical obstructions and is used either as an obstacle constraint or to set the radii of free-space corridors. Trajectories are planned between sets of waypoints which are determined by randomly selecting a start and end goal, then using RRT* [102] to plan a feasible path between them. The path from RRT* provides the set of waypoints with collision-free straight-line paths between them for setting free-space corridors. The number of waypoints can be reduced to provide a less constrained problem when planning with ESDF obstacles. The cost function used is the integral of snap squared ${ }^{6}$. Planning instances are deemed a failure if the resulting trajectory has a collision, a time limit of 50 s is exceeded, continuity constraints are violated or if snap exceeds 10.0 (indicating a degenerate solution). Example trajectories in the environment are showing in Fig. 4.10, and Fig. 4.20.

ASTRO is run on all cases using the ESDF as an obstacle, then using free-space corridors. Each batch of optimisations for each obstacle representation is then repeated with variations in a range of optimisation features. The baseline set of features used in ASTRO is:

- Iterative sub-problem solutions
- Criteria to exit when feasible
- Inflation of constraints
- Approximate-path-integral cost function
- Custom weighting (when using the ESDF obstacle)
- Quadratic line-search (when using the free-space corridors)
- Randomised perturbations after three iterations if not feasible (when using ESDF obstacles)

Tests are run with individual features removed, to isolate the impact of each feature: no iterative sub-problems, exiting only when converged, no inflation of constraints, using the maximum cost function method, fixed weighting (when using the ESDF as an obstacle), and a standard line-search (corridors case only). The baseline results also capture how frequently the perturbations were required. The success rate (feasibility) and computation time are presented in Table 4.1. All computations are run with a Python 2.7 implementation of ASTRO running on an Intel Core i7-7500U, 2.70 GHz processor.

The failed cases for the baseline with ESDF obstacles are due to the time limit being exceeded: the optimisation gets stuck in local minima, and successive randomised perturbations are unable to free the solution within the allocated time. The failed cases with free-space corridors are due to slight errors in the ESDF. The generation of the ESDF is approximate [171], allowing for small errors where a voxel represents an obstacle, but surrounding cells do not have their signed distances adjusted to indicate proximity to a surface collision. The result is that some trajectories can converge inside the free-space corridors, but still be in a collision. Using free-space corridors is susceptible to these errors, whereas ESDF obstacles are not. If the ESDF is used for checking feasibility with corridor constraints, the ESDF errors can be detected and, in most cases, avoided (see analysis in Section 4.4.5).

[^12]TABLE 4.1. Comparison of optimisation configurations: feasibility and computation time

| Constr. Case | Optim. Variant | \% Feas. | $t_{\text {comp }}{ }^{*}$ |
| :--- | :--- | ---: | ---: |
| ESDF | Baseline | 91 | 3.11 |
|  | Exit when converged | 27 | 13.19 |
|  | Max cost method | 75 | 8.04 |
|  | No inflation | 78 | 5.81 |
|  | Static weight 1e-4 | 70 | 5.82 |
|  | Static weight 1e-7 | 80 | 4.00 |
|  | No sub-problems | 50 | 1.43 |
| Corridors | Baseline | 88 | 6.46 |
|  | Exit when converged | 88 | 25.59 |
|  | Max cost method | 88 | 2.97 |
|  | No inflation | 87 | 5.62 |
|  | No sub-problems | 57 | 19.87 |
|  | Backtracking line-search | 82 | 10.17 |

${ }^{*} t_{\text {comp }}$ is the average computation time for all feasible cases

Each optimisation feature is discussed below, highlighting what the results in Table 4.1 reveal about the importance of each feature.

### 4.4.4.1 Iterative Sub-Problems

In these scenarios, the iterative sub-problem approach involves first generating a solution with only the trajectory cost, before solving the full problem. Using this approach can be seen to have a substantial impact on the frequency of success of ASTRO for both ESDF obstacles and free-space corridors: there is a substantial drop in feasibility percentage from the baseline when no sub-problems are used (Table. 4.1).

The failures in the ESDF obstacle case are due to the solution becoming stuck in an infeasible local minimum that pushes the solution towards a degenerate case, where the velocity is so high that sample points on the trajectory jump to either side of an obstacle, from free-space into unknown space. Unknown space is modelled as free to accommodate gaps in the generated ESDF that can occur in the middle of free-space. This degenerate case is reached in only a few iterations; hence the computation time is low. Without first optimising with only trajectory cost, the seed to the optimisation with ESDF obstacles violates many constraints, making it more likely for the degenerate case to occur, or for the seed to be in the homotopy of an infeasible local minimum. Several cases time-out, with the randomised perturbations not able to release the trajectory from such an infeasible local minimum.

With the keep-in corridors, there are similar issues where a degenerate solution is produced, but in this case, the cause is a large gradient step from large violations of the seed trajectory. The degenerate solution is an artificial local minimum that is possible because of the discrete trajectory sampling. The non-optimised seed for planning with the corridor constraints has large violations all along a trajectory. For the cases that succeed, the large violations can be strongly violating boundary conditions, meaning the projected gradient will be small. These small steps avoid the degenerate solution but lead to long optimisation times.

In summary, the results show the substantial benefit in utilising iterative sub-problems.

### 4.4.4.2 Exiting When Feasible and Randomised Perturbations

The difficulty of optimising in a highly non-convex search space with many local minima becomes apparent when removing the criteria to exit when the first feasible solution is obtained. Removing this criterion also requires removing the randomised perturbations, which use the feasibility criteria to flag a required perturbation. The result of removing these features is that the optimisation is very susceptible to both local minima and oscillations between obstacles in the non-convex search space. With ESDF obstacles and the option to exit only when converged, 46 out of 100 converge in an infeasible local minimum, with the remaining failures exceeding the allocated time. Randomised perturbations are required in $13 \%$ of the baseline cases; hence the majority of the failures are because of changing the exit criteria.

By contrast, in a convex search space, with free-space corridors, optimising with a flag to exit when converged will still generate a feasible solution. The differences in computation time between exiting on convergence and the baseline show that ASTRO quickly generates a feasible solution (baseline), after which the trajectory can be further optimised (exit on convergence).

For non-convex problems, it is shown to be very important to exit on the first feasible trajectory. This approach loses optimality, but by seeding the process with an optimal trajectory from the reduced sub-problem, the resulting trajectory can be near-optimal.

### 4.4.4.3 Constraint Inflation

The results in Table 4.1 show inflation to be an important factor when using obstacle constraints, allowing reduced computation time and more successful trajectories. Most of the trajectories that fail without inflated obstacles run out of the allocated computation time, highlighting how the inflation of obstacles is important to give a sufficiently large gradient across the whole trajectory to push it from obstacle boundaries. Without inflation, there is a larger difference between the gradient at obstacle boundaries compared to gradients when in a collision. The difference in gradient is because the boundaries are closer to the minimum of a quadratic function. Inflation effectively shifts all points up a quadratic curve, resulting in a lower relative difference in gradient. The outcome is that, without inflation, optimisation steps are slower to bring the whole trajectory sufficiently into free-space for a feasible solution.

For convex problems, inflation is less critical, as lesser gradients near boundaries are permissible in a convex problem, and can sometimes result in a better-projected gradient, leading to faster computation times.

### 4.4.4.4 Cost Function Methods

For trajectory planning in a complex obstacle-rich environment, the approximate-path-integral cost method gives superior performance to the maximum violation cost method. All of the failures with the maximum cost method are due to the algorithm exceeding the allocated computation time. This result suggests that adjusting a trajectory with a gradient from a single point per segment (the maximum violating point), in a problem with a very cluttered ESDF. The slow optimisation can be due to a number of reasons, depending on the scenario. A small step can keep the maximum violating point in the same

ESDF cell, leading to yet another small step. Alternatively, the gradient at the maximum violating point could force an update that strongly violates boundary conditions, resulting in a small projected step. Another scenario is that the update from the single point could be in a direction that does not help the overall trajectory move away from a collision (e.g. an update along the trajectory, rather than transverse to it).

For cases where the obstacles are convex shapes, such as an ellipsoid, or if there is a single large violation in an ESDF, the maximum cost function can be superior, as the step from the most violating point will not be small and will be in the right direction to move the trajectory into free-space.

For convex problems, the maximum cost method is the better approach, as it performs equally to the approximate-path-integral approach and is quicker to evaluate the cost and gradient.

### 4.4.4.5 Custom Constraint Weighting

The test cases with a static weighting highlight the sensitivity of the optimisation to the weighting for non-convex cases. Hand tuning to a value of $1 \mathrm{e}-7$ gives good results, but with 11 more failures than the baseline, all of which exceed the time limit. Too low a weighting gives very small update steps, whereas too large a weighting leads to long line-searches, as is the result with a weighting of $1 \mathrm{e}-4$, where failures are also due to the time limit being exceeded. The custom weighting allows adaptability across different scenarios to require less tuning of the algorithm.

### 4.4.4.6 Quadratic Line Search

The quadratic line-search provides quicker solution times and can be the difference between finding a solution or not within the allocated time. The differences between a quadratic line-search and a standard backtracking line-search are analysed in more detail in Section 4.4.6.

### 4.4.4.7 Summary of Optimisation Analysis

The analysis of the optimisation techniques demonstrates the most important features for efficient optimisation in complex environments. The use of iterative sub-problems is critical to successful performance for both convex and non-convex cases. Correctly handling constraints is also critical in nonconvex scenarios, where the approximate-path-integral is shown to be valuable, as well as constraint inflation and custom weighting. In cases with numerous local minima, ASTRO is best used to find a feasible solution, with randomised perturbations applied when the solution becomes stuck in an infeasible local minimum. Convex scenarios are less sensitive to settings for producing a feasible result, but using the maximum cost function, quadratic line-search, inflating the constraints and exiting when feasible give the quickest results.

### 4.4.5 Constraint Type Comparison

Three different constraint types could be used to represent the obstacles in an environment: ESDF obstacles, corridor constraints, and corridor constraints with ESDF feasibility checks. The performance of each of these methods is compared here qualitatively and quantitatively using the same batch of
test cases as in the previous section. The tests used the baseline configuration for ASTRO with ESDF obstacles and the maximum violation cost method for the two corridor constraint types.

Fig. 4.20 shows a specific test case where using corridor constraints provides the most conservative solution. If the ESDF is used to check for feasibility rather than the cylinders when using a corridor constraint, the result is identical, because the first iteration gives a trajectory that is feasible for both the corridors and the ESDF. Using the ESDF as an obstacle representation allows more freedom to move the path, and because there is not a requirement of an obstacle-free straight-line path between the waypoints, fewer waypoints can be used.


Figure 4.20. Comparison of different methods of trajectory optimisation through obstacles. The grey mesh represents the physical environment. Red opaque cylinders are free-space bounds on each segment. The seed path is optimised with no constraints. The CYL trajectory uses corridor constraints, and exits when those constraints are satisfied. The ESDF trajectory is using an ESDF as an obstacle constraint.

The resulting trajectory, when using the ESDF as an obstacle, is less conservative though, with less clearance to obstacles (Table 4.2). The computation times in Table 4.2 show that there is a trade-off between providing more freedom to explore the available free-space and computational efficiency.

TABLE 4.2. Trajectory planning results for a single trajectory in a large warehouse environment.

| Constraint Type | $t_{c o m p}(\mathrm{~s})$ | $d_{\min }(\mathrm{m})$ |
| :--- | :---: | :---: |
| ESDF Obstacle | 6.42 | 0.091 |
| Corridors | 3.66 | 0.124 |
| Corridors ESDF check | 10.20 | 0.124 |

Extending the comparison to the full batch of 100 test cases (Table 4.3), the same trends are observed. The entirely convex method of using corridor constraints is quicker, but more conservative, being restricted to be close to the path between waypoints. The use of corridor constraints and ESDF checks takes a long time because of the additional tasks of checking the ESDF for collisions. The

TABLE 4.3. Comparison of constraint types for batch of 100 test cases in a large warehouse environment

| Constraint Type | \% Feas. | $t_{c o m p}(\mathrm{~s})^{*}$ | $d_{\min }(\mathrm{m})^{*}$ |
| :--- | ---: | ---: | ---: |
| ESDF Obstacle | 91 | 3.11 | 0.10 |
| Corridors | 88 | 2.97 | 0.14 |
| Corridors ESDF Check | 97 | 5.11 | 0.12 |

* Characteristics are averages for the feasible trajectories.
time penalty buys the capability to use more free-space, though, giving less conservative trajectories and enabling feasible solutions more frequently. In particular, using ESDF checks produces feasible solutions in cases where errors in the ESDF result in an infeasible voxel inside a free-space corridor. An additional benefit for the corridor constraint types is that if more time is available, the trajectories can be further optimised (Section 4.4.4.2).

The non-convex optimisation with ESDF obstacles gives the most freedom to explore free-space, and allows fewer waypoints, reducing the number of optimisation coefficients, but it is slower due to the difficulty of the non-convex optimisation and can fail to find solutions within a set computation time.

The best constraint type depends on the goals of the application, be it having the safest trajectories, maximising the percentage of success or maximum exploitation of free-space. Other factors to consider are the available computation time and the complexity of the environment. The ESDF obstacle provides greatest use of free space, but can have slower computation times, and can fail to find a solution. The more complex the environment, the slower the algorithm will be. Using corridor constraints can give quicker, more conservative results that can be further optimised, but is susceptible to ESDF errors, and is overly restrictive in simple environments with ample free-space. Adding ESDF checks to the corridor constraints provides the highest percentage of success and less conservative trajectories, but comes with increased computation time.

### 4.4.6 Convex, Quadratic Steps and Line Search

Fig. 4.21 demonstrates what occurs when using the convex quadratic step for a scenario with 6 segments and only the trajectory cost active (the resulting trajectory is shown in Fig. 4.24.a). When using convex quadratic steps, the result correctly gives the optimal step (Fig. 4.21.a), but the component of that step on the subspace of feasible solutions is not optimal in size or direction (Fig. 4.21.b). The minimum cost of this projected step over the range of step sizes represents the step size that the backtracking line-search will aim to find. In this case a step size of 1.0. The projected cost function is still convex though, meaning a step size greater than 1.0 can give the optimal step size. The quadratic line-search gives the optimal step size: 1.35 in this case (Fig. 4.21.c). The step direction is still not optimal in the projected subspace, so 27 iterations are needed to converge. Nonetheless, by implementing the quadratic line-search instead of the standard line-search, the number of iterations is reduced from 79 to 27.

The challenge with using convex quadratic steps is that the projected component of the step will likely not be optimal. For example, if gradient steps are used, the pre-projection step is shown in Fig. 4.21.d. This step is not as optimal as the convex quadratic step, with less cost reduction at the minimum of the parabola, and too large a step with a step-size of 1 . Nonetheless, the projection of this


Figure 4.21. Cost step results at different stages of optimisation in the first iteration, for a scenario with only trajectory cost and six segments. This scenario is shown in Fig 4.24.a. The far right of each plot is the full step size that is intended to optimise the cost; ideally reaching the minimum on a parabola. Top: Using convex quadratic steps. Bottom: using gradient steps. Left: before subspace-projection. Middle: after subspace-projection. Right: after quadratic line-search
step onto the feasible subspace happens to be superior, giving a greater reduction in cost, as depicted in Fig. 4.21.e. This improved performance is not necessarily the case for every iteration though, as shown for later iterations in Fig. 4.22. Whichever step method is used, the quadratic line-search can reduce the number of iterations required, giving a better step in Fig. 4.21.f and reducing the number of iterations from 19 to 12. Fig. 4.22 shows examples in another iteration, where the projected step is far from the optimal step size for both the convex quadratic step and the gradient step, demonstrating that the quadratic line-search can give a substantially higher reduction in cost.

Because the convex quadratic step is to zero cost (Fig. 4.21.a, Fig. 4.22.a), which represents a zero length trajectory, the step tends to have a very large component that violates the boundary constraints. Therefore the projection of that step onto the subspace that adheres to the boundary conditions is small and sub-optimal. Tests presented here use the gradient step method rather than the convex quadratic step, as it has shown to give a better-projected step more consistently.


FIGURE 4.22. Cost step results at different stages of optimisation in the third iteration in a scenarion with only trajectory cost and six segments, Fig 4.24.a. The far right of each plot is the full step size that is intended to optimise the cost; ideally reaching the minimum on a parabola. Top: Using convex quadratic steps. Bottom: using gradient steps. Left: before subspace-projection. Middle: after subspace-projection. Right: after quadratic line-search

Examples of the quadratic line-search with corridor constraints is shown in Fig. 4.23, for the trajectory in Fig. 4.24. The initial projected step coming out of the gradient-descent is large, increasing the cost for all but a very small step size (Fig. 4.23.a). The standard line-search requires many iterations to get to a step size that is suitably small, whereas the quadratic line-search computes the optimal step directly without searching (Fig. 4.23.b). The step is not exactly at the optimal in this case due to inaccuracies in the numerical computation of the gradient and curvature. In this scenario, the step size from the quadratic line-search is 0.036 , from 5 cost evaluations (to compute the gradient and curvature). In contrast, the standard line-search computes a step size of 0.0536 with 18 cost function evaluations. For initial iterations, the quadratic line-search is quicker and gives a superior step size.

In the second iteration, the step post-projection is smaller (Fig. 4.23.c); hence the standard linesearch takes fewer iterations, converging in three iterations to a step size of 0.615 . This step size is still double the optimal that is computed by the quadratic line-search: 0.325. Fig. 4.24.b and Fig. 4.24.c, highlight the difference in the trajectories from the different line-search methods by showing how the quadratic line-search gives a step to get the trajectory closer to the middle of free-space. For outer-loop iterations when the solution is closer to the optimal, the quadratic line-search provides a more optimal step, though it may require more evaluations of the cost function.

The results presented here show the substantial benefit of using the quadratic line-search: enabling the use of a step size above 1, computing the optimal step size, and determining the step size in fewer iterations than the standard line-search. The result is a quicker optimisation, as demonstrated in Table 4.1. For convex problems, the quadratic lines search is the superior method to use.


Figure 4.23. Cost step results showing benefit of quadratic line-search. Results are from optimisation with trajectory cost and keep-in corridor constraints with six segments, Fig 4.24. Far right of each plot is the full step size that is intended to optimise the cost; ideally reaching the minimum on a parabola. Top: cost steps after subspace-projection that is used in a standard line-search. Bottom: cost step from quadratic line-search. Left: first iteration. Right: second iteration.


FIGURE 4.24. Convex trajectory optimisation between seven waypoints (quadrotor models), with cylindrical keep-in constraints (red opaque cylinders). Grey mesh represents the physical environment. (a) Initially optimised trajectory with no constraints considered. (b) Result after optimising with keep-in constraints using the standard line-search. (c) Result after optimising with keep-in constraints using the quadratic line-search.

### 4.4.7 Computation Time Analysis

Attributions: Results in this section were presented in Morrell et al. [155] and are the work of this author.

Table 4.4 presents comparative computation times for six test cases. Also highlighted is the improvement in computation that comes by moving from a fmincon SQP implementation to utilise the full subspace descent projected gradient implementation. The test cases listed are variations to the those presented above that include static obstacles, dynamic obstacles and performance constraints. All planning instances are with a single segment. The timing comparisons were run with 64-bit MATLAB R2014a on an Intel Core i7-3520M, 2.9 GHz processor.

The simulation examples with two robots and six robots included modelling of time delays and restrictions in computation times to simulate how the algorithm might be used for online planning. Table 4.5 shows the minimum, maximum and mean computation times for those cases. The ASTRO computations completed within the allocated time for a majority of the path plans, with only $7.4 \%$ exceeding the limit, and all by less than 1 second. With the limited computation time and the associated time delay modelled in the simulations, ASTRO was successfully able to plan safe trajectories through the environment. These results suggest that there is potential for ASTRO to be used in real-time on-line

Table 4.4. Trajectory planning computation times and improvement.

|  | ASTRO | Average times to plan trajectory (seconds) |  | $\times$ Speed |
| :--- | :---: | :---: | :---: | :---: |
| Test Case | Runs | fmincon SQP | Subspace Gradient | Increase |
| 1. Static | 1 | 3.84 | 0.40 | 9.59 |
| 2. Dynamic Obstacle | 10 | 41.04 | 1.58 | 26.00 |
| 3. 2 Robot | 6 | 43.36 | 0.41 | 106.7 |
| 4. 6 Robot | 24 | 35.65 | 1.45 | 24.63 |
| 5. 2 Robot - R | 24 | 43.36 | 0.39 | 111.2 |
| 6. 6 Robot -R | 24 | 35.65 | 1.22 | 29.22 |

R - restricted computation time used in gradient method
planning operations.
TABLE 4.5. Trajectory planning computation times

| Test <br> Case | No. path <br> plans | Min <br> time $(\mathbf{s})$ | Max <br> time $(\mathbf{s})$ | Ave. <br> time $(\mathbf{s})$ |
| :---: | :---: | :---: | :---: | :---: |
| Adversarial | 9 | 0.58 | 5.33 | 1.58 |
| 2 robot | 6 | 0.03 | 1.58 | 0.39 |
| 6 independent | 24 | 0.55 | 5.85 | 2.48 |
| 6 cooperative | 24 | 0.03 | 5.97 | 1.22 |

### 4.4.8 Summary of Simulation Tests

The set of simulated test cases have demonstrated the capability of ASTRO. The algorithm can successfully optimise trajectories with a range of constraints: static obstacles, dynamic obstacles, free-space restrictions and performance constraints. Predicting the motion and uncertainty of dynamic obstacles enables successful planning in highly dynamic and confined scenarios. Continuous trajectories can be planned between multiple waypoints to traverse through complex environments. The subspaceprojection gradient-descent, aided with a set of optimisation features, including iterative sub-problems, quadratic line-searches, and randomised perturbations, enables quick generation of feasible solutions in complex, obstacle-rich environments. These solutions can be generated for both convex and non-convex constraint types. The computation time of the algorithm shows potential for online trajectory planning, with successful results obtained when simulating computational delays.

One factor that has not been explored in detail here is the influence of different numbers of coefficients, and the trade-off between complex trajectories and simpler optimisations. Preliminary testing showed a minimal impact of varying the number of coefficients. However, this factor should be investigated at greater depth in future work.

The next section describes an implementation of ASTRO for online trajectory planning with spacebased robots. Chapter 5 then presents how ASTRO can be applied to quadrotors before hardware demonstrations are presented in Chapter 6 for tracking trajectories that are planned off-line for complex environments. These hardware tests include comparisons against state-of-the-art trajectory optimisation algorithms.

### 4.5 Trajectory Optimisation for Space-Based Robotics


#### Abstract

Attributions: Details about the on-orbit tests were presented in Chamitoff et al. [35]. The contributions from this thesis that have been presented in [35] include: analysis of dynamic obstacle test cases, analysis of lessons learned from the tests, and a discussion on how the results informed future developments of the algorithm.


Trajectory planning in three dimensions through a range of obstacles is a valuable capability for free-flying spacecraft in applications such as satellite servicing, satellite inspection, monitoring of the outside of a space station, or a range of operations inside a space station: checking inventory, filming, or checking air quality.

There is a test-bed on the International Space Station (ISS) designed to test the algorithms required for such applications: the Synchronized Position Hold, Engage, Reorient Experimental Satellites (SPHERES) [202]. These free-floating robots operate inside the ISS as a resource for researchers around the world to test autonomous navigation algorithms in micro-gravity.

An earlier version of ASTRO was tested on the SPHERES, prior to this thesis, to demonstrate the capability of the algorithm for online planning of three-dimensional trajectories in the presence of static and dynamic obstacles. The contributions in this work are the analysis of the results of the testing, and the application of the lessons learned to improve ASTRO.

First, an overview of SPHERES is presented, followed by a summary of the on-orbit testing. Analysis of the results will be presented, particularly highlighting the analysis of the dynamic obstacle results, which was performed as part of this work. Finally, there is a discussion on how the lessons learned have been applied to algorithm development.

From this analysis, the fundamental contributions are: describing the successful use of ASTRO, on-orbit, for dynamic obstacles, highlighting improvements that can be made in how dynamic obstacles are considered, compiling lessons learned from the experiment, and using these lessons to guide future development.

### 4.5.1 SPHERES

The Massachusetts Institute of Technology (MIT) Space Systems Laboratory (SSL) developed the SPHERES, which have been on the International Space Station since 2006 [145]. An image of the robots is in Fig. 4.25. There is a set of three robotic spacecraft to enable formation flying testing. The SPHERES operate within a restricted 2 m cubed volume on the ISS, where they translate and rotate with the use of 12 compressed $\mathrm{CO}_{2}$ thrusters. The robots use an ultrasonic beacon system for localisation, consisting of a set of 5 static beacon stations and 24 receivers on the SPHERES. The time-of-flight of ultrasonic signals from each of the beacons is used to determine the position of the robots in the ISS reference frame. An onboard IMU provides estimates of orientation.

The robots are connected wirelessly to a ground station laptop that controls upload and download of software as well as executing the commands for a given test case. A Guest Scientist Program provides a


Figure 4.25. The Synchonized Position Hold, Engage, Reorient Experimental Satellites (SPHERES). (a) Three SPHERES on board the International Space Station. (b) A diagram of the main SPHERES components. Credit MIT SSL [145]
software interface that allows researchers to upload their code onto the SPHERES to control the higher level navigation functions.

The robots have been used for a range of research including: formation flying, visual navigation, mapping of small bodies, docking, electromagnetic relative position control and fluid slosh investigations [145]. The SPHERES system is designed to be robust and error tolerant, allowing one of the tremendous successes of the SPHERES program: Zero Robotics [146], a high school outreach program where students have the opportunity for their code to control the SPHERES [152].

### 4.5.2 On-Orbit Testing

The testing of ASTRO on the SPHERES was unique in that the Principal Investigator (PI) was himself on the ISS. To support this, MATLAB [133] was installed on the ground station laptop and linked with the SPHERES communication architecture to enable it to send planned trajectories to the SPHERES. Having MATLAB on the ground station enabled the PI to quickly and easily make modifications to the settings for the algorithm and test cases based on initial results. The setup is described in detail in [35].

Unfortunately, due to time restrictions for implementation, there were some limitations in the architecture, with only one set-point for a given trajectory being able to be sent to the SPHERES at a time, i.e. each planned trajectory resulted in one set-point being sent to the SPHERES, with the next set-point only available after replanning. Additionally, ASTRO was only able to be implemented with the MATLAB optimisation tool fmincon, rather than the full projected subspace gradient-descent, leading to longer computation times.

The goal of the tests was to demonstrate how ASTRO could be used for online trajectory planning for free-floating spacecraft, navigating around static and dynamic obstacles. Trajectories were replanned in regular intervals to a fixed goal location. Virtual static obstacles were modelled in the environment, and the dynamic obstacles were other SPHERES, physically moving in a set path.

Four test scenarios with a single goal location where executed:

1. A single static spherical obstacle.
2. Multiple static ellipsoid obstacles.
3. A single, dynamic obstacle moving on a fixed path.
4. Multiple static ellipsoid obstacles plus a dynamic obstacle moving in a circular path.

### 4.5.3 Results From On-Orbit Testing

The performance on a static spherical obstacle was successful. However, in a more complicated scenario with multiple ellipsoid obstacles, a vital issue became evident. With the transmission of only a single waypoint, the location of the waypoint needs to be set far enough along the trajectory so that the next planned trajectory will be ready before the SPHERES reaches that waypoint, as depicted in Fig 4.26.a. With large computation times (10-20 seconds), this meant that the set-point was so far along the trajectory that the SPHERES traversed through a modelled obstacle (Fig. 4.26.b). With long computation times, the satellites also drift, potentially making the path to the transmitted setpoint invalid. This issue implied that the SPHERES was not able to accurately track a planned trajectory. Nonetheless, the subsequent tests provided valuable information on how the trajectory planning performed in the range of scenarios in which the SPHERES operated. The focus here is the analysis of the dynamic obstacle results, being work performed by the author, and of significance to the developments of ASTRO.


Figure 4.26. Implementation limitations. (a) Intended result of computational lag compensation: to select a target along the trajectory to be ahead of the robot position when the next replanned trajectory is ready. (b) Example of the transmitted single target (open, white circles) oscillating back and forth across obstacle at each solver iteration, planned from the black circles. Grey ellipsoids are the simulated obstacles. Images are from [35].

In the first dynamic obstacle case, the obstacle robot (SPHERES 2) was commanded to move between fixed start points and end points. SPHERES 1, using ASTRO as an online planner was tasked with moving from a starting point before SPHERES 2 to a final point past the end point of SPHERES 2 (i.e. overtaking the second robot). In each optimisation cycle, the true state of SPHERES 2 was communicated
to MATLAB to create multiple a spherical obstacle constraints that effectively represented a cylinder extending from the current position along the velocity vector to a predicted final position (assuming constant velocity). Each of these constraints is modelled with a radius equal to double the radius of the SPHERES. This radius is to represent the obstruction one SPHERES robot would present to another.

The results from this test case, while still affected by the implementation limitations discussed above, did show that the planner could consider dynamic obstacles and adjust plans accordingly. For instance in Fig. 4.27 the planned trajectory has adjusted to curve around the constraint represented by the obstacle.


FIGURE 4.27. Planned trajectory (bold line) in one time instance around a real dynamic obstacle (red sphere), with history of estimated robot states (blue line for SPHERE 1 and red line for SPHERE 2) and transmitted targets (black stars). The trajectory is planned from the closed black circle to the open black circle. Target points are spread due to implementation limitations, but the planned trajectory is successful.

The second dynamic obstacle test case introduces a mix of virtual static obstacles and real dynamic obstacles. In this case, SPHERE 2 was set to orbit a central, virtual spherical obstacle; a trajectory that calls for more change in the velocity vector than the previous case, and hence presents a more significant challenge as a dynamic obstacle. SPHERE 1 was tasked to move diagonally through the permissible volume (a cube assigned inside the ISS), then traverse one of the sides of the permissible volume before moving back along the other diagonal (i.e. a set of three different goal locations). The test results demonstrate that the algorithm can handle complex scenarios with significant changes in the motion of dynamic obstacles. In Fig. 4.28 it can be seen that the trajectory adjusts to curve around the dynamic constraint successfully in the two different positions of the dynamic obstacle. Note that the actual trajectory did not follow the sequence of planned paths and waypoints due to the single target issues discussed above.


Figure 4.28. Planned trajectory (bold line) at a two time instances around a real dynamic obstacle (red shape) and simulated static obstacles (grey), with history of estimated robot states (blue line for SPHERE 1 and red line for SPHERE 2). The trajectory is planned from closed black circle to open black circle.

### 4.5.4 Lessons Learned

Despite the limitations in the implementation, the investigations were still able to demonstrate the use of ASTRO in planning on-line from arbitrary locations, in a manner that could deal with dynamic obstacles. There were also some important implementation lessons learned for on-line trajectory planning, particularly:

- It is recommended not to use infrequent single waypoint communication. A full trajectory or a section of a trajectory should be sent to the robot for it to follow.
- Careful consideration is required for designing the replanning strategy, in particular for:
- Selecting the commanded target along the trajectory to avoid potential oscillations in the commanded path.
- Selecting where to initiate a new planning cycle.
- Uncertainty in the computational lag makes designing the replanning strategy very challenging.
- A suitably tuned inner-loop feedback controller to closely track the trajectory is critical to have a reliable prediction of where the vehicle will be at a point in the future, to have smooth transitions to newly planned trajectories.

Relatively simple measures could address many of the issues experienced: implementing the full ASTRO algorithm and utilising the more modern hardware now available to enable the communication of a full trajectory and a reduced computation time. Nonetheless, the lessons learned are still valuable for implementation of on-line trajectory planning for other systems where the computation times are slow compared to the vehicle dynamics.

To address the computation time issues, development work on ASTRO following the on-orbit tests have looked to improve the optimisation efficiency, as presented in section 4.4.7. Alternatively, very long planning times could be accommodated by sending a whole trajectory and having highly accurate trajectory tracking, as described in the application to quadrotors in the Chapter 4.

The test results showed successful handling of dynamic obstacles, but they also identified areas for improvement. One observation was that the method of representing the dynamic obstacle constraint over time was too conservative, taking up too much of the feasible space that in reality was free to traverse. Additionally, the modelling of the dynamic constraints by many spherical obstacles resulted in a large number of constraints to check, slowing the optimisation computation. These observations have informed the development of the dynamic obstacles presented in section 4.3.5, which represent only where the obstacle is predicted to be, along with an increase in obstacle size to account for uncertainty. The simulation test cases presented in section 4.4 are representative of scenarios similar to what might be experienced for future applications of a SPHERES-like robot onboard the ISS.

### 4.6 Conclusion

On-orbit testing of an early version of the ASTRO algorithm demonstrated a capability for trajectory optimisation with real dynamic obstacles and simulated static obstacles. Analysis of these results and lessons learned informed further developments of ASTRO to enhance performance and expand the capability of the algorithm to the current state presented in this chapter. By parameterising a trajectory with Legendre polynomials, ASTRO can use a convex cost function that enables rapid optimisation of feasible trajectories when used with a subspace-projection approach to enforce boundary conditions. Adaptive constraint weighting along with iterative solutions to relaxed sub-problems allows for quick optimisation even in problems with many constraints. When the problem is non-convex, a random perturbation technique aids solutions by jumping trajectories out of infeasible local minima. ASTRO can include a broad range of constraints, making it applicable to many scenarios. Performance constraints can limit acceleration, both discrete obstacles and whole environments can be represented, or planning can be restricted to free-space bounds and an entirely convex problem. Dynamic obstacles have their location predicted, and size grown based on position uncertainty. This formulation opens more feasible space to plan a trajectory, compared to previous methods, while remaining safe.

The set of simulated test cases presented in this chapter demonstrate the capability of the algorithm in fulfilling the needs for the planning layer of the autonomous navigation stack. The application of this capability will be expanded to quadrotors in the next chapter, including considerations of the robot dynamics in the control layer. ASTRO is compared to the state-of-the-art in quadrotor trajectory planning with a batch of simulated test cases. Flight demonstrations of ASTRO running on autonomous quadrotors are then presented in Chapter 6, and the performance analysed. Chapter 7 describes the application of ASTRO in an autonomous navigation system by combining the algorithm with localisation and mapping layers from NURBSLAM.


## Trajectory Optimisation for Quadrotor UaVs



## Attributions:

The majority of this chapter has been presented in two publications, [153] and [154], that are primarily the work of the author of this thesis. In particular, [153] presents the work on differential flatness transformations, and [154] covers the comparison of trajectory optimisation algorithms. Co-authors for the papers contributed to idea development, coding implementation of existing planning algorithms, and discussion of results.

This chapter expands from what is presented in the two publications; providing more details on the theory of the differential flatness transformation and a more substantial comparison of trajectory optimisation algorithms.

Acombination of the planning and control layers of the autonomous navigation stack are addressed in this chapter for a particular robotic system: quadrotors. With the increasing capability of UAVs, in particular, quadrotors, there and many more use cases emerging, many of which demand higher levels of autonomy, a crucial part of which is trajectory planning.

Trajectory planning for quadrotors can take advantage of the differential flatness transformation that maps from a 3D position $x, y, z$, and heading, $\psi$, plus their derivatives through the full quadrotor state to the revolutions per minute (RPM) of the motors. The transformation provides a convenient way
to consider orientation and the full dynamics of the quadrotor, while enabling a wide range of trajectory optimisation techniques (such as ASTRO, the algorithm described in Chapter 4) to be applied.

The differential flatness transformation is a crucial part to quadrotor planning and control; hence the transformation is analysed in detail in this chapter, highlighting sensitivities with existing techniques, where they fail, and proposing new techniques to handle the sensitivities.

This chapter will then outline how ASTRO can be applied to quadrotor trajectory planning, extending from and applying the theory in Chapter 4, with consideration of quadrotor dynamics. ASTRO will then be compared against the state-of-the-art in theory and with simulations in obstacle-rich environments.

### 5.1 The Differential Flatness Transformation for Quadrotors


#### Abstract

Attribution: The majority of this section has been presented in [153], which is the work of the author of this thesis. More detail is presented here on the theory, and as well as analysis of more methods of performing the differential flatness transformation.


For autonomous navigation, two key components are a trajectory planner and a trajectory tracking controller. These components introduce many challenges when striving high-speed flight close to static and dynamic obstacles. This type of flight requires an ability to fly with high linear and angular accelerations, and is often referred to as aggressive flight [140]. For quadrotors, both trajectory planning and control can utilise a property called differential flatness [164, 219]. Differential flatness allows direct mapping from the flat-outputs of $x, y, z$ and $\psi$ (where $\psi$ is yaw), plus their derivatives, through the full quadrotor state to the flat-inputs: the revolutions per minute (RPM) squared for each motor. A continuous trajectory planned in the flat-output space gives continuous motor RPMs in the flatinput space, providing a convenient method to ensure a dynamically-feasible trajectory (given RPM magnitudes are within limits).

For trajectory tracking controllers, a hierarchical architecture is often used with an outer-loop position controller, and an inner-loop attitude controller [111, 118, 128], as depicted in Fig. 5.1. The outer-loop controller gives a desired thrust vector ( $\mathbf{T}_{s p}$ ) that is transformed through the differential flatness transformation to the desired attitude $\left(\mathbf{q}_{s p}\right)$ and thrust magnitude ( $T$ ). The inner-loop attitude controller tracks this attitude, outputting torques ( $\boldsymbol{\tau}$ ) that are mixed with $T$ to compute target RPMs for each motor controller.

This planning and control architecture utilising differential flatness has been widely employed to great effect [5, 26, 61, 63, 121, 139, 165, 216]. Additionally, the differential flatness transformation is commonly discussed as a key part for trajectory planning [111, 128] and is a core part of the controller by Lee [118].

Despite the frequent and successful use of the differential flatness transformation, there are known singularities that occur: 1) when there is zero thrust (when gravity fully achieves the desired acceleration), and 2) when the desired thrust vector is in the $x y$ plane and aligned with the desired direction of travel (e.g. pitched forward at $90^{\circ}$ ). The first singularity is a fundamental limitation of the transformation, founded on the notion that the desired thrust direction sets the quadrotor attitude. The
second singularity and the sensitivity of states near this singularity are because of the use of yaw to set the desired heading and is something that can be managed with a range of methods.

In this section, the differential flatness transform for quadrotors will be described, and the singularities highlighted. Existing methods to address the singularity are reviewed, and then a combined method is proposed that is designed to be robust through all orientations. An assessment of these methods is then made to demonstrate where issues can occur.

### 5.1.1 Description of the Transformation

A thorough description of the differential flatness transformation can be found in references such as [118] and [139]; the main steps are repeated here. The states, $x, y$, and $z$ and their derivatives up to snap, along with $\psi$ and its derivatives up to acceleration, are required to perform the full transformation.


Figure 5.1. Block diagram of the hierarchical tracking controller considered in this work. Here, $\boldsymbol{x}_{s p}$ and $\psi_{s p}$ are the position and yaw set points for the outer position controller, which outputs the thrust set point $\mathbf{T}_{s p}$. The differential flatness transformation provides the attitude set point $\mathbf{q}_{s p}$ to the inner attitude controller. The attitude controller outputs torques ( $\boldsymbol{\tau}$ ) that are mixed with thrust magnitude $T$ to compute motor RPMs.


Figure 5.2. Quadrotor body-axes (subscript b), global axes (subscript $g$ ), interim yaw axes (subscript $c$ ), and second angle axes (subscript $c_{s a}$, with angle $\gamma$ ). East, North, Up (ENU) convention.

### 5.1.1.1 Orientation

The starting assumption is that the thrust vector of the quadrotor sets the direction of the $z$ body-axis, $\boldsymbol{z}_{b}$ (see Fig. 5.2). Hence, from a planned trajectory that gives the desired acceleration $\ddot{\boldsymbol{x}}_{s p}$, the thrust vector, $\mathbf{T}$, is defined, that sets the direction of the $z$ body-axis, $\boldsymbol{z}_{b}$ :

$$
\begin{array}{r}
\mathbf{T}=\ddot{\boldsymbol{x}}_{s p}-\boldsymbol{z}_{g} \tilde{g} \\
\boldsymbol{z}_{b}=\frac{\mathbf{T}}{\|\mathbf{T}\|} \tag{5.2}
\end{array}
$$

where $\tilde{g}$ is the acceleration due to gravity, and $\boldsymbol{z}_{g}$ the global $z$ unit vector. The subscript $g$ denotes the global axes. The first type of singularity occurs in Eq. 5.2 when the thrust magnitude is zero.

The desired yaw angle, $\psi_{s p}$ sets the vector $\boldsymbol{x}_{c}$ : a unit vector in the $x y$ plane pointing in the desired heading direction. The cross product of $\boldsymbol{z}_{b}$ with $\boldsymbol{x}_{c}$ gives an orthogonal vector, $\boldsymbol{y}_{b}$, that represents the $y$ body-axis direction. The cross product of this vector with $\boldsymbol{z}_{b}$ gives a third orthogonal vector $\boldsymbol{x}_{b}$; this provides the $x$ body-axis, giving the overall orientation $\mathbf{R}$ :

$$
\begin{align*}
\boldsymbol{x}_{c} & =\left[\begin{array}{ll}
\cos \left(\psi_{s p}\right), & \sin \left(\psi_{s p}\right), \\
0
\end{array}\right]^{T}  \tag{5.3}\\
\boldsymbol{y}_{b} & =\frac{\boldsymbol{z}_{b} \times \boldsymbol{x}_{c}}{\left\|\boldsymbol{z}_{b} \times \boldsymbol{x}_{c}\right\|}  \tag{5.4}\\
\boldsymbol{x}_{b} & =\boldsymbol{y}_{b} \times \boldsymbol{z}_{b}  \tag{5.5}\\
\mathbf{R} & =\left[\begin{array}{lll}
\boldsymbol{x}_{b}, & \boldsymbol{y}_{b}, & \boldsymbol{z}_{b}
\end{array}\right] \tag{5.6}
\end{align*}
$$

The second source of singularities occurs in Eq. 5.4 , when $\boldsymbol{z}_{b} \times \boldsymbol{x}_{c}=0$, i.e. when $\boldsymbol{z}_{b}$ is parallel with $\boldsymbol{x}_{c}$.

### 5.1.1.2 Angular Rates

The angular rates, $\boldsymbol{\omega}$, are computed using both jerk, $\boldsymbol{x}^{(3)}$, and the $z$ body-axis. The pitch and roll rates ( $\omega_{1}$ and $\omega_{2}$ ) can be extracted using:

$$
\begin{align*}
\mathbf{h}_{w} & =\frac{m}{T}\left(\boldsymbol{x}^{(3)}-\left(\boldsymbol{z}_{b} \cdot \boldsymbol{x}^{(3)}\right) \boldsymbol{z}_{b}\right)  \tag{5.7}\\
\omega_{1} & =-\mathbf{h}_{w} \cdot \boldsymbol{y}_{b}  \tag{5.8}\\
\omega_{2} & =\mathbf{h}_{w} \cdot \boldsymbol{x}_{b} \tag{5.9}
\end{align*}
$$

The yaw rate, $\omega_{3}$, is then extracted by projecting the rate of change of yaw, $\dot{\psi}$, onto the z body-axis, $z_{b}$.

$$
\begin{equation*}
\omega_{3}=\dot{\psi} \boldsymbol{z}_{g} \cdot \boldsymbol{z}_{b} \tag{5.10}
\end{equation*}
$$

### 5.1.1.3 Angular acceleration

Angular acceleration, $\dot{\boldsymbol{\omega}}$ proceeds similarly to angular velocity, using snap, $\boldsymbol{x}^{(4)}$, with a utility vector $\mathbf{h}_{\alpha}$ first being computed:

$$
\begin{align*}
\mathbf{h}_{\alpha} & =\frac{m}{T}\left[\boldsymbol{x}^{(4)}-\left(\boldsymbol{z}_{b} \cdot \boldsymbol{x}^{(4)}\right) \boldsymbol{z}_{b}\right.  \tag{5.11}\\
& +\frac{T}{m}\left(\boldsymbol{z}_{b} \cdot\left(\boldsymbol{\omega} \times \boldsymbol{\omega} \times \boldsymbol{z}_{b}\right)\right) \boldsymbol{z}_{b} \\
& -\frac{T}{m} \boldsymbol{\omega} \times \boldsymbol{\omega} \times \boldsymbol{z}_{b} \\
& \left.\left.-2 \boldsymbol{\omega} \times\left(\boldsymbol{z}_{b} \cdot \boldsymbol{x}^{(3)}\right) \boldsymbol{z}_{b}\right)\right]
\end{align*}
$$

that then gives the first two rotational accelerations:

$$
\begin{align*}
& \dot{\omega}_{1}=-\mathbf{h}_{\alpha} \cdot \boldsymbol{y}_{b}  \tag{5.12}\\
& \dot{\omega}_{2}=\mathbf{h}_{\alpha} \cdot \boldsymbol{x}_{b} \tag{5.13}
\end{align*}
$$

Deriving the third angular acceleration uses the yaw acceleration:

$$
\begin{equation*}
\dot{\omega}_{3}=\ddot{\psi} \boldsymbol{z}_{w} \cdot \boldsymbol{z}_{b} \tag{5.14}
\end{equation*}
$$

### 5.1.1.4 Controls

The extraction of the torques, $\boldsymbol{\tau}$, uses the angular accelerations and moments of inertia $\bar{I}$ :

$$
\begin{equation*}
\boldsymbol{\tau}=\bar{I} \dot{\omega}+\omega \times \bar{I} \omega \tag{5.15}
\end{equation*}
$$

The torques, along with the thrust magnitude (length of thrust vector in Eqn. 5.1), gives the control inputs, which can be mapped back to the required $\mathrm{RPM}, \Omega$, through:

$$
\left[\begin{array}{l}
\Omega_{1}{ }^{2}  \tag{5.16}\\
\Omega_{2}^{2} \\
\Omega_{3}^{2} \\
\Omega_{4}{ }^{2}
\end{array}\right]=\left[\begin{array}{cccc}
k_{f} & k_{f} & k_{f} & k_{f} \\
0 & k_{f} L & 0 & -k_{f} L \\
-k_{f} L & 0 & k_{f} L & 0 \\
k_{m} & -k_{m} & k_{m} & -k_{m}
\end{array}\right]^{-1}\left[\begin{array}{c}
\|\mathbf{T}\| \\
\tau_{1} \\
\tau_{2} \\
\tau_{3}
\end{array}\right]
$$

The variables $k_{f}$ and $k_{m}$ are the thrust and torque coefficients respectively (i.e. Force ${ }_{1}=k_{f} \Omega_{1}{ }^{2}$ ), and $L$ is the moment arm of the rotors about the centre of mass.

### 5.1.1.5 Differential Flatness and Optimisation of Snap

The transformation progresses from the flat-outputs, of $x, y, z$ and $\psi$ through to the flat-inputs, of the RPM squared, $\Omega^{2}$. These steps also compute the full quadrotor state, including the attitude, angular rates, angular accelerations, force and torques. The reason for minimising snap in trajectory optimisation is now more clear with the full explanation of transformation. The RPM squared, $\Omega^{2}$, is
directly proportional to the torques, $\boldsymbol{\tau}$ (Eqn. 5.16), which in turn are directly proportional to the angular acceleration, $\dot{\omega}$ (Eqn. 5.15). The angular accelerations for pitch and roll are then directly proportional to the $\mathbf{h}_{\alpha}$ vector (Eqn. 5.12). $\mathbf{h}_{\alpha}$ is is a linear function of snap, $\boldsymbol{x}^{(4)}$, (Eqn. 5.11). Hence snap has a linear relationship to the RPM squared, $\boldsymbol{x}^{(4)} \propto \Omega^{2}$. This relationship means that by minimising snap, there is a direct minimisation of the RPM squared. Similarly, the yaw torque is directly proportional to the yaw acceleration; this is why it is desired to minimise the yaw acceleration in trajectory optimisation.

### 5.1.2 Singularities

There are two singularities in the transformation, which were identified above. Firstly, in Eqn. 5.2 when there is zero desired thrust (when gravity fully achieves the desired acceleration). Secondly, in Eqn. 5.4 when the desired thrust vector is in the $x y$ plane and aligned with the desired direction of travel (e.g. pitched forward at $90^{\circ}$ ).

The first singularity is a fundamental limitation of the transformation, which is founded on the notion that the desired thrust direction sets the quadrotor attitude. This singularity can be avoided though by setting a constraint on the minimum thrust in both the planning algorithms and controllers.

The second singularity, when $\boldsymbol{z}_{b}$ is parallel with $\boldsymbol{x}_{c}$, can occur in two orientations, with both positive $\boldsymbol{x}_{c}$ and negative $\boldsymbol{x}_{c}$. While it is not likely that a quadrotor will exactly reach this singularity, the transformation is highly sensitive at states near this singularity, when $\left\|\boldsymbol{z}_{b} \times \boldsymbol{x}_{c}\right\|$ is small. Manoeuvres that have the quadrotor thrust vector passing through the horizontal plane have a chance of experience the sensitivity in the transform, resulting in erroneous or rapidly changing output transformations, as will be shown in the following sections and flight tests in Chapter 6. Therefore, the sensitivity of states near this second singularity is something that needs to be managed carefully. Methods to address the second singularity will be explored here, assessing existing methods, and proposing new methods, with the goal of supporting flight through all range of orientations. Throughout the rest of this thesis, singularity refers to this second type.

### 5.1.3 Existing Methods to Address the Singularity

In this section, a range of methods to address the singularity in the differential flatness transform are reviewed. In the subsequent subsection, each of these approaches will be analysed to assess their limitations.

### 5.1.3.1 Standard Method

The transformation described in the previous section is referred to as the Standard transformation. Many quadrotor systems use the Standard transformation without any modifications [5, 62, 118, 165] and it is the transformation that is described in review papers [111, 128]. Numerous quadrotor systems applications have used the controller by Lee et al. [118], which assumes the singularity is not met. These applications have been very effective, such as Allen et al. [5] navigating through crowded indoor environments, and deal with the singularity by operating in a regime far from it: flying with relatively small deviations from the hover state. Falanga et al. [62] and Neunert et al. [165] push the dynamics up
to $45^{\circ}$ and $30^{\circ}$ of roll respectively, using the Standard transformation, however, their demonstrations are far from the singularity at $90^{\circ}$.

### 5.1.3.2 Negative Check Method

Mellinger et al. [139] demonstrate more aggressive flight, with up to $90^{\circ}$ of roll through a window, operating in states that could be approaching the singularity in the transformation (also subsequently demonstrated by Loianno et al. [121]). They note that the negative $x$ and $y$ body-axes can be consistent with the desired yaw angle and desired $\boldsymbol{z}_{b}$, and check which of $\left(\boldsymbol{x}_{b}, \boldsymbol{y}_{b}, \boldsymbol{z}_{b}\right)$ or $\left(-\boldsymbol{x}_{b},-\boldsymbol{y}_{b}, \boldsymbol{z}_{b}\right)$ are closest to the current orientation. The closest axes-set is then used as the output of the transformation. The approach of [139] has been used very effectively for autonomous navigation in ships [63], and through indoor environments [26], however only with moderate roll or pitch angles. [139] and [121] only demonstrate roll in isolation and do not exceed $90^{\circ}$; therefore they do not operate near the sensitive regions of the transformation.

### 5.1.3.3 Second Angle Method

Thomas et al. [216] use a method that enables orientations holding at $90^{\circ}$ for an extended period, and extending past $90^{\circ}$, with flight demonstrations of a quadrotor perching on both vertical surfaces and the underside of inclined surfaces. To move away from the singularity their approach employs an additional working angle, $\gamma$, to rotate the $\boldsymbol{x}_{c}$ vector in the vertical plane:

$$
\boldsymbol{x}_{c}=\left[\begin{array}{lll}
\cos (\psi) \cos (\gamma), & \sin (\psi) \sin (\gamma), & \sin (\gamma) \tag{5.17}
\end{array}\right]
$$

This second angle moves $\boldsymbol{x}_{c}$ away from the $x y$ plane to avoid being parallel with $\boldsymbol{z}_{b}$. Fig. 5.2 illustrates the movement of an axes-set from $\gamma$. While their approach moves the location of singularity, it does not remove it. The demonstrations in [216] show the effectiveness of their approach; however, the examples only operate with one axis of rotation (pitch), rather than the full range of quadrotor motion.

### 5.1.3.4 Angular Rates

Another approach to the differential flatness transformation is described in [87, 157]. They take the flat-outputs of $x, y, z$, and their derivatives up to jerk, to get inputs of the $x$ and $y$ angular rates ( $\omega_{x}$ and $\omega_{y}$ ) and thrust. The $z$ angular rate ( $\omega_{z}$ ) is user set, normally to a constant 0 . Their transformation avoids the singularity but relies on the assumption that the current attitude is already in the correct direction for the planned acceleration. This approach is shown to be sufficient for agile manoeuvres, but in cases with highly accurate state estimation (with external tracking systems), and low tracking error. In systems with lower quality estimates of attitude, more significant disturbances and more substantial tracking errors, the core assumption: that the desired acceleration is aligned with the current attitude, is likely to break down.

### 5.1.3.5 Inverted flight

None of the previous work described above has demonstrated highly-aggressive flight with $360^{\circ}$ rotations through inversion. Some examples demonstrate such flight, but these tend to employ switching controllers that split the trajectory into simplified segments including launch, ballistic trajectory, constant roll rates, and recovery. These methods either learn a set of parameters for the simplified manoeuvre, such as multiple flips [126, 140], or fit a model to the flight of an expert pilot [1, 71]. The segmentation and simplification of the manoeuvres avoid any singularities; however, this approach is restricted to a limited range of dynamics.

### 5.1.3.6 Recovery

One of the key components of aggressive manoeuvres, such as those mentioned above, is the recovery phase. Faessler et al. [61] developed an autonomous recovery controller that manages the differential flatness transformation effectively. Their approach splits control of the pitch and roll rates from the control of the yaw rate. First, the angular error for pitch and roll is derived, independent of the singularity, to give the correct $\boldsymbol{z}_{b}$. Multiplying the errors by a gain gives desired pitch and roll rates. The additional rotation required due to yaw is computed using the standard differential flatness transformation, but as the roll and pitch rates are already determined, the yaw rate can simply be set to zero when the singularity is met. Additionally, if the $\boldsymbol{z}_{b}$ axis is pointing down, then negative $\boldsymbol{x}_{c}$ is used, very similar to the default transformation in the PX4 flight controller [138]. This approach was shown to be successful as part of aggressive manoeuvres in Falanga's work [62], including from inverted attitudes. However, these recoveries are to a commanded horizontal attitude, and hence the system is never commanding $90^{\circ}$ pitch or roll or inverted orientations, where the sensitivities occur.

### 5.1.3.7 Summary

Each of the differential flatness transformation methods described above has limitations that can cause the transformation to fail in certain scenarios. While the singularity may be avoided, the next section shows that there are still issues around the singularity and in transitions near it, where the transformations give large changes in attitude with a small change in the acceleration vector.

### 5.1.4 Analysis of Differential Flatness Transformations

This section presents an analysis of the transformation methods described in the previous section, highlighting where issues may arise. New methods are proposed that address these issues, which are subsequently analysed. A summary of this analysis is presented in Table 5.1. The goal is to show the limitations of each method and identify the best method for applying differential flatness throughout the entire flight envelope of a quadrotor.

### 5.1.4.1 Standard

The Standard transformation suffers directly at the singularity, yet of greater concern, is the change in attitude on either side of the singularity. For example; when the $\boldsymbol{z}_{b}$ axis transitions from above the $x y$
plane to below in a pitching forward manoeuvre (Fig. 5.3), the direction of the $\boldsymbol{y}_{b}$ axis coming out of the cross product in Eq. 5.4 flips $180^{\circ}$. This flip of the $y$-axis then flips the $\boldsymbol{x}_{b}$ axis, giving a substantial change in attitude.


Figure 5.3. Pitching forward by $10^{\circ}$ through a $90^{\circ}$ pitch angle, and the singularity. Desired attitude is commanded through the $z$ body-axis (acceleration) and desired yaw angle (zero here). (a) Standard method failing. (b) Negative Check method succeeding. Axes slightly before $90^{\circ}$, are in blue, and axes slightly after $90^{\circ}$ are in red. The black dashed axes represent the global coordinate frame.

### 5.1.4.2 Negative Check

A solution to this issue is to take the Negative Check method proposed by Mellinger [139], or the PX4 method [138]. Negating $\boldsymbol{x}_{b}$ and $\boldsymbol{y}_{b}$, maintains a similar attitude through the transition (see Fig. 5.3.b). The Negative Check approach is still susceptible at the singularity though and is sensitive when pitching through $90^{\circ}$ near the singularity (Fig. 5.4.a). Both the Negative Check method and the PX4 method can have issues when there are fast dynamics near the singularity.

### 5.1.4.3 Second Angle

The Second Angle method described by Thomas et al. [216], is successfully able to avoid the singularity by moving where it is encountered, but introduces challenges in controlling the second angle, $\gamma$, when striving for robust performance through all orientations. Setting a constant angle only shifts the singularity. Not controlling $\gamma$ correctly could also lead to discontinuous jumps. If the approach is taken to set $\gamma$ so that $\boldsymbol{x}_{c}$ is always chasing or leading $\boldsymbol{z}_{c}$, a $360^{\circ}$ pitching manoeuvre can successfully be described. Nevertheless, this same approach will fail when rolling through $90^{\circ}$. If this approach is combined with checking the negative set of the result, then the second angle approach can perform well in most scenarios.


FIGURE 5.4. Pitching forward by $10^{\circ}$ through a $90^{\circ}$ pitch angle near the singularity (yaw at $5^{\circ}$ ). Desired attitude is commanded through the $z$ body-axis (acceleration) and desired yaw angle. (a) Negative Check method failing. (b) Combined methods succeeding. Axes before the transition are in blue and axes after the transition are in red. The black dashed axes represent the global coordinate frame.

### 5.1.5 New Approaches to Address the Singularity

Continuing from the discussion on the limitations of methods to address the singularity, several methods are proposed and analysed here, taking inspiration from other methods in the literature.

### 5.1.5.1 Using Other Axes

Another approach for dealing with the singularity is to recognise that it is possible to use both $\boldsymbol{x}_{c}$ and $\boldsymbol{y}_{c}$ as the axis in the intermediate frame, when taking the problematic cross product in Eq. 5.4. i.e. the body-axes could be equivalently determined by:

$$
\begin{align*}
& \boldsymbol{y}_{c}=\left[\begin{array}{lll}
-\sin (\psi), & \cos (\psi), & 0
\end{array}\right]^{T}  \tag{5.18}\\
& \boldsymbol{x}_{b}=\frac{\boldsymbol{y}_{c} \times \boldsymbol{z}_{b}}{\left\|\boldsymbol{y}_{c} \times \boldsymbol{z}_{b}\right\|}  \tag{5.19}\\
& \boldsymbol{y}_{b}=\boldsymbol{z}_{b} \times \boldsymbol{x}_{b} \tag{5.20}
\end{align*}
$$

Two methods can be used to select the best vector to use: 1) taking the vector that is closest to $90^{\circ}$ from $\boldsymbol{z}_{b}$ (the Check Furthest method) or 2) selecting the vector that gives a resulting $\boldsymbol{x}_{b}$ closest to the current $\boldsymbol{x}_{b}$ (the Check Current $\boldsymbol{x}_{b}$ method). These approaches can be quite effective, but still have sensitive regions: when transitioning through the decision point for $\boldsymbol{x}_{c}$ and $\boldsymbol{y}_{c}$ (Fig. 5.5.b), and when the body $x$-axis is moving through a vertical orientation (Fig. 5.5.a).


Figure 5.5. Two examples where the differential flatness transformations fail. The desired attitude is commanded through the $z$ body-axis (acceleration) and desired yaw angle. (a) Pitching backward by over $100^{\circ}$ through a $-90^{\circ}$ pitch angle with zero yaw: Negative Check, Check Furthest and Check Current $x_{b}$ methods fail with a $180^{\circ}$ flip in the $y$-axis. The scenario comes from a split-S manoeuvre. (b) A small change in yaw with thrust in the $x y$ plane: Check Furthest and PX4 methods fail with $90^{\circ}$ rotation in $x$ and $y$ axes. The axes before the transition are in blue, and the axes after the transition are in red. The black dashed axes represent the global coordinate frame.

Alternatively, the current body-axes, $\boldsymbol{x}_{b}$ and $\boldsymbol{y}_{b}$ can be used in place of $\boldsymbol{x}_{c}$ and $\boldsymbol{y}_{c}$, ensuring that the axes used in the cross products will always be nearly orthogonal to $\boldsymbol{z}_{b}$ (referred to here as the Use Current $\boldsymbol{x}_{b}$ method). While effective, this approach sacrifices the ability to control yaw, instead simply maintaining the previous orientation, or drifting with the current attitude estimate.

### 5.1.5.2 Quaternions

Recognising that some of the issues experienced are due to the use of Euler angles (as yaw is used to describe orientation), quaternions could instead be used in the differential flatness transformation. Rather than operating with the flat-outputs of $x, y, z, \psi$ and their derivatives, this approach uses $x, y, z$, and $q_{3}$, the $\mathbf{z}$ component of the vector part of the quaternion. Details of this representation are presented in Appendix D. Issues with a large invalid region around $180^{\circ}$ yaw reduce the effectiveness of the current version of this approach.

### 5.1.5.3 Pitch and Roll Only

If the application is agnostic to the yaw angle, then a method inspired by the angular error calculations in [61] can be used. The required rotation from the global $z$-axis, $\boldsymbol{z}_{g}$, to the desired body $z$-axis, $\boldsymbol{z}_{b}$, can be described by a quaternion (q), giving the attitude of the vehicle:

$$
\begin{align*}
& \beta=\arctan 2\left(\frac{\left\|\boldsymbol{z}_{g} \times \boldsymbol{z}_{b}\right\|}{\boldsymbol{z}_{g} \cdot \boldsymbol{z}_{b}}\right)  \tag{5.21}\\
& \mathbf{n}=\frac{\boldsymbol{z}_{g} \times \boldsymbol{z}_{b}}{\left\|\boldsymbol{z}_{g} \times \boldsymbol{z}_{b}\right\|}  \tag{5.22}\\
& \mathbf{q}=\left[\begin{array}{c}
\cos (\beta / 2) \\
\mathbf{n} \sin (\beta / 2)
\end{array}\right] \tag{5.23}
\end{align*}
$$

where $\beta$ is the angle of rotation and $\mathbf{n}$ is the axis of rotation. When $z_{g}$ is parallel to $z_{b}$, the orientation is known: the identity quaternion. This method avoids any issues with singularities or sensitive regions but does not grant any control of yaw.

### 5.1.5.4 Combined Methods

The proposed methods to address the issues with the singularity, while maintaining the ability to set yaw, is to combine methods. The underlying philosophy is to compute the transformation with multiple different approaches, and then select the result giving the smallest change in orientation. The body-axes can be computed with $\boldsymbol{x}_{c}, \boldsymbol{y}_{c}$ and the second angle approach (using a constant, fixed $\gamma=15^{\circ}$ ). These computations give three possible solutions, plus another three from taking the negative $\boldsymbol{x}_{b}$ and $\boldsymbol{y}_{b}$ of their results, giving a total of six axes-sets. Each of these axes-sets can be compared against the previously computed orientation ${ }^{1}$ and the closest matching result selected.

Using just four axes-sets ${ }^{2}$, referred to as the Four Axes Combined method, can achieve good performance for the scenarios above. However, issues can still be encountered, as depicted in Fig. 5.6.a for a scenario of pitching when at $90^{\circ}$ roll, a scenario that was experienced in flight.

[^13]
(a)

(b)

Figure 5.6. Pitching by $5^{\circ}$ while at $90^{\circ}$ roll (yaw at $90^{\circ}$ ). Desired attitude is commanded through the $z$ body-axis (acceleration) and desired yaw angle. (a) Four Axes Combined method failing. (b) Six Axes Combined method succeeding. Axes before the transition are in blue and axes after the transition are in red. The black dashed axes represent the global coordinate frame.

Introducing more axes from the second angle method gives an orientation option of lower error in the region of sensitivity. This combination is referred to as the Six Axes Combined method. Fig. 5.7 shows the selection of the closest axes result and the error for each axes-set throughout a trajectory. At the point where $90^{\circ}$ of roll is reached, there is a substantial change in the axes using $\boldsymbol{x}_{c}, \boldsymbol{y}_{c}$, as shown by the large changes in the axes errors (Fig. 5.7.a centre). The second angle method is not as sensitive in that same region and can maintain a feasible trajectory, (Fig. 5.7.b centre, purple line).

The Six-Axes Combined method is robust enough to cover the spread of possible dynamic transitions tested without large discrete changes in orientation.


Figure 5.7. Closest axes selection and error between each axes-set and the current orientation. Values presented through an aggressive roll manoeuvre with pitching when at $90^{\circ}$ roll. Two methods compared: (a) Four Axes Combined method, (b) Six Axes Combined method. For (a) and (b), top: selected axes index, center: error for each axes-set and bottom: resulting orientation as Euler angles. The Four Axes method results in $180^{\circ}$ yaw rotation. The Six Axes method does not and has a smaller error throughout. Axes order for errors is: 0: $\boldsymbol{x}_{c}, 1: \boldsymbol{x}_{c}$ negated, 2: $\boldsymbol{y}_{c}, 3: \boldsymbol{y}_{c}$ negated, 4: second angle, 5: second angle negated.

### 5.1.6 Summary of Analysis

The performance of each of the approaches for a range of challenging scenarios is summarised in Table 5.1. A method is deemed to have failed if there is a discontinuous jump or an extremely rapid, and large, change in attitude. Across the full range of possible dynamics, the Six Axes Combined method is the best approach, being the only method to succeed in all cases and still enable control of yaw.

The analysis of the different methods for dealing with the singularity is hoped to provide insight into the limitations of the different approaches, to enable selection of the most appropriate method for a given application.

While the Six Axes Combined method performs the best over the range of scenarios, it comes with approximately three times greater computational expense than the Standard method. If the application for a quadrotor will not have any large expected roll or pitch angles, then the Standard method may be sufficient. If control of yaw is not essential in the application, then the Pitch-and-Roll-Only method is a

TABLE 5.1. Performance summary of differential flatness methods.

|  |  | Test Case |  |  |  |  |  |  |  |
| :--- | :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Method | Section | A | B | $\mathbf{C}$ | $\mathbf{D}$ | $\mathbf{E}$ | F | G | All |
| Standard | $(5.1 .3 .1)$ | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| Neg. Check[139] | $(5.1 .3 .2)$ | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| PX4 [61, 138] | $(5.1 .3 .2)$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 2nd Angle[216] | $(5.1 .3 .3)$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Check Furthest | $(5.1 .5 .1)$ | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| Check Current $\boldsymbol{x}_{b}$ | $(5.1 .5 .1)$ | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| Quaternions* | $(5.1 .5 .2)$ | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| ${\text { Use Current } \boldsymbol{x}_{b}{ }^{*}}^{(5.1 .5 .1)}$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |
| Pitch and Roll* | $(5.1 .5 .3)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 Axes Comb. | $(5.1 .5 .4)$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 6 Axes Comb. | $(5.1 .5 .4)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

For each Method and Test Case, 1 is pass and 0 is fail. Test Case A:
Singularity. B: Pitch $90^{\circ}$ through singularity. C: Pitch $90^{\circ}$ yaw $5^{\circ}$
D: Roll through $90^{\circ}$ with $180^{\circ}$ yaw. E: Yaw with thrust in $x y$ plane.
F: Pitch backwards through $90^{\circ}$. G: Pitch at $90^{\circ}$ roll, through $x$-axis.
*Full control of yaw not possible.
good selection, providing a quick and straightforward method for computing the required orientation.
For highly-aggressive trajectories through $90^{\circ}$ and inversion, the Six Axes Combined method is selected and demonstrated in both simulated and real-world flights in Chapter 6.

### 5.2 ASTRO for Quadrotors

The state-of-the-art algorithms for planning trajectories for quadrotors use polynomial optimisation of $x$, $y, z$, and yaw $(\psi)$ to minimise the integral of snap squared between a set of waypoints. As was described in Section 5.1.1.5, snap is directly proportional to RPM squared, hence a smooth and minimised snap trajectory gives smooth and minimised motor RPMs. A sampling-based planner, such as RRT* [102], is most commonly used to generate the waypoints between which a trajectory is optimised. For obstacle avoidance, the leading approaches either use a check-and-adjust approach, with the pre-planned, known obstacle-free path [26, 63], an obstacle representation of the environment [170], or sets of convex regions within which to plan [32, 114, 147].

The approach taken here for planing with quadrotors is similar to the state-of-the-art: using a polynomial optimisation with separate polynomials for $x, y, z$ and $\psi$, and the objective to minimise snap for $x, y, z$ and acceleration for $\psi$. The trajectory is planned in multiple segments between waypoints, which could come from a sampling-based planner, by a flown and recorded trajectory, or by user input.

As will be described in this section, only minor modifications are required for ASTRO to be applied to quadrotors. While similar to the state-of-the-art techniques, the key differences with ASTRO are how the optimisation considers obstacles, the types of obstacle representations that are possible and the method of optimisation.

### 5.2.1 Modifications to ASTRO for Application to Quadrotors

The first change for ASTRO is adding a dimension to the standard $x, y, z$, to include $\psi$. The second change is to have the integral of snap squared as the cost function; this change involves choosing $\xi=4$ in Eq. 4.1 and 4.2. The snap trajectory for dimension $i$ is then represented by:

$$
\begin{equation*}
x_{i}^{(4)}\left(t^{\prime}\right)=\sum_{k=0}^{N} C_{i k} P_{k}\left(t^{\prime}\right) \tag{5.24}
\end{equation*}
$$

With the cost function unchanged (Eq. 4.8). Boundary and continuity conditions are considered up to snap ( $q=\xi=4$ in Eq. 4.24) and are implemented to have all derivatives fixed for the first and last waypoints, and only the position fixed for inner waypoints. This configuration of boundary conditions, along with the continuity conditions restricts six degrees-of-freedom per segment: five at the start of a segment from continuity constraints and one at the end for the position boundary condition. The final segment grants no additional degrees-of-freedom with continuity constraints at the start and boundary conditions at the end.

In the differential flatness transformation, the yaw acceleration is directly proportional to the RPM squared (see Section 5.1.1.5), similarly to how snap is for position, hence yaw is optimised with $\xi=2$, i.e., the cost is the integral of acceleration squared, and the yaw acceleration is represented with Legendre polynomials:

$$
\begin{equation*}
\psi^{(2)}\left(t^{\prime}\right)=\sum_{k=0}^{N} C_{i k} P_{k}\left(t^{\prime}\right) \tag{5.25}
\end{equation*}
$$

### 5.2.2 Comparison with Existing Planners

The key areas of contrast between ASTRO and leading algorithms in literature are the method of optimising snap and the method of considering obstacles. These areas will be expanded below. In the discussion, ASTRO will be compared in particular to two leading algorithms from the literature: from Bry et al. [26] and from Campos-Macias et al. [32]. These planners will further be compared with simulations and flight tests in later sections and chapters.

### 5.2.2.1 Snap Optimisation

The changes in parameterisation to adapt ASTRO for quadrotors allows the subspace-projection gradient-descent optimisation to be used for $x, y$ and $z$ to minimise snap, and for $\psi$ to minimise acceleration. This projected gradient-descent method of snap optimisation differentiates ASTRO from the state-of-the-art. The review of literature in Section 2.3, outlined how the optimisation of snap is proposed, for a single dimension, as a quadratic program that is constrained by boundary conditions:

$$
\begin{equation*}
\min _{\mathbf{C}_{i}} \mathbf{C}_{i}^{T} \boldsymbol{W} \mathbf{C}_{i} \quad \boldsymbol{G} \mathbf{C}_{i}=\mathbf{d}_{i} \tag{5.26}
\end{equation*}
$$

where the $\boldsymbol{W}$ matrix encapsulates the snap squared cost, $\mathbf{C}_{i}$ are the polynomial coefficients to be optimised, and $\boldsymbol{G} \mathbf{C}_{i}=\mathbf{d}_{i}$ are the equality constraints representing the boundary conditions. A direct solution can be computed using Lagrange multipliers if the number of coefficients (the order of the
polynomial) is equal to the maximum number of boundary conditions for a segment: e.g. 10 if boundary conditions are included up to snap. This direction solution is [63]:

$$
\begin{equation*}
\mathbf{C}_{i}=\boldsymbol{W}^{-1} \boldsymbol{G}^{T}\left(\boldsymbol{G} \boldsymbol{W}^{-1} \boldsymbol{W}^{T}\right)^{-1} \mathbf{d}_{i} \tag{5.27}
\end{equation*}
$$

The $\boldsymbol{W}$ matrix can often be ill-conditioned though, making inversion problematic; hence adjustments are made, such as by Fang et al. [63] who add a regularisation term: $\boldsymbol{W}=\boldsymbol{W}+\epsilon \boldsymbol{I}$ to make the matrix invertible, where $\epsilon$ is a small number and $\boldsymbol{I}$ is the identity matrix. In contrast, Bry et al. [26] modify the problem via substitution of the derivatives at the waypoints, $\mathbf{C}_{i}=\boldsymbol{G}^{-1} \mathbf{d}_{i}$, to give a cost function of:

$$
\begin{equation*}
\min _{\mathbf{d}_{i}} \mathbf{d}_{i}^{T} \boldsymbol{G}^{-T} \boldsymbol{W} \boldsymbol{G}^{-1} \mathbf{d}_{i} \tag{5.28}
\end{equation*}
$$

where $\mathbf{d}_{i}$ includes all derivatives at each waypoint. Only the free-derivatives need to be solved for; therefore the matrix $\boldsymbol{G}^{-T} \boldsymbol{W} \boldsymbol{G}^{-1}$ is partitioned to simplify the problem. The substitution for the derivatives inherently enforces the boundary conditions, translating the problem into an unconstrained optimisation. Other approaches, such as by Thomas et al. [216], use commercial optimisers to generate solutions, but for more complex scenarios with the inclusion of inequality constraints.

The subspace-projection gradient-descent optimisation of ASTRO has three main benefits over existing approaches. Firstly, higher order polynomials can be used to represent the trajectory. Additionally, all dimensions can be considered together in the optimisation, rather than being optimised separately. Finally, the formulation allows for incorporation of a range inequality constraints. These benefits are elaborated below.

The order of the polynomial for ASTRO is flexible and can be increased as desired. In contrast, the substitution approach of Bry et al. [26] fixes the order so that the number of coefficients is equal to the number of free and fixed derivatives at the boundary conditions. Increasing the order of the polynomial grants the capability to have more complex trajectories when other constraints are introduced. The trade-off with having higher order polynomials is higher complexity in generating a solution because there are more coefficients to optimise.

There is a similar trade-off in complexity by incorporating all dimensions in the same optimisation. The problem becomes large, with many parameters to optimise when the coefficients from each dimension are included in the one optimisation. Combining dimensions, though, enables the use of constraints that mix dimensions, such as obstacles, ellipsoidal performance constraints, free-space corridors, or perception constraints. Without mixing dimensions, the constraints can only be specified independently for each dimension, giving box type constraints. Dimensions can also be separated for ASTRO though, such as planning yaw separately, if there are no constraints that mix yaw and position.

The minimum snap formulations of [63] and [26] assume that there are only the equality constraints as boundary conditions, hence if it is desired to include additional constraints such as obstacles and performance limitations, a more sophisticated optimisation tool is required, as done by Thomas et al. [216]. ASTRO is capable of including obstacles, free-space bounds and performance constraints in the snap optimisation, as outlined in the previous chapter. This enhanced capability brings in greater complexity, though, and longer computation times.

### 5.2.2.2 Quadrotor Planning with Obstacles

The most robust existing algorithms for trajectory planning with obstacles use a combination of a sampling-based planner to get an obstacle-free path, giving the waypoints for a trajectory optimisation algorithm to produce a dynamically smooth trajectory, as explained in Chapter 2.

ASTRO provides the middle ground between a sampling-based planner that can produce paths in geometrically complex obstacle fields and a trajectory optimisation algorithm that creates a dynamically optimal trajectory but does not consider obstacles. The algorithm can produce dynamically optimal trajectories through obstacle fields by using a mix of obstacle representations incorporated into the trajectory optimisation, as described in Section 4.3.3. This capability allows fewer waypoints to be specified, relaxes the requirements for a prior obstacle-free path and grants more freedom to where the trajectory can move to optimise dynamics, e.g. minimising snap.

In contrast, Bry et al. [26] use a trajectory optimiser that does not consider obstacles in the environment and relies on a prior collision-free path. Extra waypoints are added from the collision-free path in segments that are found to be in a collision, and the trajectory re-optimised. This approach can be over-constraining in environments with tight confines, as it forces the trajectory closer to the planned path each time a waypoint is added. The planned path is not dynamically optimal; hence the resulting trajectory will not be as dynamically smooth as otherwise possible. ASTRO takes away the need to add waypoints, allowing smoother trajectories, but at the cost of increased computation time. For further discussion, the approach of Bry et al. [26] is referred to as the UNConstraint Optimiser (UNCO).

The algorithm from of Campos-Macias et al. [32] plans in free-space rather than with consideration of obstacles. Hypercube constraints (bounds separately on each dimension) are used on position, velocity and acceleration at a set of sub-waypoints. The sub-waypoints have specified spacing and segment times to assure the trajectory remains within sets of cylindrical, free-space bounds. Each segment between waypoints is sampled to find the minimum distance to an obstacle for that segment. This minimum distance sets the size of the free-space bounds: $l_{\text {max }}$, that is used to set the physical spacing, $l$, between sub-waypoints along the straight-line paths in $d$ dimensions:

$$
\begin{equation*}
l=\frac{2 l_{\max }}{3 \sqrt{d}} \tag{5.29}
\end{equation*}
$$

This spacing, along with a user set maximum acceleration, $A_{\max }$ are used to define hypercube constraints on position, velocity and acceleration. The bounds of these constraints are defined by $l$ for position, $V_{\max }$ for velocity and $A_{\max }$ for acceleration. $V_{\max }$ is given by:

$$
\begin{equation*}
V_{\max }=\sqrt{l A_{\max }} \tag{5.30}
\end{equation*}
$$

The time for each sub-segment is fixed to $h$, based on the maximum acceleration and waypoint spacing:

$$
\begin{equation*}
h=\sqrt{\frac{4 l}{A_{\max }}} \tag{5.31}
\end{equation*}
$$

These bounds, sub-waypoint spacing and time specification assure the trajectory will remain within $l_{\text {max }}$ from the straight-line paths, hence restricting the trajectory to stay in free space. The resulting
constrained optimisation of a trajectory between the sub-waypoints could be performed in several ways. Campos-Macias et al. [32] optimise polynomial coefficients to minimise acceleration in a constrained quadratic program with both equality and inequality constraints. Alternatively, the formulation of Bry et al. [26] could be used to comply with all fixed boundary conditions, and the inequality constraints included in a convex optimisation with an available solver such as cvxopt [8], to minimise snap. For further comparisons, the approach of Campos-Macias et al. [32] is referred to as the Tube And Cube Optimiser (TACO). TACO gives an assured collision-free trajectory but does so with very conservative cylindrical bounds around each segment.

ASTRO could also be used with representations of free-space by using cylindrical keep-in corridor constraints around each segment, with a radius equal to $l_{\text {max }}$. This representation gives an entirely convex problem to solve, enabling quicker solutions, but with the result of more conservative trajectories. When using the criteria to exit when feasible, as outlined in Section 4.3.6.5, the full environment could be used to check feasibility, instead of the corridor constraints. This method of feasibility checking allows more use of free-space and less conservative trajectories, at the penalty of performing additional collision checks. These two variants of ASTRO are referred to as ASTRO-C when using the corridors to check feasibility, and ASTRO-CE when using the full environment to check feasibility.

Another advantage of ASTRO is that a mix of obstacle representations can be used in the one optimisation, with no change to the formulation. For instance, an ESDF could be used to represent the large-scale static environment, with ellipsoid obstacles added in for newly observed features, or dynamic obstacles.

A comparison of ASTRO and the state-of-the-art techniques will be presented in Section 5.3, with a particular focus on the differences in handling obstacles.

### 5.2.2.3 Performance Constraints

Dynamic-feasibility is already considered, to a large extent, by utilising the differential flatness transformation and minimising snap. This approach gives continuous evolution of the states and controls for the quadrotor. The remain dynamic-feasibility element to check is the control limits. Ideally, the control limits could be checked directly with the RPM of the motors, where the physical restrictions are. Many existing approaches use a check-and-adjust approach to observe the limit. These approaches decrease the trajectory time until the controls exceed a limit, as done with UNCO. Alternatively, The limits on RPM can be mapped back into constraints on velocity, acceleration, jerk and snap to include a performance limitation directly in the optimisation, such as done by Thomas et al. [216]. For example, a limit on acceleration could be used to represent a net thrust limit.

Performance constraints can be included in ASTRO by using keep-in spherical ellipsoid constraints (Eqn 4.43) that operate on velocity, acceleration or higher derivatives. The inclusion of performance constraints becomes another trade-off though, as it adds to the complexity of the search space, making the problem slow to solve. Performance constraints are not used in the problems presented here; they are a capability that could be explored in future work.

There may also be dynamic violations if the rate of change of RPM is too large, in which case more detailed modelling of the rotor dynamics could be used. A simple proxy for this though is to minimise the integral of snap squared, i.e. to minimise the variation in RPM, as is done here.

### 5.3 Quadrotor Trajectory Optimisation - Simulation Comparisons

## Attributions:

The analysis and results presented in this chapter have been previously presented in [154], and are the work of the author of this thesis.

In this section, ASTRO is compared to two algorithms from the literature to assess the relative performance for planning high-speed trajectories for quadrotors near obstacles. The desired characteristics for planned trajectories are:

1. Feasibility: to safely avoid all obstacles.
2. Low computation time.
3. Dynamic-feasibility: to accurately track the trajectory.

An assessment of the first and second criteria is currently presented here with a batch of planning test-cases in representative environments. Flight tests presented in the next chapter assess the third criteria. The batch of test cases run the algorithms over a broad range of scenarios, to assess how reliably the algorithms can generate feasible trajectories, and the time taken to generate those trajectories.

First, the implementation of the algorithms will be presented, followed by details on the generation of tests cases. Results will then be presented and discussed.

### 5.3.1 Algorithm Implementation

UNCO and TACO are compared with three variants of ASTRO. The algorithms are selected to have the equivalent capability with a spread of approaches to optimising trajectories with obstacles. Each of the algorithms plans $x, y, z$, and $\psi$, with $\psi$ set to be along the trajectory. Euclidean Signed Distance Fields (ESDFs) are the obstacle representation used, being a compact and effective way to model all of an environment, and a representation that is suitable for each of the three planners assessed (see Section 4.3.4.4 for details on ESDFs).

UNCO has no consideration of obstacles but is run iteratively, with an extra waypoint inserted from a known obstacle-free path for segments with collisions. The algorithm requires an obstacle-free path as input to be able to plan collision-free trajectories.

TACO also requires an obstacle-free path as input. The implementation of TACO uses optimisation methods from UNCO to solve for free-derivatives, and inherently comply with all fixed boundary conditions. The remaining inequality constraints (for position, velocity and acceleration) are added to the free derivative formulation to be solved in a convex optimisation with the python library cvxopt [8], taking advantage of the fact that all constraints are convex.
ASTRO is implemented with three variants:

1. Using the ESDF as an obstacle constraint, referred to as ASTRO-E.
2. Using free-space corridor constraints, referred to as ASTRO-C.
3. Using free-space corridor constraints with ESDF checks for feasibility, referred to as ASTRO-CE.

To clarify notation, when using the name ASTRO, the overall algorithm is being referred to, regardless of the variation of the constraint type. The full set of optimisation techniques for ASTRO, as described in Section 4.3.6, are used in the tests, including iterative sub-problem solutions starting with an optimisation without obstacles, inflation of constraints, and the criteria to exit when the trajectory is feasible. The approximate-path-integral cost method is used for all three variants. For the non-convex optimisation of ASTRO-E, customised weighting is used, as well as randomised perturbations after three iterations to escape from infeasible local minima. The convex optimisations with ASTRO-C and ASTRO-CE use the quadratic line search.

The trajectory time is optimised with gradient-descent, using the segment times as the decision variables. The cost function to be minimised is a weighted sum of the total time, and the snap cost, as outlined in [26]. The same optimisation approach is used for UNCO and ASTRO and is performed without any obstacles considered. TACO includes time specifications in the formulation hence has no optimisation of time.

### 5.3.2 Test Case Generation

The two algorithms from the literature and the three variants of ASTRO are run on a batch of trajectory planning scenarios in two real indoor environments: a small lab ( $9 \times 13 \times 2.5 \mathrm{~m}^{3}$ ) and a large warehouse $\left(17 \times 22 \times 2.2 \mathrm{~m}^{3}\right) .100$ sets of waypoints are generated in each environment. A given set of waypoints is produced by first generating random start and goal locations, then using RRT* [102], in the Open Motion Planning Library [94] to plan a set of waypoints with collision-free paths between them. Examples of the waypoints used are shown in Fig. 5.8. With a common snap cost function, a single time optimisation is run for UNCO and ASTRO, without any obstacles included. For the full set of waypoints (used by ASTRO with corridor constraints) this optimisation takes on average 4.11 s and 4.39 s for the small and large environments, respectively. A reduced set of waypoints ${ }^{3}$ can be used by UNCO and ASTRO-E, for which the time optimisation takes on average 0.74 s and 1.32 s , for the two environments, respectively. These times are included in the overall computation time results.

### 5.3.3 Results

Table 5.2 presents the results from the batch of test-cases. UNCO is the most successful planner, as the algorithm will eventually converge to the known collision-free path. The algorithm only fails in two cases, where the sampling of the trajectory is not fine enough to pick up a slight collision. These failures, though, are when there are small errors in the ESDF that give a voxel in a collision, surrounded by free voxels that do not properly indicate the correct Euclidean distance to the collision. While TACO is designed to ensure no collisions, it is very susceptible to these ESDF errors when defining the free-space bounds, and hence produces many solutions that are in collision. ASTRO with corridor constraints (ASTRO-C) is similarly susceptible to ESDF errors, leading to the failed test-cases. Using the ESDF to check for feasibility (ASTRO-CE) alleviates many of these failures, resulting in a high feasibility percentage. ASTRO-E can succeed in all but 14 cases, even with the challenge of a non-convex

[^14]

Figure 5.8. Environments used for trajectory planning. Not to scale. (a) small lab environment ( $9 \times 13 \times 2.5 \mathrm{~m}^{3}$ ) with example seed path, (b) large wharehouse environment $\left(17 \times 22 \times 2.2 \mathrm{~m}^{3}\right)$ with two example seed paths. Images are 2D but trajectories include 3 D components.
optimisation. The randomised perturbations help to release the solution from a local minimum in $25 \%$ of the tests, with the failures occurring when a feasible solution can not be found within the time limit.

UNCO is the quickest algorithm, with only unconstrained optimisations to solve. TACO is slower as it is solving a constrained, convex optimisation with many sub-waypoints. TACO, though, is quicker than the convex optimisations of ASTRO-C and ASTRO-CE as TACO requires no time optimisation, and only has constraints active on the sub-waypoints rather than the entire trajectory. ASTRO-CE can be slower than ASTRO-C because of the extra time to perform collisions checks on the ESDF, but can also produce a solution more quickly if early iterations are found to be feasible with the ESDF but not the corridor constraints. ASTRO-C and ASTRO-CE are also slow because of the time optimisation with many waypoints. ASTRO-E has fewer waypoints and as a result has quicker overall computation times, even though the computation time to optimise with constraints can be longer. For example, in the small environment, the optimisation time with constraints takes on average 4.45 s for ASTRO-E compared to 2.67 s for ASTRO-C and 3.03 s for ASTRO-CE. ASTRO-E is slower than UNCO and TACO because it is solving a non-convex optimisation which may require the randomised perturbations to escape from local minima.

The minimum distances highlight how UNCO and ASTRO-E utilise more free-space, resulting in less conservative trajectories than the algorithms with free-space bounds. The ESDF checks in ASTRO-CE enables more use of free-space, resulting in slightly less conservative trajectories than ASTRO-C.

The change from the small to the large environment tends to increase computation times for all algorithms, as there are more waypoints involved. ASTRO-E, in contrast, has a decrease in computation time, as fewer waypoints can be used, and the algorithm can exploit the greater amount of free-space that is available in the large environment.

Table 5.2. Results from Simulation Batch Test

|  |  |  | Values for Feasible Trajectories |  |  |
| :---: | ---: | ---: | :--- | :--- | :--- |
| Algorithm | Env. | \% Feas. | $\underline{d}(\mathrm{~m})$ | $\underline{t}(\mathrm{~s})$ | $t_{m}(\mathrm{~s})$ |
| UNCO | Small | 98 | 0.15 | 0.92 | 1.50 |
| TACO | Small | 72 | 0.17 | 2.77 | 8.06 |
| ASTRO-C | Small | 94 | 0.19 | 6.78 | 20.92 |
| ASTRO-CE | Small | 98 | 0.18 | 7.14 | 36.99 |
| ASTRO-E | Small | 93 | 0.15 | 5.19 | 34.89 |
| UNCO | Large | 100 | 0.10 | 1.51 | 2.08 |
| TACO | Large | 96 | 0.15 | 5.57 | 22.41 |
| ASTRO-C | Large | 86 | 0.13 | 10.78 | 51.91 |
| ASTRO-CE | Large | 97 | 0.12 | 9.50 | 54.47 |
| ASTRO-E | Large | 93 | 0.10 | 4.35 | 35.14 |

$\underline{d}$ is the mean minimum distance, $\underline{t}$ and $t_{m}$ are the mean and min computation times, respectively. ASTRO
variants are: E: ESDF obstacles, C: free-space corridors,
CE: free-space corridors with ESDF feasibility checks.

### 5.3.4 Summary and Assessment of Simulation Comparisons

The simulation results highlight differences in feasibility and computation time. UNCO, an implementation of the work from [26], gives the best overall performance with a high success rate, and low computation time. ASTRO also has a high success rate when using free-space corridor constraints combined with an ESDF to check feasibility. ASTRO produces more conservative trajectories than UNCO, but with longer computation times. TACO: a combination of the work from [32] and [26], also produces conservative trajectories, and with lower computation time, but is susceptible to ESDF errors, as is ASTRO when using only the corridor constraints to check for feasibility. All of the algorithms require an initial collision-free path, except ASTRO with ESDF obstacles, which uses fewer waypoints and utilises more of free space to successfully plan trajectories, despite the challenges of a non-convex optimisation. Randomised perturbations assist ASTRO to solve the non-convex optimisation by jumping a trajectory out of infeasible local minima. One point of comparison that could be explored more in future work is how the performance of the algorithms change over a sliding scale of obstacle density. Where the tests presented here focused on real environments, simulated environments could be used to create specific stages of obstacle density to test the algorithms on.

### 5.4 Conclusion

A critical component for both the planning and control layers for quadrotors is the differential flatness transformation, as described in this chapter. The transformation provides a convenient way to plan dynamically-feasible trajectories for quadrotors and to combine position and attitude controllers. Of concern are the singularities in the transformation and sensitivities near those singularities. An analysis of existing and proposed methods to handle these sensitivities highlighted the scenarios in which the methods have issues, and found that a combined method, proposed here, gives the most robust performance. This transformation is demonstrated in aggressive flight in the next chapter.

This chapter also presented the adaptations of ASTRO to apply to quadrotors, by utilising the differential flatness transformation and optimising snap for $x, y$ and $z$ to produce dynamically-feasible trajectories. Compared to the state-of-the-art for planning quadrotor trajectories near obstacles, ASTRO provides the ability to plan collision-free trajectories without a prior collision-free path. Additionally, with ASTRO, obstructions in the environment can be represented by either free-space corridor constraints or as obstacles. When using free-space corridors, ASTRO has a high success rate, but with conservative trajectories. When using obstacles, more free-space can be used, with fewer waypoints than other algorithms, but the non-convex optimisation can lead to longer computation times. Using obstacles, though, means that there is not a reliance on a known collision-free path. The next chapter assesses the benefits of ASTRO in providing dynamically-feasible trajectories, by combining planning and control layers to test the accuracy of trajectory-tracking in flight.

## UAV FLIGHT DEMONSTRATIONS



## Attributions:

The hardware system described in this chapter comes from [192] and is described in detail here to give context to the flight tests. The author of this thesis contributed to [192] in writing the descriptions of the system, developing components of the ground control station, developing trajectory optimisation algorithms, performing tests and supporting hardware development and modifications. These contributions will be mentioned in the chapter.

Flight test results have been presented in [153] for the testing of the differential flatness transformation, and [154] for the comparison of trajectory optimisation algorithms. These publications are primarily the work of the author of this thesis, with co-authors contributing to carrying out tests, and discussion of results. The comparison of trajectory optimisation algorithms presented here includes additional results that were not presented in [154].

The true test of the planning and control layers in the autonomous navigation stack is when they are operating on a real hardware system. Such demonstrations provide validation that the algorithms can achieve the desired purpose, as well as stress testing the algorithms by introducing noise, disturbances and model uncertainties. This chapter presents results from test flights with a quadrotor to assess the effectiveness of differential flatness transformations in aggressive flight
and the dynamic-feasibility of trajectory optimisation algorithms. The goal is to validate the proposed algorithms described in previous chapters and highlight additional factors to consider when operating real hardware.

The chapter starts with a description of the hardware system used for the tests, before presenting tests of the differential flatness transformation, followed by the results from the assessment of the dynamic-feasibility of trajectories.

### 6.1 Description of Hardware System

Attribution: Unless otherwise noted, the design and development of the hardware systems is work from the authors of [192]. The descriptions of the system are the work of the author of this thesis.

The hardware system on which testing is performed is a quadrotor developed by the NASA Jet Propulsion Laboratory with support from Google and is designed to enable high-speed flight amongst obstacles in a known environment with visual localisation. The system is demonstrated in this video: https: //youtu.be/SrqrGweKQAU. For the context of the tests performed, the system is described here.

The concept of operations for the system is outlined in Fig. 6.1. The first step of operations is to map the environment, collecting data from visual and depth cameras that is processed off-line to generate several representations of the environment. These representations include an Area Descriptor File (ADF) for localisation, a Euclidean Signed Distance Field (ESDF) for obstacle representation, and a 3 D mesh for visualisation. With the ADF uploaded onto the quadrotor, localisation algorithms can be run to give accurate state information on the quadrotor. This state information allows the position of the quadrotor to be recorded as it is piloted or walked around the desired path, producing a dense set of waypoints. These waypoints are simplified to a smaller set on the Ground Control Station (GCS) and are then used to optimise a trajectory. To ensure these trajectories are collision-free the ESDF is used to represent obstacles in the planning algorithms. An operator can view, check, edit and replan these trajectories on the 3D graphical user interface on the GCS. The GCS also allows the ESDF to be edited and provides a 3D mesh visualisation of the environment. Once the operator is satisfied with the trajectory, it is sent to the quadrotor to be flown, with feedback on the position being sent back for the GCS to display.

A high-level overview of the system will first be presented, before explaining each subsystem in more detail.

### 6.1.1 High-Level Architecture

Fig. 6.2 outlines the key components of the system. The processors and sensors onboard the quadrotor perform the tasks of localisation, estimation, control and interfacing with the GCS. Trajectories are planned on the GCS and sent to the quadrotor over WiFi with ROS messages, a link that is also used to send back state information from the localisation module for the GCS to display. The main components outlined in Fig. 6.2 will be described below.


Figure 6.1. Concept of operations. Purple boxes are processes run with the quadrotor, green components are data collection, blue components are map processing and the white box is trajectory optimisation. Arrows indicate sequence of steps and the data sent between processes.


Figure 6.2. High-Level Architecture. Green boxes are inputs and outputs, blue boxes and arrows are related to mapping processes, yellow boxes are communications and the purple is the software and communications link between the two main on-board processors. All components other than the Ground Control Station are on-board the quadrotor. Arrows indicate the data sent between processes.

### 6.1.2 Airframe

The quadrotor (Fig. 6.3) is based on a 250 size frame ( 250 mm from the centre of the front-left rotor to the centre of the back-right rotor) with a carbon base, in an X configuration. A custom, 3D printed frame provides the housing for the electronics, with a vibration-isolated carriage for the processors, IMUs and cameras. The cameras consist of one down-facing fish-eye camera and one front-facing fish-eye camera. This airframe design enables the quadrotor to be very compact, while still having the required capability.


Figure 6.3. Quadrotor used for flight tests (without battery). Image from [192].

### 6.1.3 On-Board Computing

The core component of the online computing architecture, as depicted in Fig. 6.2, is the Qualcomm Snapdragon flight board [185]. The Snapdragon incorporates two different processor units: 1) the CPU quad-core Krait ${ }^{1}$, running at 200 Hz and 2) the Digital Signal Processor (DSP), a Hexagon core ${ }^{2}$ running at 1 kHz .

The Krait core runs a Linux Linaro distribution, with Linux Kernel 3.4, and serves as the primary interface with the quadrotor, providing connection to the GCS over a WiFi link and communicating with the Tango localisation module. The PX4 [138] flight software is run on both the Krait core and the DSP, with all the real-time components running on the DSP. The DSP operates at 1 kHz and is designed to support hard-real-time functions, specifically the position and attitude controllers, thrust mixers, attitude estimator, and set-point evaluation (Poly. Eval. in Fig. 6.2).

A ROS bridge runs on the Krait core to exchange messages with the GCS over WiFi. Planned trajectories are sent to the quadrotor from the GCS, and state information, consisting of position, orientation and set-points are sent to the GCS from the quadrotor.

The localisation module, Tango ${ }^{3}$, has a separate, dedicated processor, which communicates with the Krait processor, sending the position, velocity and attitude estimates. The two fish-eye cameras, one front facing and one down facing, are connected directly to the Tango board, which, along with an

[^15]integrated Inertial Measurement Unit (IMU) are used for localisation. The attitude estimator running on the DSP utilises a different IMU that is integrated on the Snapdragon.

### 6.1.4 Actuation

Four Electronic Speed Controllers (ESCs) control the four motors, and each can get feedback to measure the RPM. This feedback is used to close the loop around the RPM control, to track the commanded RPM more accurately. For the demonstrations presented here, 2300 kV Luminier brushless motors were used with DL45 bull-nose tri-blade props. See [192] for more details.

### 6.1.5 Control

> Attribution: The author of this thesis contributed to controller development in the implementation of SI units and assisting implementation of aerodynamic drag compensation.

The control software builds from the PX4 flight software with modifications to the controller for highspeed tracking of trajectories. Modifications are also made to incorporate SI units for more intuitive gain tuning. The control architecture is outlined in Fig. 6.2. A polynomial evaluator computes set-points from a trajectory to send to a position controller. The position controller computes an attitude set-point for, which is used by an attitude controller to produce torque and force commands. These commands are sent to a mixer to compute RPM commands for the ESCs.

The polynomial evaluator uses the piecewise polynomial trajectory generated by the planning algorithms and frequently evaluates the polynomials at increasing points in time when the quadrotor is commanded to follow the trajectory. The evaluated points give position, velocity, acceleration and yaw set-points for the position controller to track. An important distinction here is that set-points are generated and updated on the DSP at the same rate as the position controller, allowing close following of aggressive trajectories.

The position and attitude controllers are modified from [118], to include aerodynamic drag compensation and a simpler attitude controller.

### 6.1.5.1 Position Control

Using the set-points and current state estimate from Tango, the position controller has feedback gains on position error and velocity error, along with a feed-forward gain on the desired acceleration, giving a PDFF (Proportional, Derivative, Feed-Forward) controller. There is additionally feed-forward compensation for the parasitic and propeller drag:

$$
\begin{equation*}
\mathbf{T}_{s p}=-K_{p}\left(\boldsymbol{x}-\boldsymbol{x}_{s p}\right)-K_{d}\left(\dot{\boldsymbol{x}}-\dot{\boldsymbol{x}}_{s p}\right)-m \tilde{g} \boldsymbol{z}_{g}+m K_{f f} \ddot{\boldsymbol{x}}_{s p}+\mathbf{R} \mathbf{D}_{p}\left(\dot{\boldsymbol{x}}^{2}\right)+\mathbf{R} \mathbf{D}_{h}(T, \dot{x}) \tag{6.1}
\end{equation*}
$$

where all vectors are in the global frame, the $s p$ subscript denotes the set-points, with other terms being the state estimates. The $K$ terms are the gains applied to the system, and $m$ is the drone mass. $\boldsymbol{z}_{g}$ is the global $z$-axis that is used to give compensation for the acceleration due to gravity, $\tilde{g}$. The current
attitude is represented in the rotation matrix $\mathbf{R}$, which is used to transform drag terms from the body frame to the world frame. $\mathbf{D}_{p}$ is the parasitic drag vector, which is a function of the velocity squared and $\mathbf{D}_{h}$ is the propeller drag vector, which is a function of rotor thrust and body velocity. The output from the controller is the desired thrust vector, $\mathbf{T}_{s p}$. The magnitude of this vector is the commanded thrust, which is sent through to the thrust mixer.

The direction of the thrust vector combines with a yaw set-point to go through the differential flatness transformation (as described in Section 5.1) to produce a quaternion set-point.

### 6.1.5.2 Attitude Control

The attitude controller uses the state information from the attitude estimator and the quaternion set-point to compute the desired torques. The quaternions are converted to rotation matrices (see Appendix C) to compute the angular error:

$$
\begin{equation*}
\mathbf{e}_{q}=\frac{1}{2}\left(\mathbf{R}_{s p}^{T} \mathbf{R}-\mathbf{R}^{T} \mathbf{R}_{s p}\right)^{V} \tag{6.2}
\end{equation*}
$$

where $\mathbf{R}_{s p}$ is the desired attitude and $\mathbf{R}$ is the current attitude. The vee operator, ${ }^{V}$ maps a skewsymmetric matrix to a vector as described in Eqn. 4.52.

The angular rate error requires the desired angular rate, $\omega_{s p}$, which needs to be computed from the derivative of the quaternion set-point from the past two set-point commands. Details of this derivative computation are in Appendix C.9. The rate error is then computed with:

$$
\begin{equation*}
\mathbf{e}_{\omega}=\omega-\mathbf{R}^{T} \mathbf{R}_{s p} \omega_{s p} \tag{6.3}
\end{equation*}
$$

The attitude and angular rate errors errors are then used in a PD controller ${ }^{4}$ :

$$
\begin{equation*}
\boldsymbol{\tau}=-K_{q} \mathbf{e}_{q}-K_{\omega} \mathbf{e}_{\omega} \tag{6.4}
\end{equation*}
$$

The output, $\boldsymbol{\tau}$ is the desired torques (about the $x, y$ and $z$ axes), which is sent to the mixer, along with the desired thrust magnitude.

The desired moments and thrust are then sent to a mixer, where iterative thrust mixing solves for the thrust coefficients, which are a function of velocity and RPM, to get the motor RPMs.

### 6.1.6 Localisation

Tango is the package developed by Google for visual localisation, which consists of a code stack, and a dedicated processor, complete with on-board IMU [129]. Tango extracts visual features from the environment and matches them to an Area Descriptor File (ADF), that stores a map of the same visual features and is generated in the pre-mapping phase. The feature matching, along with the IMU provides the framework for a localisation filter to update the estimated position and orientation of the quadrotor. If observed features do not match to the ADF , then Tango can run in visual odometry mode,

[^16]simply tacking features frame to frame. A re-localisation algorithm is also running in this case, as a loop closure algorithm to correct drift when sufficient numbers of observed features are matched to the ADF. Full details of the algorithms are proprietary, but a similar type of system of visual navigation is ORB-SLAM2 [163].

For this section, the Tango system can be regarded as providing highly robust and accurate localisation. Refer to [192] for an analysis of the localisation performance of the system.

### 6.1.7 Mapping

A key part of the operation of the system is the pre-mapping stage, where a Tango phone with a depth sensor ${ }^{5}$ is walked around the environment, requiring the camera to be 1-5 metres from features in the environment. Data from the colour camera, depth camera and IMU are recorded through this sequence for off-line processing.

The recorded dataset is processed as a large bundle adjustment, including searching for loop closures with a bag-of-words approach. The output from the computations are:

1. An Area Descriptor File (ADF)
2. A 3D textured mesh
3. A Truncated Signed Distance Field (TSDF)

The ADF is a collection of the visual landmarks in the environment and is the map loaded onto the quadrotor to run map-based localisation. The 3D mesh is primarily for visualisation in the GCS for user awareness of the environment. An example of the mesh is shown in Fig. 4.5 and Fig. 6.5. A Signed Distance Field (SDF) is a gridded representation of an environment where each cell gives the signed distance to the nearest obstacle, with negative distances being inside an obstacle. In a Truncated SDF, cells are only defined in a small region around the surface of obstacles, to improve the efficiency of representation and update.

The TSDF is converted to a Euclidean Signed Distance Field (ESDF) to represent the obstacles throughout the environment for trajectory optimisation. The open source Voxblox library, with python bindings, is used for generating and modifying both TSDFs and ESDFs [171]. Manual modifications are made to the ESDF in the GCS as required, to clear free-space or add occupied space, such as a wall to block a trajectory from going into a particular area.

3D point-based queries of the ESDF give the signed distance and the distance gradient to be used in trajectory optimisation and for visualisation.

[^17]
### 6.1.8 Planning

> Attribution: The waypoint generation, and implementation of a base version of a trajectory optimisation algorithm are contributions by co-authors from [192]. The author of this thesis extended from the base trajectory optimisation algorithm to include consideration of obstacles, and also implemented two more obstacle-aware algorithms.

The planning for the quadrotor takes from the state-of-the-art described in Chapter 5 . The waypoints, which represent the desired path to be flown, are first generated and adjusted. Then, an optimal trajectory is planned through the waypoints, taking the obstacles into considerations.

### 6.1.8.1 Waypoint Generation

The specification of waypoints can either be done manually, through the graphical interface, or through a teach-and-repeat process. To "teach" the quadrotor, the localisation module is turned on, and the quadrotor is flown (or hand carried) around the desired course. The location of the quadrotor is recorded at discrete intervals (for instance every 10 cm ) to be saved as waypoints. The resulting dense set of waypoints provides a collision-free path through the desired locations.

For trajectory optimisation, it is desired to have a minimal number of waypoints, to enable smoother trajectories and reduce computation time. Therefore, the number of waypoints is reduced using the Ramer-Douglas-Peucker algorithm (RDP) [90]. RDP selects points to remove that cause the least variation to the path. A user-defined setting, $\epsilon$, controls how aggressively to remove points (higher $\epsilon$ will remove more waypoints). The resulting reduced set of waypoints maintains the highest curvature points, i.e. the corners of the trajectory. An example of RDP with varying values of epsilon is shown in Fig. 6.4

Once reduced, the waypoints can be manually manipulated to achieve the desired path using the GCS, including moving, adding, and deleting waypoints. Varying $\epsilon$ and manually modifying the waypoints gives the flexibility to provide a suitable starting point for the trajectory optimisation algorithms.

### 6.1.8.2 Planning Algorithms

Three different planners are implemented in the system: UNCO, an implementation of the work of Bry et al. [26], TACO, a combination of the work of Campos-Macias [32] and Bry et al., and ASTRO with ESDF obstacles, choosing the ASTRO variant that provides the greatest contrast in approaches to considering obstacles. Chapter 5 describes these planners. The input to each of the planners is the set of waypoints, with different $\epsilon$ settings used for each planner in the RDP algorithm. Each algorithm sets the yaw at the waypoints to follow the trajectory. To achieve this, the position estimation is run first, before computing the desired yaw at each waypoint to then running the yaw optimisation independently. All trajectory optimisation is run offline on the GCS.

Piecewise polynomials represent the trajectories produced by each of the planners. The polynomial coefficients and breakpoints of the trajectory are placed in a custom ROS message to be sent over WiFi


FIGURE 6.4. Example of RDP reducing the number of waypoints (small quadrotor figures) with varying values of the setting $\epsilon$. (a) $\epsilon=0.1$, (b) $\epsilon=0.3$, (b) $\epsilon=0.5$, (b) $\epsilon=0.7$. The path started with dense waypoints along the straight-line segments in (a).
to the quadrotor. The polynomial evaluator on the quadrotor then uses the coefficients and break-points to compute the set-point at a given time.

### 6.1.9 Ground Control Station

Attribution: A base version of the Ground Control Station was created by co-authors from [192]. The author of this thesis extended from this work to add functionality for manipulating trajectories and giving feedback to the operator.

The Ground Control Station (GCS) is the human interface to control the operations of the quadrotor. It is Python-based using QT4 and RViz [100] ${ }^{6}$. Communications between the GCS and the quadrotor use WiFi, over which: a) the scripts running on the quadrotor are initialised and b) communications occur between the ROS node on the quadrotor and ROS nodes on the GCS computer. A handheld radio-controlled transmitter provides a link for the safety pilot, with a kill switch and manual override capabilities.

The primary interface of the GCS is a 3D Graphical User Interface (GUI), the primary component of which is a 3D mesh of the environment, generated in the pre-mapping phase of operations (see Fig. 6.5). The state information on the quadrotor that is sent through ROS messages to the GCS is displayed on

[^18]the mesh, allowing the operator to check localisation and track the progress of the quadrotor, such as during the waypoint collection stage of operations and when flying trajectories.

To generate, modify and send trajectories, the GCS has a set of widgets, which work with the GUI to display the trajectory with the mesh (see Fig. 6.5). Referring to the workflow outlined in Fig. 6.1, an RDP widget is used to load waypoints, save waypoints, and run the RDP reduction. This widget is primarily used to load the waypoints from the waypoint collection stage and to reduce it to a minimal set of waypoints (as decided by the operator). A planner widget then provides the capability to optimise and adjust a trajectory between waypoints. The widget's capabilities include moving, adding and deleting waypoints with the 3D interface, optimising with different time penalties, adding obstacles, saving and loading trajectories, creating laps, adding take-off and landing, and sending the trajectory. The sent trajectories are packaged with polynomial coefficients into a ROS message to be processed by the polynomial evaluation script on the quadrotor. Through this process, the GCS provides the operator with the ability to assess the feasibility of a trajectory; hence there are numerous forms of information on the trajectory. This information includes text fields showing the minimum distance to obstacles, maximum acceleration, and trajectory completion time, along with graphical feedback on the trajectory showing acceleration arrows and colouring for obstacle clearance (see Fig. 6.5).


Figure 6.5. Ground Control Station with operator controls, 3D graphics and coloured trajectory information. The coloured mesh is produced in the pre-mapping stage of operations, which also produces an ESDF. The trajectory is planned by UNCO and was flown in flight tests.

A third widget, building on the Voxblox python bindings [171], enables adjustments to the ESDF, and visualisation at different vertical slices (see Fig. 4.5). These capabilities are used to clear out free-space around the set of generated waypoints (known to be collision-free), and to add planes of occupied space to areas not captured in the original map (such as ceilings).

While the trajectory planning looks to generate feasible and safe trajectories automatically, the GCS
is an important component to efficiently integrate a human into the loop to control the high-level goals and have extra checks to ensure safe operation.

### 6.2 Differential Flatness Testing - Aggressive Flights

Attribution: The results presented in this section come from [153], and are the work of the author of this thesis.

The analysis presented in Section 5.1 looked to assess methods of performing the differential flatness transform to enable successful operation across all orientations. This capability enables aggressive trajectories that pass through $90^{\circ}$ of pitch or roll. A series of flights tests have been performed to show the need for such a robust transformation. These tests were designed to show examples where the singularities and sensitivities in the transformations are met. The flight tests assess how the transformation performs in the controller, as this is the critical application when the quadrotor is flying. Therefore, the tests presented here modify the transformation in the controller from the PX4 transformation to the Four Axes Combined method or the Six Axes combined method. Trajectories are planned with UNCO between three waypoints, such as in Fig. 6.6, to provide an appropriate test for the controller.

A software-in-the-loop simulation will first be presented, followed by flight tests with the quadrotor system described in this chapter. Videos of the flight experiments described can be found here: https: //youtu.be/M-1jA1KCqb8.


Figure 6.6. Planned aggressive trajectory between three waypoints. Arrows represent direction and relative magnitude of commanded acceleration

### 6.2.1 Software-in-the-Loop Tests

A sharp pitching manoeuvre is tested in a software-in-the-loop simulator, RotorS [72], running PX4 with the 3DR Iris quadrotor model from 3DR. The default PX4 transformation is compared to the new combined method. The PX4 method fails near to $90^{\circ}$ pitch, with a $180^{\circ}$ change in the direction of the $x$-axis, shown in the discrete change in the quaternion in Fig. 6.7. While the drone does not crash, the behaviour is highly undesirable when striving for close tracking of aggressive trajectories. The Six Axes Combined method, in contrast, does maintain continuous set-points throughout (Fig. 6.7.b).


Figure 6.7. Software in the loop simulation results for pitching trajectory. (a) using the standard PX4 controller, showing discontinuous jumps. (b) using the Six Axes Combined method. For both (a) and (b): top is the thrust set-point and the bottom is the output attitude set-point coming out of the differential flatness transformation in the controller (yaw is constant).

### 6.2.2 Flight Tests

The trajectory that is shown in Fig 6.6 was flown with three different transformation methods: a) the standard PX4 method, b) the Four Axes Combined method and c) the Six Axes Combined method.

The results from these flight experiments are shown in Fig. 6.8, including the input to the differential flatness transformation: the thrust set-point (in addition to a constant desired yaw of $90^{\circ}$ ), and the output: the desired attitude as a quaternion set-point. Discontinuities are visible in the quaternion set-points for both the PX4 and Four Axes Combined methods, which is not ideal when accurate tracking is desired.

The discontinuous jumps for the Four Axes Combined method is a result that was only observed in flight. The planned trajectory worked smoothly for that method, as shown in Fig. 6.9. Flying the trajectory introduces more variables, with tracking errors and disturbances changing the desired thrust that is output from the position controller. An important point here is that even though the problem
(a)

(b)




FIgURE 6.8. Aggressive trajectory flight results for three differential flatness transformations: (a) standard PX4 method, (b) Four Axes Combined method and (c)Six Axes Combined method. In each of (a), (b), and (c): top: thrust direction set-point, bottom: quaternion set-point. These values are the input and output, respectively, of the differential flat transformation.
areas for a transformation method might not be expected to be encountered, disturbances from the nominal trajectory could push the quadrotor into such areas.


Figure 6.9. Planned aggressive trajectory acceleration and corresponding attitude set points as quaternions. (a) Acceleration for the trajectory, (b) quaternion from the Four Axes Combined method, showing smooth changes in quaternions, in contrast to Fig. 6.8.b.

A limitation that is evident in the flight results is that it is possible for the yaw to move to 180 degrees of error. For example, at orientations of $90^{\circ}$ roll or pitch, a yaw angle of $0^{\circ}$ and $180^{\circ}$ become equivalent. If the $180^{\circ}$ pathway is taken, it will continue to be tracked because the methods are checking against the last orientation (which is now at $180^{\circ}$ yaw). Nonetheless, the Six Axes Combined method enables highly aggressive trajectories to be tracked, such as demonstrated in Fig. 6.10, in a pitching manoeuvre beyond $90^{\circ}$ where the thrust vector passes below the $x y$ plane (when the $z$ component of the thrust vector goes below zero). Running the standard PX4 transformation on the same sequence of thrust set-points produces multiple discrete jumps in orientation (Fig. 6.10.c).

These flight test results show that the types of orientations discussed in Section 5.1.4 that cause differential flatness transformations to fail can indeed occur in flight when attempting aggressive manoeuvres. Hence it is important to employ an adequate transformation method for applications striving for aggressive flight.


Figure 6.10. Flight results from highly aggressive trajectory pitching beyond $90^{\circ}$. (a) Thrust direction for the trajectory, showing a z component of thrust below zero (inverted). (b) Quaternion set-point from controller in flight, using Six Axes Combined method. (c) Quaternion from PX4 method applied to the flight data.

### 6.2.3 Conclusions - Differential Flatness

The differential flatness transformation is a crucial part of trajectory planning and control of quadrotors. With a push towards highly aggressive flight, progressing through $90^{\circ}$ pitch or roll, and inverted flight, the commonly used transformation becomes susceptible to the singularity when the desired thrust vector aligns with the desired heading in the $x y$ plane. Numerous methods have been proposed to deal with the singularity, but each has limitations arising from sensitivities around the singularity, and in transitions near it; these can cause more issues than the singularity itself.

An analysis of existing and newly proposed transformation methods highlights the limitations that are present. A new method that checks six body-axes-sets against the previously computed axes is shown to be the most stable method through all orientations. A susceptibility exists, however, for the yaw to have an error of $180^{\circ}$ after completing aggressive manoeuvres. Different methods may be more suitable for a given application; hence the analysis presented here may serve as a resource for others to understand the characteristics and limitations of the various methods. Simulation and flight tests with aggressive trajectories show that the problematic flight conditions for the transformations can be experienced and do cause issues, demonstrating a need to carefully consider the differential flatness transformation method when designing a quadrotor system for highly-aggressive flight.

### 6.3 Comparison of Planners

Attribution: These results come from [154], and are the work of the author of this thesis.

ASTRO, implemented on the quadrotor system is compared against two of the state-of-the-art algorithms in trajectory planning with obstacles, as described in section 6.1.8.2: UNCO, and TACO.
Repeating from section 5.3, the key goals for the trajectories produced are:

1. Feasibility: to safely avoid all obstacles
2. Low computation time
3. Dynamic-feasibility: to accurately track the trajectory

The batch simulations in Section 5.3 assessed the first and second criteria. Analysis of flight tests assess the third criteria: dynamic-feasibility, that refers to trajectory tracking in flight: an important consideration when flying near obstacles, where deviation from the trajectory could result in a collision. The flight tests provide a real-world assessment of dynamic-feasibility by looking at tracking error for flights within dense obstacle fields.

Flight tests were performed in a medium lab environment of $4 \times 20 \times 3 \mathrm{~m}^{3}$ (shown in Fig. 6.11). Dense waypoints are generated by recording positions of the quadrotor in a hand-walked trajectory, after which RDP is used to reduce the number of waypoints for the trajectory optimisation algorithms, with the original set satisfying the dense waypoints needed by UNCO.


Figure 6.11. Medium lab environment with numerous obstacles where the test flights were conducted. (a) 3D view of the environment. (b) Top-down view of environment with example trajectory produced by ASTRO that was flown.

Each of the algorithms assessed includes an optimisation to minimise for snap; however, there are further tuning parameters that can be adjusted. The main tuning parameter for UNCO and ASTRO is a weighting, $W_{t}$, on the time-cost in the outer-loop optimisation for total trajectory time (Eq. 4.97). Larger $W_{t}$ leads to slower, more conservative trajectories. The tuning factor for TACO is the maximum acceleration value, $A_{\max }$, which sets the velocity constraints (Eq. 5.30) and time between waypoints (Eq. 5.31). Larger $A_{\max }$ increases the aggression of the trajectories. In the following analysis, efforts were made to have comparable overall trajectory times, and the same $W_{t}$ was used for ASTRO and

TACO. ASTRO also has several other tuning factors, as explored in depth in Section 4.3.6, with the most influential factors being associated with obstacle cost functions. Finally, the number of waypoints used is a factor that can be adjusted. TACO is limited to starting with obstacle-free paths. UNCO can start with fewer waypoints, and will add waypoints until the trajectory is obstacle free. ASTRO requires the fewest number of waypoints as long as a feasible trajectory can be found. In the tests here, ASTRO and UNCO started with the same number of waypoints.

The set of waypoints, localisation system and tracking controller are all consistent between the algorithms; hence the variable being analysed is the dynamic-feasibility of the trajectories: a characteristic that is evident in the tracking performance.

Trajectories were planned over a range of total trajectory times (time to fly the trajectory) to observe the trends in performance, and compare tracking performance between algorithms at a range of equivalent trajectory times. This range of flights included speeds up to $5.5 \mathrm{~m} / \mathrm{s}$, and accelerations up to $5.6 \mathrm{~m} / \mathrm{s}^{2}$. Flights were also repeated multiple times at a trajectory time of 35 s and 25 s to average out noise from different flights. A video showing the range of flights can be found at https: //youtu.be/oQoOJ69-Dgk. The results from these tests are presented and analysed below.

### 6.3.1 Obstacle-Aware Flight Tests

A plot of the planned and flown trajectories at 35 s trajectory time is shown in Fig. 6.12. This figure highlights the differences in the planned trajectories. TACO can move sub-waypoints within the hypercube bounds; hence the algorithm can reduce the radius of certain corners and loops, but it requires a tighter turn on the lower right of the trajectory to ensure the conservative bounds are met. ASTRO and UNCO are quite similar, except for at the upper right where UNCO produces a very tight radius turn, in contrast to ASTRO which gives a smoother turn of larger radius.

The change in tracking performance with respect to trajectory time is shown in Fig. 6.13, where the expected trend of larger error at higher speeds (lower trajectory time) can be observed. In comparing the different planners, the smoother nature of the obstacle aware path from ASTRO leads to superior tracking across the range of trajectory times. ASTRO consistently gives better RMS tracking errors than UNCO by a small margin. TACO fails to solve at higher speeds, and hence is not able to produce trajectories at lower trajectory times. With increasing speeds, the times allocated to segments between sub-waypoints gets very small, which causes the constraints from the boundary conditions to become almost identical, making the problem ill-defined. TACO has superior position and velocity tracking but requires large yaw movements which leads to more substantial attitude tracking error.

A summary of the tracking performance for the 35 s flights is shown in Table 6.1. Although TACO gives better RMS errors for position and velocity, the maximum errors are greater. The trajectories generated by TACO tend to have inconsistencies in tracking, with more substantial standard deviations in maximum thrust. ASTRO marginally out-performs UNCO in all categories, including smaller standard deviation in the position error, suggesting the smoother trajectories generated by ASTRO are easier to track more consistently.


FIGURE 6.12. Planned and executed trajectories for each algorithm with consideration of obstacles, at a trajectory time of 35 s ( 37 s for TACO). A 2D view is presented, but the trajectory has 3D components. An overlay of the trajectory on the environment for ASTRO is in Fig. 6.11.b, and for UNCO in Fig. 6.5.

Table 6.1. Tracking errors for flights at 35 s

| Errors | UNCO |  | TACO |  | ASTRO |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Mean | StDev | Mean | StDev | Mean | StDev |
| $\boldsymbol{x}_{\text {RMS }}(\mathrm{m})$ | 0.084 | 0.021 | 0.067 | 0.005 | 0.072 | 0.006 |
| $\boldsymbol{x}_{\text {max }}(\mathrm{m})$ | 0.268 | 0.204 | 0.395 | 0.327 | 0.189 | 0.041 |
| $\dot{x}_{\text {RMS }}(\mathrm{m} / \mathrm{s})$ | . 097 | 0.019 | 0.077 | 0.006 | 0.089 | 0.006 |
| $\dot{\boldsymbol{x}}_{\text {max }}(\mathrm{m} / \mathrm{s})$ | 0.404 | 0.133 | 0.470 | 0.236 | 0.362 | 0.080 |
| $\mathbf{q}_{\text {RMS }}(\mathrm{rad})$ | 0.070 | 0.004 | 0.087 | 0.003 | 0.067 | 0.003 |
| $\mathbf{q}_{\text {max }}(\mathrm{rad} / \mathrm{s})$ | 0.262 | 0.045 | 0.598 | 0.325 | 0.255 | 0.071 |
| $\omega_{\text {RMS }}(\mathrm{rad} / \mathrm{s})$ | 1.034 | 0.016 | 1.378 | 0.012 | 0.934 | 0.011 |
| $\omega_{\text {max }}(\mathrm{rad} / \mathrm{s})$ | 2.674 | 0.118 | 7.080 | 0.058 | 2.129 | 0.4 |

### 6.3.1.1 Higher-Speed Flights

Flights were also repeated at 25 s from UNCO and ASTRO (TACO could not generate such a trajectory). These results emphasise the differences between UNCO and ASTRO (see Fig. 6.14). The top right of the plot clearly shows ASTRO taking a smoother trajectory, as it can have a sparse number of waypoints, but still avoid obstacles, whereas UNCO has added waypoints to avoid collisions, but also constrain the trajectory to have a tight turning radius.


Figure 6.13. Tracking Root-Mean-Square (RMS) errors across a range of flights of increasing speed for each algorithm. RMS for position ( $x$ ), velocity ( $\dot{x}$ ), attitude ( $q$, analysed as a single angular error), and angular velocity ( $\omega$ ). Each flight represents one data point with a quadratic fit drawn between them.

### 6.3.2 Conclusions - Comparison of Planners

The flight results highlight the impact of the approaches taken by each algorithm in considering obstacles. By including obstacles in the optimisation, ASTRO with ESDF obstacles can exploit freespace to generate collision-free trajectories that are easier to track, allowing greater freedom to take smooth trajectories. UNCO also produces very smooth and trackable trajectories. However, if the reference path has tight turns within tight confines, then UNCO will add extra waypoints from the reference path. The extra waypoints can result in tighter turns and a trajectory that is more difficult to track. TACO can generate smooth and safe trajectories effectively and can give a lower distance path with the ability to adjust waypoints, but the solution method has limitations with the speed that TACO can plan a trajectory.

Combining with the simulation results in Section 5.3, the best planner depends on the given application. If conservative, slow, and safe trajectories are required, then TACO or ASTRO with corridor constraints may be the best choice. If higher speed trajectories are desired, and the input route is relatively smooth, then UNCO would be the best option. If instead, the input route has tight turns in tight constraints, then ASTRO would give the better quality trajectories. For a general case, UNCO has the best trade-off between computation time, collision-feasibility and dynamic-feasibility, if it is


Figure 6.14. Planned and executed trajectories for UNCO and ASTRO at a trajectory time of 25 s . A 2 D view is presented, but the trajectory has 3 D components.
possible to access an initial, collision-free path. If such a path is not available, then ASTRO with ESDF obstacles can be used.

These results not only provide an assessment of the three algorithms considered but give insight into the components to consider when selecting a trajectory optimisation algorithm, in particular, the trajectory tracking performance during flight.

### 6.4 Conclusions

The autonomous quadrotor system described in this chapter is capable of accurate trajectory tracking at high speeds near obstacles, using a robust visual localisation module, advanced tracking controllers and a sophisticated ground control station. The capability of this system enables the assessment of controllers and trajectory planning algorithms in flight.

The flight tests presented here show that the sensitivities in the differential flatness transformation can be experienced in aggressive flight, even if not expected, and do cause issues. The proposed Six Axes Combined method was shown to handle challenging manoeuvres robustly, in scenarios where existing transformations failed. There is a remaining susceptibility, though, to a $180^{\circ}$ yaw error after completing an aggressive manoeuvre.

The accurate localisation and trajectory tracking of the quadrotor system enables assessment of the dynamic-feasibility of planned trajectories for high-speed flight near to obstacles. The quadrotor system was used to perform a comparative analysis between different methods of considering obstacles in trajectory optimisation. By including obstacles directly in the optimisation, the algorithm described in this thesis, ASTRO, can exploit free-space to generate collision-free trajectories that were shown to be easier to track than the state-of-the-art. Flight tests also validated ASTRO as being able to produce suitable trajectories for high-speed flight of quadrotors near obstacles.

This validation of ASTRO for the planning layer of the autonomous navigation stack leads to the following chapter, where ASTRO is combined with NURBSLAM to test a system consisting of localisation, mapping and planning layers.


## INTEGRATED SYSTEM

| Sensors |
| :---: |
| Image Processing |
| Odometry |
| Localisation |
| Mapping |
| Planning |
| Control |

Attributions: The robotic simulation framework presented in this chapter is work from an equal collaboration between the author of this thesis, Mauricio Coen and Anne Bettens.

T
he previous chapters have presented different layers of the autonomous navigation stack: localisation and mapping with NURBSLAM, and planning with ASTRO. The primary motivation behind NURBSLAM was for the map produced to be useful for both localisation and trajectory planning; hence, the combined demonstration of NURBSLAM with ASTRO provides a necessary validation of what NURBSLAM aims to achieve. In this chapter, NURBSLAM and ASTRO are brought together along with sensing and image processing layers to demonstrate a near-complete autonomous navigation stack (control is not implemented). What will be presented are tests that demonstrate online mapping, localisation and planning in an unknown environment.

The combined autonomous navigation system will be demonstrated in a novel simulation framework, utilising a game development engine, Unreal [59], connected with the Robot Operating System (ROS). The framework, which is a part of the SpaceCRAFT space mission simulation project [2, 136], is designed to allow rapid testing and evaluation of autonomous navigation algorithms.

In addition to demonstrating the combined NURBSLAM and ASTRO system, the algorithms developed in this thesis will be compared to the current state-of-the-art in heterogeneous systems: with separate algorithms for localisation and mapping.

First, an overview of the SpaceCRAFT robotics simulation framework will be presented, highlighting the benefits of testing robotic navigation algorithms. Then, demonstrations of the combined system of NURBSLAM and ASTRO will be presented, before comparing the performance against the current state-of-the-art.

### 7.1 SpaceCRAFT Robot Simulation Framework

SpaceCRAFT is a large-scale space-mission simulation tool that is in development at the time of publication. The goals for SpaceCRAFT are large, from low-level physics simulations right through full space-mission simulations. The low-level physics simulations include areas such as orbital mechanics, gravity fields, aerodynamics and radiation. Subsystems can also be simulated, such as solar panels, antennas, thrusters. These subsystems can then be combined to create a full system simulation, such as for a satellite, or a Mars rover. Finally, a space mission might integrate multiple systems to assess the operation of the system of systems. A particular simulation might use a subset of these different levels of simulation to satisfy the given purpose. These purposes could include mission design, testing a single component in a larger system, or testing a system in a particular environment. On top of the simulation is an immersive user interface, including virtual reality tools, to visualise, inspect and control a simulation. Many of the capabilities in SpaceCRAFT come from combining existing tools, such as orbit propagators, and high fidelity physics simulators. A large component of SpaceCRAFT is the use of the Unreal Engine [59], a game development tool that provides high-quality graphics, a robust simulation framework and a physics engine. While SpaceCRAFT is initially focused on space applications, the concepts are more broadly applicable to other applications, as are the tools behind SpaceCRAFT. For more details on SpaceCRAFT, refer to [2, 136].

One component of SpaceCRAFT is the simulation of robotic systems, and in particular the autonomous navigation of robotic systems. For this component, SpaceCRAFT is linked with the Robotic Operation System, ROS [186]. ROS brings numerous benefits for robotic applications. Firstly, ROS provides a middle-ware to communicate information between different processes with standardised messaging and timing. By having standardised messages, and robotic systems that are expecting standardised messages, a set of algorithms that works for one robot can be directly applied to another robot. Secondly, ROS comes with many existing robotics algorithms that can readily be applied. Additionally, most of the leading open source algorithms for SLAM, mapping, image processing and trajectory planning are integrated with ROS. By integrating with ROS, SpaceCRAFT can draw from this abundant source of powerful robotic navigation algorithms. The connection with ROS also means that algorithms developed with SpaceCRAFT can be ported onto hardware systems more efficiently.

The combination of Unreal, with powerful visualisations and simulation, with ROS, provides a tool that enables rapid testing and evaluation of algorithms in a wide range of environments. The remainder of this section describes the design of the robotic component of SpaceCRAFT and how it can be used for testing robotic algorithms.

### 7.1.1 Framework Design

The design of the SpaceCRAFT robotic simulation framework is outlined in Fig. 7.1. There are two main components: Unreal and ROS. Unreal provides the simulation engine, environment models, sensor simulation (capturing images of the environment), visualisation and user interaction. The UnrealCV plugin [184] is used for image capture from the virtual environment. ROS provides the robotic navigation algorithms and standardised messaging for robotics. A WebSocket bridge connects these two components, with ROS messages being exchanged across the bridge. On the ROS side, this WebSocket interface uses the rosbridge functionality. On the Unreal side, the UROSBridge project is utilised [85], which allows Unreal functions to receive and send ROS messages.


Figure 7.1. System diagram for the ROS-Unreal robot simulation framework. Purple components connect ROS and Unreal over a web-socket. Yellow components handle trajectory planning and control. Red components are related to localisation. Blue components are related to maps. Green components are related to image capture and processing.

The communications over the ROS bridge are depicted in Fig. 7.1. From Unreal to ROS, sensor messages are sent, in addition to the simulation time and transform messages that give the true state of the robot. From ROS to Unreal, planned trajectories are sent, for the simulated robot to follow, along with the estimated state. For user interaction in an immersive environment, goal locations can also be sent from Unreal to ROS.

An example flow of data starts with a simulation of the sensor layer of the autonomous navigation stack. RGBD images are extracted from an environment in Unreal using UnrealCV [184]. These images are then packed into ROS message to send over the ROS bridge. Tools within ROS are used to fill the role of the image processing layer, by subscribing to the RGBD images and producing a point cloud (a set of 3D points). SLAM and mapping nodes subscribe to point cloud or image messages and use them localise and map. The map and state estimate are used by a trajectory planner, which then sends a planned trajectory back over the ROS bridge to Unreal for the simulated robot to follow.

The data flow is designed so that from the perspective of the algorithms in ROS, Unreal represents the robot. This design means that the same algorithms can be run with a real robot, instead of Unreal, by subscribing to and publishing the same messages. The ability to perform this substitution of a simulated robot with a real robot is where there is much power in integrating ROS with a simulator. The simulation allows for rapid testing, development and iteration of navigation algorithms across a range of environments. Then, once the algorithm is ready, minimal changes are required to apply it to a real robot.

There are previous integrations of ROS with simulators, such as Gazebo [107], Rotor-S [72]. The benefits that Unreal has over existing simulators are the realistic graphics and the immersive user interface. Being a game development engine, Unreal can produce high-fidelity visual graphics, which is hugely beneficial for simulating visual navigation algorithms. The superior user interface can enable more intuitive inspection of the performance of algorithms, even allowing virtual reality interaction to view how a robot is moving through an environment. For example of graphics from Unreal, see Fig. 7.2.


Figure 7.2. Example graphics generated in an Unreal simulation. (a) Comet with spacecraft and a trajectory. (b) Close-up of spacecraft and comet. (c) Dense asteroids with a spacecraft.

The framework can be used to evaluate the performance of robotic navigation algorithms by having Unreal provide the true state of the robot, which is published with a ROS transform message over the ROS bridge. This information can be logged in a rosbag, along with the SLAM tracked states, for comparison and analysis.

The framework has been developed in Ubuntu 16.04 using Unreal Engine 4.18 and ROS Kinetic. The source code of the framework is available at https://github.com/maucoen/UnrealNavigation/.

### 7.2 SLAM Demonstration

Before testing the full integrated system, a test case is run to analyse the performance of NURBSLAM for localisation and mapping, with the use of the simulation framework. Unreal provides the sensor layer and algorithms in ROS provide the image processing layer. The localisation performance is then compared to a current leading SLAM algorithm: ORB-SLAM2 [163].

### 7.2.1 Test Case

A test case is established with a single, central object that the robot travels around in a circular orbit with the view fixed to the object. Only one object is in the space to isolate how NURBSLAM performs with a single object. This scenario is equivalent to observing a spinning object. The object is comet $67-\mathrm{P}$, as depicted in Fig. 7.2.a and Fig. 7.2.b, but not at a true scale for the nature of observations: it is used simply as an example object. RGBD observations are made of the comet at $640 \times 480$ resolution. The images are sent to the image processing and SLAM algorithms, through the ROS bridges. 110 observations are made in $110 \%$ of a complete orbit. Comparing NURBSLAM to ORB-SLAM2 when following the same, circular trajectory, allows a clear comparison between localisation capabilities.

### 7.2.2 Results

The tracked trajectories from both algorithms are shown in Fig. 7.3, where it is clear that ORB-SLAM2 provides far superior localisation. Both algorithms have a slow drift away from the truth to the outside of the circle, but the drift of ORB-SLAM2 is substantially less, as shown in Figs. 7.4 and 7.5. NURBSLAM suffers from large angular errors, especially near observation 65 , which place the tracked trajectory into a different plane and hence build up linear errors. This source of inaccuracies can also be seen in the odometry-error analysis in Fig. 7.6, where the translational odometry-error is generally low, but large spikes in angular error occur regularly. ORB-SLAM2, in contrast, has a low and consistent drift in error, and generally low odometry-error. The method of computing angular-error and odometry-error is the same as is explained in Section 3.3. Loop-closure is also achieved by ORB-SLAM2 after one full orbit, which corrects the drift and dramatically reduces the error (the loop closure also causes the jump in odometry-error in Fig. 7.6.b). A summary of the tracking errors is presented in Table. 7.1, emphasising the differences in performance.


Figure 7.3. Orbit test case trajectories, comparing NURBSLAM and ORB-SLAM2. Orientation axes are plotted at even intervals along the trajectory. (a) 3D plot. (b) 2D plot highlighting the drift away from the true, circular trajectory.


FIGURE 7.4. Position errors for orbit test case. (a) Errors for NURBSLAM. (b) Errors for ORB-SLAM2.


Figure 7.5. Angular errors for orbit test case. (a) Errors for NURBSLAM. (b) Errors for ORB-SLAM2.


Figure 7.6. Odometry error for orbit test case. (a) Errors for NURBSLAM. (b) Errors for ORB-SLAM2.

Table 7.1. Tracking errors for orbit test case

|  | NURBSLAM | ORB-SLAM2 |
| :---: | :---: | :---: |
| RMSE Position (m) | 1.9 | 0.3 |
| RMSE Angular (deg) | 25.7 | 4.7 |

While ORB-SLAM2 demonstrates superior tracking, it in-fact was unable to initialise on the same test case as NURBSLAM, and required the comet to be doubled in size (compare Fig. 7.2.b which has the comet as twice the size as in Fig. 7.2.a). Additionally, ORB-SLAM2 does not produce a useful 3D map of what was observed. NURBSLAM does, with some example NURBS surfaces from the test shown in Fig. 7.7. The combination of the surfaces does not provide an accurate model of the object, due to the drift in tracking, but they do provide an adequate representation of the obstacle that the object represents.

(a)

(b)

(c)

FIGURE 7.7. Mapping examples from NURBSLAM in the orbit test case. (a) - (c) are different NURBS objects that were generated from the orbit of the object. The true object is a model of 67P, as depicted in Fig. 7.2.

### 7.2.3 Comments

The performance of ORB-SLAM2 is far superior, but there are scenarios where ORB-SLAM2 can not localise: when there are few visual features in the view of the camera. For these scenarios, if there is still depth information, NURBSLAM could provide adequate tracking over a short window.

### 7.3 Full System Demonstration

ASTRO is connected to NURBSLAM through ROS to add the planning layer to the autonomous navigation stack. The resulting system is first tested to demonstrate a capability for online trajectory planning, localisation and mapping. The system is then compared to the current state-of-the-art in heterogeneous approaches, with ORB-SLAM2 [163] being combined with Voxblox [171] to produce an obstacle representation for ASTRO to plan trajectories.

### 7.3.1 Test Case

A similar scenario is tested as in the previous section, with a single central object at the origin that is enlarged for tests with ORB-SLAM2 to allow the algorithm to initialise. The planning task is to travel from one side of the object $(x=-3.8 \mathrm{~m}, y=0 \mathrm{~m}, z=0 \mathrm{~m})$ to the other $(x=3.8 \mathrm{~m}, y=0 \mathrm{~m}, z=0 \mathrm{~m})$. Throughout the trajectory, the gaze of the spacecraft is fixed to the centre of the object. Fig. 7.2.a shows an example stage of the test case with the object, spacecraft and planned trajectory. Observations are made at 0.1 Hz , with replanning at 0.025 Hz , and an average speed of $0.07 \mathrm{~m} / \mathrm{s}$. RGBD observations are made with image sizes of $640 \times 480$. The systems tested are not the full stack of autonomous navigation, as control is not included in the loop.

As with the previous test, this scenario is not intended to represent a realistic environment but is instead designed to demonstrate the concept of integrating SLAM, 3D mapping and trajectory planning with a single representation of the environment. The test case is also designed to demonstrate how the SpaceCRAFT robotics simulation framework can be used to test an evaluate different navigation algo-
rithms. Results will first be presented to demonstrate NURBSLAM in a full system, before comparing the performance to ORB-SLAM2 and Voxblox.

### 7.3.2 Results - NURBSLAM

The test successfully demonstrates the concept of using a single 3D representation for localisation, mapping and trajectory planning. After an initial straight-line trajectory through the object, a NURBS object was generated, and ASTRO successfully planned a trajectory around the obstacle, as shown in Fig. 7.8.a. This plan was updated as more observations were made to adjust the trajectory and use replanning opportunities to further optimise the trajectory (Fig. 7.8.b)


Figure 7.8. Planned, tracked and true trajectories from NURBSLAM in the full system demonstration. Orientation axes are plotted at even intervals along the trajectory. (a) Initial straight-line trajectory and first replanned trajectory with the truth and tracked trajectories. (b) Examples of successively replanned trajectories.

While following the planned trajectory, NURBSLAM tracked position, but with significant error, as shown in Fig. 7.8.a. There is a large amount of drift from the true position, which is mainly in the $z$ direction. This drift is due to ambiguities in rotation that are present when observations are of largely spherical parts of the object, suggesting NURBSLAM relies more on the larger scale structure of an object, then the small scale details. Despite the poor tracking in $z$, the tracking in the $x$ and $y$ plane remains accurate, as shown in Fig. 7.9. The plot of the linear errors in Fig. 7.10.a also shows how the main source of error is in the $z$ direction. The start of the drift in $z$ is evident in the plot of angular error, Fig. 7.10.b. Near step 30, there is a large jump in angular error because of an observation that has poor alignment with the map object. NURBSLAM is still able to track after this jump, but from the state with large angular error, hence the estimate moves away from the true trajectory, increasing the error in $z$. The tracking errors are summarised in Table 7.2.


Figure 7.9. Trajectory from NURBSLAM in the full system demonstration with top-down view. Orientation axes are plotted at even intervals along the trajectory.


Figure 7.10. Tracking errors for NURBSLAM in the full system demonstration. (a) Position errors. (b) Angular errors.

The results show that NURBSLAM has a soft failure mode: it can recover from large errors and continue to track odometry accurately, as shown in Fig. 7.11. There are large spikes in odometry error around step 30 , after which the odometry error becomes low again. This soft failure is because of the generation of new NURBS surfaces when there are failed alignments. New surfaces ensure there is consistently a NURBS surface for observations to be matched to so that the change in state can be tracked.


Figure 7.11. Odometry errors for NURBSLAM in the full system demonstration.

NURBSLAM generated five NURBS objects in the test, each with accurate modeling of the observed surface of the object, as shown in Fig. 7.12.a-c. The drift in tracking, though, means that newly generated objects can be offset from the true position, leading to a slightly enlarged representation of the 3D object from each of the NURBS surfaces (Fig. 7.12.d). Nonetheless, the collection of objects provides an adequate representation of the object for use as an obstacle representation.


Figure 7.12. Mapping examples from NURBSLAM in the full system demonstration. (a)-(c) Different NURBS objects that were generated in the test. (d) Combination of the objects from (a)-(c). The true object is a model of 67 P , as depicted in Fig. 7.2.

### 7.3.3 Results - Performance Comparison

The system combining ORB-SLAM2 and Voxblox was run on the same test case as for NURBSLAM, but with 67P enlarged to enable ORB-SLAM2 to initialise. Multiple tests were run with a range of trajectories ${ }^{1}$ and ORB-SLAM2 failed to track the complete trajectory in approximately $90 \%$ of cases. These failures were due to three different scenarios. The first scenario is when there were low numbers of features in the image, as occurs when the robot is too far away from the object. The second scenario is when the feature movement from frame to frame is too fast for the frame-rate, which is the case when the robot is too close to the object. The final scenario is when there are changes in lighting conditions, causing variations in features, which can be more dramatic when moving close to the object.

When the tracking is successful throughout a trajectory, the result is very accurate, as shown in Fig. 7.13.a. Accurate tracking allows Voxblox to create an ESDF that is suitable for trajectory planning, as shown in Fig. 7.13.b. For trajectory planning, the ESDF representation performs equivalently to the NURBS representation, showing the suitability for NURBS as an obstacle representation. The errors from the tracking are summarised in Table. 7.2.


Figure 7.13. Full system results for ORB-SLAM2 with Voxblox and ASTRO. (a) True and tracked trajectory. Orientation axes are plotted at even intervals along the trajectory. (b) Example planned trajectory from ASTRO in green around the ESDF that is generated by Voxblox from the entire trajectory. Purple voxels are further away from the surface of the object, with a transition of colours to red voxels that are on the surface of the object.

TABLE 7.2. Tracking errors for full system demonstration

|  | NURBSLAM | ORBSLAM |
| :---: | :---: | :---: |
| RMSE Pos (m) | 0.95 | 0.04 |
| RMSE Ang (deg) | 18.98 | 0.10 |

[^19]
### 7.3.3.1 Computational Load Analysis

With the full systems running, NURBSLAM requires five times more computational resources than ORB-SLAM2 with Voxblox. The primary source of this computational load in NURBSLAM is the RANSAC alignment step. Future work could look to first extract robust keypoints, and then use keypoints for alignment to give a substantial improvement in computational speed.

### 7.4 Conclusions and Discussion

The results presented in this section have successfully demonstrated the concept of using one central 3D representation for mapping, localisation and trajectory planning, with RGBD observations. However, the hypothesis that a homogeneous approach to SLAM and 3D mapping would provide superior efficiency to a heterogeneous approach is not supported by the results. The current version of NURBSLAM has a higher computational load and is less accurate than the combination of ORB-SLAM2 and Voxblox, which have been optimised for their respective tasks of SLAM and 3D mapping. Nonetheless, the testing results have shown the potential for using 3D objects as the representation for SLAM by providing the ability to localise when there are few visual features or dynamic lighting: scenarios that cause ORB-SLAM2 to fail. NURBSLAM also has a softer failure mode, to continue to track odometry after an error, where other SLAM algorithms have a hard failure and completely lose localisation. Additionally, by modelling individual 3D objects, NURBSLAM provides a representation with potential applications for dynamic obstacles, object interaction and object classification.

The testing has identified that further developments to NURBSLAM to improve efficiency and accuracy could see the algorithm filling a role in autonomous navigation where visual features are sparse, and observations are infrequent. A possible avenue to strengthen the capability of NURBSLAM is to develop a hybrid combination of 3D point features with a NURBS representation. Feature points could anchor the NURBS surfaces and support more accurate localisation, while the NURBS surface would define the physical shape and obstacle characteristics. The modification of the SLAM algorithm to use pose-graph optimisation in contrast to Kalman filtering may also enable more accurate results from NURBSLAM.

Further extension of the tests presented here could stress-test the combined system of localisation, mapping and trajectory planning with more numerous obstacles in close proximity and a goal that requires flight near to these obstacles. Such a test would highlight how well the system can adapt to localisation errors by updating the map and then updating the planned trajectory. The proximity to obstacles would also more clearly show the potential impact of localisation errors.

The comparison of NURBSLAM to ORB-SLAM and Voxblox demonstrates the benefit of the SpaceCRAFT robotics simulation framework. It enables the quick testing and evaluation of autonomous navigation algorithms in a visually detailed environment. Future work could further exploit this capability to test algorithms in a broad range of environments.

The tests presented in this chapter were missing an essential layer of the autonomous navigation stack that is needed for further evaluation and development of NURBSLAM. A critical next step for the simulation framework is to integrate the control layer, to have the simulated robot tracking a planned trajectory based on its estimated position. With control implemented, there is a feedback loop
with localisation, mapping and trajectory planning that is important to characterise. For example, an estimation drift could lead to the true robot being closer to an obstacle. This obstacle would be mapped, and the trajectory adjusted to avoid the obstacle. If the adjustment is too slow or the error too large, then there could be a collision with the obstacle. The following step is to replace the sensor layer, which is currently simulated by Unreal, with images from real cameras. The impact of the noise and distortion of real cameras should be considered when testing autonomous navigation systems. Finally, the system should be integrated and tested on to run on hardware suitable for flying robots.


## Conclusion

This thesis has presented work on several layers of the autonomous navigation stack, with a goal to improve capabilities for flying robots to navigate autonomously near obstacles. A thorough review of the current state-of-the-art provided the context for the work presented and highlighted the gaps in current capabilities that were tackled in the thesis.

Within the localisation and mapping layers, one of the gaps addressed is that no existing Simultaneous Localisation And Mapping (SLAM) algorithms that used stereo or depth cameras can produce a map that is suitable for trajectory planning. Instead, the current leading is to use a separate 3D mapping algorithm. Therefore, a new algorithm was presented to combine localisation and 3D mapping with a common representation of 3D objects. Non-Uniform Rational B-Spline (NURBS) is the 3D representation used, with algorithms presented to use NURBS for mapping, localisation and obstacle representation for trajectory planning. NURBS provide a continuous 3D representation that can be sampled at any resolution, making it adaptable to the range of required tasks. Beyond localisation and mapping, the representation has the potential to be used for dynamic obstacles, object interaction and object classification. The algorithm, referred to as NURBS Localisation and Mapping (NURBSLAM), was tested to successfully demonstrate the concept of using one 3 D representation for localisation, mapping and trajectory planning. Mapping from a known state was able to achieve average errors of less than 5 mm , and localisation with a pre-mapped environment had tracking errors of less than $1 \%$ of the total path length. Running SLAM, tracking errors were within $2 \%$ of the total path length with mapping errors as low as 2 cm . The NURBS objects produced were able to be successfully used for trajectory planning and provided a suitable cost profile by using surface normals to compute a signed distance.

The planning layer was addressed by identifying that there is a loss in optimality in current leading approaches that combine a global, obstacle-aware path planner with a local trajectory optimiser that does not consider obstacles. The Admissible Subspace TRajectory Optimiser (ASTRO) was extended from an early version to provide the capability to plan dynamically-optimised trajectories with consideration of a range of obstacles. This capability provides a middle ground between a global planner and local
optimiser. Developments to the constraint formulations of ASTRO enable a wide range of constraints to be efficiently handled, as is demonstrated in a set of simulated tests cases, including complex scenarios with many obstacles. ASTRO can efficiently solve these complex, non-convex scenarios with a suite of optimisation techniques. These techniques were developed and analysed to show the benefit they bring to optimisation performance.

Existing methods of handling dynamic obstacles for flying robots are either short-term planners, not considering the dynamic-optimality of the trajectory over a long time-horizon, or are very conservative. A class of dynamic obstacles are included in ASTRO where the movement of the obstacles are encoded in the formulation, allowing the algorithm to consider time-dependent obstructions. The result is the ability to utilise all the free-space that is available to enable more optimal trajectories. This capability is demonstrated in challenging test cases with multiple moving objects and restricted areas of operation. The modelled size of dynamic obstacles is grown based on the uncertainty in position to ensure safe trajectories. These developments to ASTRO were partly informed by an analysis of experiments carried out with an earlier version of the algorithm on the SPHERES robotic satellites on the International Space Station.

The layer below planning, control, was analysed through an investigation of the singularities of the differential flatness transformation for quadrotors, a critical part in trajectory tracking controllers. The analysis identified where existing transformation methods fail and proposed a new, combined method that is robust throughout all scenarios tested. Flight tests demonstrated that the issues with the transformation can indeed be experienced in flight and that the combined method addresses these issues. Challenges remain to control yaw through orientations were it is ill-defined.

A combination of the planning and control layer was also considered by addressing a gap in the literature: there has not been an analysis of how the method of considering obstacles in trajectory optimisation impacts the dynamic-feasibility of the resulting trajectory. Three methods of planning trajectories with obstacles were assessed, through a batch of tests cases and a series of flight tests. The flight tests demonstrated that there is an impact on tracking performance related to the dynamicfeasibility of the trajectory. ASTRO was one of the algorithms tested, with modifications to adapt to quadrotors. By including obstacles directly in the optimisation, ASTRO was shown to produce the smoothest trajectories that were the easiest to track, for scenarios when flying near to obstacles. The trade-off is that ASTRO has larger computation times as it needs to solve a non-convex optimisation.

Finally, the localisation, mapping and planning layer were considered together to demonstrate the concept of using one 3 D representation for all tasks. The combined system was tested in a novel simulation framework that utilises a game development engine to provide high fidelity visuals. Using this simulator, NURBSLAM was combined with ASTRO to successfully run online with RGBD data, proving the concept that a single 3 D representation can be used for the localisation, mapping and planning layers. This system was then compared with the current state-of-the-art visual SLAM and 3D mapping algorithms. The results do not support the hypothesis that a homogeneous approach to SLAM and mapping would be more efficient than a heterogeneous approach: the current implementation NURBSLAM was less efficient and less accurate in tracking. However, NURBSLAM was shown to be more robust in scenarios with sparse visual features, successfully operating in cases where visual SLAM algorithms fail. Additionally, NURBSLAM has a soft failure mode, with the ability to quickly
recover from errors and continue tracking odometry, rather than completely losing localisation.
Overall, this thesis has presented contributions throughout the autonomous navigation stack. The concept of NURBSLAM has been demonstrated, with further work required to improve performance and characterise scenarios where it provides an advantage over split SLAM and mapping approaches. ASTRO provides a capability in between a global planner and a local optimiser to provide dynamically optimal trajectories around obstacles; a capability that could be applied to a range of flying robots. The analysis of the differential flatness transformation and the dynamic-feasibility of trajectories provides insight on the limitations of existing algorithms, which can be of use to those developing and implementing such algorithms. The new method for performing the differential flatness transformation provides a robust alternative to support high acceleration flight.


## Future Work

There are numerous avenues for further development of the work presented in this thesis, to improve performance and enhance capability. These areas of future work are summarised below. For NURBSLAM, this work presented an initial formulation and a demonstration of the concept. From this state the algorithm can be improved with the following work:

- Using keypoints for the localisation alignment to improve computational speed.
- Improving mapping performance by using overlapping data from a new observation to update the existing surface.
- Improving the formulation for observation uncertainty in the EKF to better reject bad alignments.
- Integrating IMU information into the EKF.
- Investigating a pose-graph optimisation version of NURBSLAM using the GTSAM toolbox [47].
- Investigating a hybrid system of representation combining 3D point-feature descriptors with NURBS surfaces.

The tests performed for NURBSLAM were for a single object, to isolate the operation of the algorithm and test how effective it is with minimal information. Similarly, the tests were with simulated data, to isolate the operation of the algorithm from sensor and image processing considerations. Critical next steps are to expand to more test cases:

- Testing NURBSLAM for multi-object environments.
- Testing the combined NURBSLAM and ASTRO system with many obstacles in close proximity to stress test the localisation, mapping, planning interactions.
- Testing NURBSLAM with real data from an RGBD sensor, and segmentation of the observations.
- Testing NURBSLAM in a range of environments, and compare to ORB-SLAM2, to characterise performance and identify where NURBSLAM is superior.
- Testing of NURBSLAM with IMU measurements integrated into the filter.

The map produced by NURBSLAM is not only a map of obstacles but also a map of objects. Therefore, the NURBS representation could potentially be used for object recognition and grasping by using the
shape information for the given object. The objects could also be given motion models to represent dynamic obstacles. These areas could be investigated further.
ASTRO can be improved, and its capability expanded in the following areas:

- Further development and analysis of performance constraints.
- Including attitude in the optimisation, potentially using Modified Rodrigues Parameters as the attitude representation.
- Flight tests of ASTRO with dynamic obstacles to thoroughly assess how ASTRO handles dynamic obstacles.
- Incorporation of perception constraints.
- A tighter integration with perception to include uncertainty of obstacle locations and predict the movement of dynamic obstacles.
- An in-depth analysis to understand the impact of changing the number of optimisation coefficients.
- Tests in simulated environments to analyse the varying performance of ASTRO, UNCO and TACO with varying obstacle density.

Additionally, future developments of ASTRO could look to make the algorithm more robust and adaptable to a range of scenarios, with less tuning of parameters required.

For the differential flatness transformation, there remain areas of improvement to have a robust transformation method through all orientations. Avenues for investigation in this area include:

- Deriving the differential flatness transformation with Modified Rodrigues Parameters.
- Removing specification of yaw, and only planning the yaw rate.
- Analyse, implement and test recent work from Watterson and Kumar [224] that use the Hopf Fibration to manage the singularity.
- Demonstrate flight tests with full inversion.

The SpaceCRAFT robotic simulation framework has numerous avenues of development to implement a range of robots and environments, to improve the efficiency of image transfer over the ROS bridge, and to provide more immersive visualisation and user control. This tool can allow NURBSLAM, ASTRO and other algorithms to be tested in a large range of environments, from indoor offices and warehouses to inside the International Space Station and on the surface of Mars. In doing so, the tool can help to develop algorithms for application in a particular environment or to be robust in a variety of environments.

The system demonstrations of NURBSLAM and ASTRO, compared to ORBSLAM, Voxblox and ASTRO were missing the control layer of the autonomous navigation stack. It is an essential next step to implement that control layer so that the robot is moving to minimise the error between the estimated pose and the desired pose. Following these tests, the combined autonomous navigation systems should be implemented in hardware. First, a hardware-in-the-loop simulation can be performed, where a real robot is flying, but the observations come from within a simulated environment from SpaceCRAFT, similarly to what is presented in [204]. The final goal is to implement and test the full system onboard a quadrotor for autonomous navigation in an unknown environment.

## SVD for Determining Transformations

Given a dataset of $n 3 \mathrm{D}$ observation points, $d_{i}$ and $n 3 \mathrm{D}$ points on a surface to align these points to, $p_{i}$, the algorithm to compute alignment with the Singular Value Decomposition (SVD) is effectively a least squares optimisation of:

$$
\begin{equation*}
d_{i}-R p_{i}-T \tag{A.1}
\end{equation*}
$$

The rotation matrix, $R$ and the translation $T$ are what is being solved for. First the datasets are normalised by their mean ( $\bar{d}, \bar{p}$ ):

$$
\begin{align*}
& d_{c_{i}}=d_{i}-\bar{d}  \tag{A.2}\\
& p_{c_{i}}=m_{i}-\bar{p} \tag{A.3}
\end{align*}
$$

Then the data is combined to produce a $3 \times 3$ correlation matrix:

$$
\begin{equation*}
W=\sum_{i=0}^{n} p_{c_{i}} d_{c_{i}}^{T} \tag{A.4}
\end{equation*}
$$

where each data point is a $3 \times 1$ vector. SVD is then performed on $W$ to get $W=U \Lambda^{`} V^{T}$. The rotation matrix is then computed with:

$$
\begin{equation*}
R=V U^{T} \tag{A.5}
\end{equation*}
$$

Using the rotation matrix, the translation is computed as:

$$
\begin{equation*}
T=\bar{d}-R \bar{p} \tag{A.6}
\end{equation*}
$$

Refer to [53] for a comparison of using SVD for alignment compared to other methods.

## Subspace Projection

Presented here are the derivations of the projected subspace formulation, for both a gradient step in coefficients and a full coefficient step. The derivation is for a single dimension, but the $i$ subscript is dropped for cleaner notation.

## B. 1 Gradient

A gradient step is split into components that comply with and do not comply with the boundary conditions:

$$
\begin{equation*}
\delta \mathbf{C}=\delta \mathbf{C}_{\perp}+\delta \mathbf{C}_{\|} \tag{B.1}
\end{equation*}
$$

where the split of components is defined by:

$$
\begin{equation*}
\mathbf{P}_{B C} \delta \mathbf{C}_{\perp}=\mathbf{0} \tag{B.2}
\end{equation*}
$$

What we want to derive is an expression to get $\delta \mathbf{C}_{\perp}$ from $\delta \mathbf{C}$. This amounts to a projection of $\delta \mathbf{C}$ onto the subspace of feasible solutions. Eq. B. 1 is used, along with:

$$
\begin{align*}
\mathbf{P}_{B C} \delta \mathbf{C} & =\mathbf{P}_{B C} \delta \mathbf{C}_{\perp}+\mathbf{P}_{B C} \delta \mathbf{C}_{\|}  \tag{B.3}\\
& =\mathbf{P}_{B C} \delta \mathbf{C}_{\|}
\end{align*}
$$

We isolate $\mathbf{C}_{\| \mid}$in Eq. B.3, using the Moore-Penrose pseudo-inverse of $\mathbf{P}_{B C}: \mathbf{P}_{B C}^{+}$:

$$
\begin{equation*}
\delta \mathbf{C}_{\|}=\left[\mathbf{P}_{B C}^{T}\left(\mathbf{P}_{B C} \mathbf{P}_{B C}^{T}\right)^{-1}\right] \mathbf{P}_{B C} \delta \mathbf{C}=\mathbf{P}_{B C}^{+} \mathbf{P}_{B C} \delta \mathbf{C} \tag{B.4}
\end{equation*}
$$

Substituting Eq. B. 4 into Eq. B.3, we get:

$$
\begin{equation*}
\delta \mathbf{C}=\delta \mathbf{C}_{\perp}+\mathbf{P}_{B C}^{+} \mathbf{P}_{B C} \delta \mathbf{C} \tag{B.5}
\end{equation*}
$$

Rearranging to solve for $\mathbf{C}_{\perp}$, we get:

$$
\begin{align*}
\delta \mathbf{C}_{\perp} & =\delta \mathbf{C}-\mathbf{P}_{B C}^{+} \mathbf{P}_{B C} \delta \mathbf{C}  \tag{B.6}\\
& =\left[\mathbf{I}-\mathbf{P}_{B C}^{+} \mathbf{P}_{B C}\right] \delta \mathbf{C}
\end{align*}
$$

This equation is the projection of $\delta \mathbf{C}$ onto the subspace of feasible solutions.

## B. 2 Coefficients

When solving directly for coefficients, the goal is to project a coefficient set onto the space of feasible coefficient sets. The process to derive this projection is very similar to that for the gradients. We first split the coefficients:

$$
\begin{equation*}
\mathbf{C}=\mathbf{C}_{\perp}+\mathbf{C}_{\|} \tag{B.7}
\end{equation*}
$$

Defining $\mathbf{C}_{\perp}$ and $\mathbf{C}_{\|}$such that Eq. B. 7 holds and:

$$
\begin{equation*}
\mathbf{P}_{B C} \mathbf{C}_{\perp}=\mathbf{X}_{B C} \tag{B.8}
\end{equation*}
$$

That is, $\mathbf{C}_{\perp}$ satisfies the boundary conditions, $\mathbf{X}_{B C}$. This property is used to get an expression for $\mathbf{C}_{\|}$:

$$
\begin{align*}
\mathbf{P}_{B C} \mathbf{C} & =\mathbf{P}_{B C} \mathbf{C}_{\perp}+\mathbf{P}_{B C} \mathbf{C}_{\|}  \tag{B.9}\\
& =\mathbf{X}_{B C}+\mathbf{P}_{B C} \mathbf{C}_{\|} \tag{B.10}
\end{align*}
$$

Taking the pseudo-inverse, $\mathbf{C}_{| |}$is isolated :

$$
\begin{equation*}
\mathbf{C}_{\|}=\mathbf{P}_{B C}^{+}\left(\mathbf{P}_{B C} \delta \mathbf{C}-\mathbf{X}_{B C}\right) \tag{B.11}
\end{equation*}
$$

On substitution back into Eq. B.7, the projection to get $\mathbf{C}_{\perp}$ can be produced:

$$
\begin{equation*}
\mathbf{C}_{\perp}=\mathbf{C}-\mathbf{P}_{B C}^{+}\left(\mathbf{P}_{B C} \mathbf{C}-\mathbf{X}_{B C}\right) \tag{B.12}
\end{equation*}
$$

Because $\mathbf{C}_{\perp}$ satisfies the boundary conditions, it is the coefficient component that we want to maintain the boundary conditions. Eqn. B. 11 takes that component from an arbitrary coefficient set.

## QuATERNION MATHS

The quaternion is an important representation of attitude that is singularity free throughout all orientations. The key mathematical definitions and operations are summarised here for reference. Useful resources for more information are [45, 49, 81].

## C. 1 Quaternion Definition

At the core of a quaternion representation of attitude is a vector, e, and an angle to rotate about that vector, $\theta$. This representation of attitude is the easiest to understand conceptually. The quaternion gives a convenient mathematical way to represent this vector and rotation, and use it for a range of operations.

The mathematical definition of a quaternion takes from complex numbers:

$$
\begin{equation*}
\mathbf{q}=q_{0}+i q_{1}+j q_{2}+k q_{3} \tag{C.1}
\end{equation*}
$$

where $q_{1}, q_{2}$ and $q_{3}$ contain information on the rotation vector, and $q_{0}$ on the rotation angle. This definition is used for derivations of mathematical operations. The quaternion is normally represented as a $4 \times 1$ vector of these terms. When using a quaternion for representing attitude, it is defined as:

$$
\begin{align*}
\mathbf{q} & =\left[\begin{array}{l}
q_{0} \\
q
\end{array}\right]=\left[\begin{array}{r}
\cos \left(\frac{\theta}{2}\right) \\
\mathbf{e} \sin \left(\frac{\theta}{2}\right)
\end{array}\right]  \tag{C.2}\\
& =\left[\begin{array}{l}
q_{0} \\
q_{1} \\
q_{2} \\
q_{3}
\end{array}\right]=\left[\begin{array}{c}
\cos \left(\frac{\theta}{2}\right) \\
e_{1} \sin \left(\frac{\theta}{2}\right) \\
e_{2} \sin \left(\frac{\theta}{2}\right) \\
e_{3} \sin \left(\frac{\theta}{2}\right)
\end{array}\right]
\end{align*}
$$

The rotation $\theta$ is present in each term, the bottom three quaternion terms, $\underline{q}$, are the vector components, where $\mathbf{e}$ is the unit vector for the vector of rotation. The $q_{0}$ term is referred to as the scalar component.

With this definition, it can be useful to mentally think of a few simple rotations, and how they are represented with quaternions. For instance, for a pitch of $90^{\circ}$, we have $\cos (\theta / 2)=\sin (\theta / 2)=0.707$, and the rotation is purely about the positive $y$-axis, so: $\mathbf{q}=[0.707,0,0.707,0]$. Similarly for a $90^{\circ}$ yaw, we have $\mathbf{q}=[0.707,0,0,0.707]$.

There are different conventions for representing the quaternion, including having the scalar component last, or using the notation:

$$
\mathbf{q}=\left[\begin{array}{c}
q_{w}  \tag{C.3}\\
q_{x} \\
q_{y} \\
q_{z}
\end{array}\right]
$$

to clearly differentiate the vector and scalar components. The definition in Eqn. C. 2 will be used here.
Taking from complex numbers, a quaternion conjugate is obtained by negating the complex terms:

$$
\mathbf{q}^{c}=\left[\begin{array}{c}
\cos \left(\frac{\theta}{2}\right)  \tag{C.4}\\
-\mathbf{e} \sin \left(\frac{\theta}{2}\right)
\end{array}\right]
$$

The inverse of a quaternion uses this conjugate:

$$
\begin{equation*}
\mathbf{q}^{-1}=\frac{\mathbf{q}^{c}}{|\mathbf{q}|} \tag{C.5}
\end{equation*}
$$

where the quaternion norm is computed similarly to any other vector:

$$
\begin{equation*}
|\mathbf{q}|=\sqrt{q_{0}^{2}+q_{1}^{2}+q_{2}^{2}+q_{3}^{2}}=\sqrt{\mathbf{q} \cdot \mathbf{q}} \tag{C.6}
\end{equation*}
$$

The quaternion as used for attitude representation is a unit quaternion: $|\mathbf{q}|=1$. There are cases where it would be desired to normalise the quaternion, to make it a unit quaternion. This is done by:

$$
\begin{equation*}
\mathbf{q}_{u n i t}=\frac{\mathbf{q}}{|\mathbf{q}|} \tag{C.7}
\end{equation*}
$$

Because unit quaternions are used for rotation, the inverse is equal to the conjugate: $\mathbf{q}^{-1}=\mathbf{q}^{c}$. With regards to rotation, this inverse makes intuitive sense: to do an inverse rotation about a rotation vector, you can rotate by the negative of $\theta$. Because $\sin (-\alpha)=-\sin (\alpha)$ and $\cos (-\alpha)=\cos (\alpha)$, you get $\mathbf{q}^{c}$ if $-\theta$ is used in place of $\theta$ in Eq. C.2. The inverse can also be thought of as a positive $\theta$ rotation about a vector in the opposite direction to $\mathbf{e}$, i.e. the vector $\mathbf{e}$ is negated.

Pure quaternions are quaternions where the scalar term is zero, and are not necessarily unit quaternions. These are generally used to represent a spatial vector, $\mathbf{v}$ as quaternions for use in quaternion multiplication:

$$
\left[\begin{array}{l}
0  \tag{C.8}\\
\mathbf{v}
\end{array}\right]
$$

## C. 2 Quaternion Multiplication

The multiplication of quaternions is derived from the complex numbers definition, Eq. C.1, using the fact that $i^{2}=j^{2}=k^{2}=i j k=-1, i j=k, j i=-k, k i=j, k j=-i$, etc. (see [45] for a derivation). For two quaternions, $\mathbf{q}$ and $\mathbf{p}$, this results in:

$$
\mathbf{q} \otimes \mathbf{p}=\left[\begin{array}{c}
q_{0} p_{0}-\underline{q}^{T} \underline{p}  \tag{C.9}\\
\underline{q} p_{0}+q_{0} \underline{p}+\underline{q} \times \underline{p}
\end{array}\right]
$$

This multiplication is not commutative, $\mathbf{q} \otimes \mathbf{p} \neq \mathbf{p} \otimes \mathbf{q}$. It can be convenient to represent this operation instead with matrices, $\mathbf{q} \otimes \mathbf{p}=\mathbf{q}^{+} \mathbf{p}$, where $\mathbf{q}^{+}$is a pre-multiplication matrix:

$$
\mathbf{q}^{+}=\left[\begin{array}{cc}
q_{0} & -\underline{q}^{T}  \tag{C.10}\\
\underline{q} & q_{0} \mathbf{I}+\underline{\hat{q}}
\end{array}\right]
$$

The matrix $I$ is the $3 \times 3$ identity matrix, and the hat operator, a maps a vector to a cross product matrix:

$$
\mathbf{a} \times \mathbf{b}=\hat{\mathbf{a}} \mathbf{b}=\left[\begin{array}{rrr}
0 & -a_{3} & a_{2}  \tag{C.11}\\
a_{3} & 0 & -a_{1} \\
-a_{2} & a_{1} & 0
\end{array}\right] \mathbf{b}
$$

The matrix $\mathbf{q}^{+}$can also be represented with the original quaternion and a sub-matrix:

$$
\begin{align*}
& \mathbf{q}^{+}=\left[\begin{array}{ll}
\mathbf{q} & E_{q}^{+}
\end{array}\right]  \tag{C.12}\\
& E_{q}^{+}=\left[\begin{array}{c}
-\underline{q}^{t} \\
q_{0} \mathbf{I}+\underline{\hat{q}}
\end{array}\right]=\left[\begin{array}{rrr}
-q_{1} & -q_{2} & -q_{3} \\
q_{0} & -q_{3} & q_{2} \\
q_{3} & q_{0} & -q_{1} \\
-q_{2} & q_{1} & q_{0}
\end{array}\right] \tag{C.13}
\end{align*}
$$

Similar matrices can be constructed for the post-multiplication, $\mathbf{q} \otimes \mathbf{p}=\mathbf{p}^{-} \mathbf{q}$ :

$$
\begin{align*}
\mathbf{p}^{-} & =\left[\begin{array}{cc}
p_{0} & -\underline{p}^{T} \\
\underline{p} & p_{0} \mathbf{I}-\underline{\hat{p}}
\end{array}\right]  \tag{C.14}\\
& =\left[\begin{array}{ll}
\mathbf{p} & E_{p}^{-}
\end{array}\right]  \tag{C.15}\\
E_{p}^{-} & =\left[\begin{array}{c}
-\underline{p}^{t} \\
p_{0} \mathbf{I}-\underline{\hat{p}}
\end{array}\right]=\left[\begin{array}{rrr}
-p_{1} & -p_{2} & -p_{3} \\
p_{0} & p_{3} & -p_{2} \\
-p_{3} & p_{0} & p_{1} \\
p_{2} & -p_{1} & p_{0}
\end{array}\right] \tag{C.16}
\end{align*}
$$

The forms of multiplication with matrices are useful for implementation in code, and for doing any derivations with quaternions. The complete form of the matrices and their terms is:

$$
\begin{align*}
\mathbf{q} \mathbf{p} & =\left[\begin{array}{rrrr}
q_{0} & -q_{1} & -q_{2} & -q_{3} \\
q_{1} & q_{0} & -q_{3} & q_{2} \\
q_{2} & q_{3} & q_{0} & -q_{1} \\
q_{3} & -q_{2} & q_{1} & q_{0}
\end{array}\right] \mathbf{p}  \tag{C.17}\\
& =\left[\begin{array}{rrrr}
p_{0} & -p_{1} & -p_{2} & -p_{3} \\
p_{1} & p_{0} & p_{3} & -p_{2} \\
p_{2} & -p_{3} & p_{0} & p_{1} \\
p_{3} & p_{2} & -p_{1} & p_{0}
\end{array}\right] \mathbf{q} \tag{C.18}
\end{align*}
$$

The primary uses for quaternions all use quaternion multiplication.

## C. 3 Quaternion Multiplication properties

Quaternion multiplication is not commutative, $\mathbf{q} \otimes \mathbf{p} \neq \mathbf{p} \otimes \mathbf{q}$, but is associative: $\mathbf{q} \otimes(\mathbf{p} \otimes \mathbf{r})=(\mathbf{q} \otimes \mathbf{p}) \otimes \mathbf{r}$, and distributive: $\mathbf{p} \otimes(\mathbf{q}+\mathbf{r})=\mathbf{q} \otimes \mathbf{p}+\mathbf{q} \otimes \mathbf{r}$. The quaternion multiplied by its inverse gives the identity quaternion:

$$
\mathbf{q} \otimes \mathbf{q}^{c}=\mathbf{q}^{c} \otimes \mathbf{q}=\left[\begin{array}{l}
1  \tag{C.19}\\
0 \\
0 \\
0
\end{array}\right]
$$

These properties allow, for instance: $\mathbf{q} \otimes \mathbf{p}=\mathbf{r}$ to be rearranged to give $\mathbf{p}=\mathbf{q}^{c} \otimes \mathbf{r}$.
A scalar multiplied by a quaternion scales the terms in the quaternion vector:

$$
\alpha \mathbf{q}=\left[\begin{array}{c}
\alpha q_{0}  \tag{C.20}\\
\alpha q_{1} \\
\alpha q_{2} \\
\alpha q_{3}
\end{array}\right]
$$

## C. 4 Transformations vs. Rotations

A rotation is the inverse of a transformation, and there can be confusion on which is being used. A transformation does not move a vector but instead describes it in a different coordinate frame. A rotation moves a vector to change where it is pointing in any reference frame. See [81] for a discussion of the differences.

For example: consider a runway and a quadrotor. The global axis is fixed to the runway with the $x$ axis along the runway, pointing North, the $z$ axis is up, and the $y$ axis is west. The body-axis is fixed to the quadrotor with $x$ through the forward end, that is initially pointing down the runway. The quadrotor takes off and turns to point due West.
Rotation As the quadrotor takes off and turns, the vector describing the direction of the forward end in the global frame is rotated by $90^{\circ}$. It starts as $[1,0,0]_{g}$, and ends as $[0,1,0]_{g}$.
Transformation The vector describing the direction of the forward end in the body frame is $[1,0,0]_{b}$, which never changes, as the body frame is fixed to the quadrotor. When hovering and pointing west, if we want to describe this body-axis in the global frame, then we need to do a transformation, the result of which will be $[0,1,0]_{g}$. Note how although the result is the same as for rotation (it should be as both are describing the forward end in the global frame), the quadrotor has not moved at all. No vector has changed, just the frame in which that vector is described. ${ }^{1}$
Transformation 2 Let's say there is a camera on the quadrotor and we want to point it in the direction of the runway. What direction do we need to point it in the body frame? We need to transform the vector giving the direction of the runway in the global frame, $[1,0,0]_{g}$, to the body frame. This transformation is the inverse of the transformation from the body frame to the global frame and gives $[0,-1,0]_{b}$. Again, this is the same vector, it still points down the runway, but it is described in a different frame.
Rotation 2 The camera is pointing forward, $[1,0,0]_{b}$, so we need to rotate it to point it in the right direction, $[0,-1,0]_{b}$. This step amounts to a negative rotation of $90^{\circ}$ about the body $z$ axis. Note how this is a rotation from the body $x$ axis to the global $x$ axis and is the negative of the transformation from the body frame to the global frame.
Summary Rotations move a vector and stay in the same frame of reference. Transformations do not move vectors but change the frame of reference.
The same mathematical operations apply for rotations and transformations, but care needs to be taken to perform the operations in the right direction and order.

## C. 5 Quaternions for Attitude Transformations

For describing the attitude of a vehicle, the attitude quaternion $\mathbf{q}$ encodes the transformation from the body frame to the global frame. It can also be thought of as representing how the body of a vehicle (and the body fixed axes) has rotated from the global axes. The attitude quaternion can be used to transform a spatial vector in the body frame, $\mathbf{x}_{b}$ to the global frame, $\mathbf{x}_{g}$ with:

[^20]\[

\left[$$
\begin{array}{c}
0  \tag{C.21}\\
\mathbf{x}_{g}
\end{array}
$$\right]=\mathbf{q} \otimes\left[$$
\begin{array}{c}
0 \\
\mathbf{x}_{b}
\end{array}
$$\right] \otimes \mathbf{q}^{c}
\]

where all multiplications are quaternion multiplications. For ease of notation, from here forward the pure quaternion representation of a vector will be be assumed whenever a quaternions is multiplying a vector, hence:

$$
\begin{equation*}
\mathbf{x}_{g}=\mathbf{q} \otimes \mathbf{x}_{b} \otimes \mathbf{q}^{c} \tag{C.22}
\end{equation*}
$$

The same equation can be used for rotation, to rotate a vector in the same frame by quaternion $\mathbf{p}$ :

$$
\begin{equation*}
\mathbf{x}_{b}^{\prime}=\mathbf{p} \otimes \mathbf{x}_{b} \otimes \mathbf{p}^{c} \tag{C.23}
\end{equation*}
$$

To compose multiple transformations or rotations, successive quaternions are multiplied in order of application from right to left. We define here the notation $\mathbf{q}_{a b}$, to describe a quaternion that will transform from frame $b$ to frame $a$ (e.g. the quaternion in Eq. C. 22 is $\mathbf{q}_{g b}$ ). Successive transformations from frame $c$ to $b$ to $a$ is given by:

$$
\begin{equation*}
\mathbf{q}_{a c}=\mathbf{q}_{a b} \otimes \mathbf{q}_{b c} \tag{C.24}
\end{equation*}
$$

An intuitive way to reason about this ordering is by observing Eq. C. 22 and composing rotations of that vector. For example, if there is a camera frame $c$, with orientation in the body frame, $b$, given by $\mathbf{q}_{b c}$, and the body attitude in the global frame, $g$ is given by $\mathbf{q}_{b g}$, then representing a camera frame vector in the global frame can be constructed as below:

$$
\begin{align*}
\mathbf{x}_{b} & =\mathbf{q}_{b c} \otimes \mathbf{x}_{c} \otimes \mathbf{q}_{b c}^{c}  \tag{C.25}\\
\mathbf{x}_{g} & =\mathbf{q}_{g b} \otimes \mathbf{x}_{b} \otimes \mathbf{q}_{g b}^{c}  \tag{C.26}\\
& =\mathbf{q}_{g b} \otimes \mathbf{q}_{b c} \otimes \mathbf{x}_{c} \otimes \mathbf{q}_{b c}^{c} \otimes \mathbf{q}_{g b}^{c}  \tag{C.27}\\
& =\mathbf{q}_{g c} \otimes \mathbf{x}_{c} \otimes \mathbf{q}_{g c}^{c} \tag{C.28}
\end{align*}
$$

Therefore:

$$
\begin{equation*}
\mathbf{q}_{g c}=\mathbf{q}_{g b} \otimes \mathbf{q}_{b c} \tag{C.29}
\end{equation*}
$$

The same holds for rotations, but again have caution in that the order of application of rotations to a vector may be the opposite to the order of transformations.

## C.5.1 Quaternions to Rotation Matrices

A rotation matrix can be formed from quaternions with the following matrix (see [49] for a derivation):

$$
\mathbf{R}=\left[\begin{array}{ccc}
\left(q_{0}^{2}+q_{1}^{2}\right)-\left(q_{2}^{2}+q_{3}^{2}\right) & 2\left(q_{1} q_{2}-q_{0} q_{3}\right) & 2\left(q_{0} q_{2}+q_{1} q_{3}\right)  \tag{C.30}\\
2\left(q_{1} q_{2}+q_{0} q_{3}\right) & \left(q_{0}^{2}-q_{1}^{2}\right)+\left(q_{2}^{2}-q_{3}^{2}\right) & 2\left(q_{2} q_{3}-q_{0} q_{1}\right) \\
2\left(q_{1} q_{3}-q_{0} q_{2}\right) & 2\left(q_{2} q_{3}+q_{1} q_{0}\right) & \left(q_{0}^{2}-q_{1}^{2}\right)-\left(q_{2}^{2}-q_{3}^{2}\right)
\end{array}\right]
$$

To extract a quaternion from a rotation matrix, four different inverse mappings can be used, for which we refer to [49]. Conversions to other attitude representations are also presented in [49].

## C. 6 Quaterion Rates

Given a body rates vector, $\underline{\omega}$, the quaternion rates are given by:

$$
\begin{equation*}
\dot{\mathbf{q}}=\frac{1}{2} \mathbf{q} \otimes \underline{\omega} \tag{C.31}
\end{equation*}
$$

Using the matrix form of multiplication, the rates are given by:

$$
\dot{\mathbf{q}}=\left[\begin{array}{rrrr}
0 & -\omega_{x} & -\omega_{y} & -\omega_{z}  \tag{C.32}\\
\omega_{x} & 0 & \omega_{z} & -\omega_{y} \\
\omega_{y} & -\omega_{z} & 0 & \omega_{x} \\
\omega_{z} & \omega_{y} & -\omega_{x} & 0
\end{array}\right]\left[\begin{array}{l}
q_{0} \\
q_{1} \\
q_{2} \\
q_{3}
\end{array}\right]
$$

Note that the current orientation, $\mathbf{q}$ is required.
The inverse; computing the body rates from the quaternion rates can be derived with the quaternion multiplication rules from Eq. C.31, and is given by:

$$
\begin{equation*}
\underline{\omega}=2 \mathbf{q}^{c} \otimes \dot{\mathbf{q}} \tag{C.33}
\end{equation*}
$$

The quaternion acceleration uses the body rate acceleration, $\underline{\underline{\dot{ }}}$ and the quaternion rates:

$$
\begin{equation*}
\ddot{\mathbf{q}}=\frac{1}{2} \mathbf{q} \otimes \underline{\dot{\omega}}-\left(|\dot{\mathbf{q}}|^{2}\right) \mathbf{q} \tag{C.34}
\end{equation*}
$$

The quaternion rates are not unit quaternions, hence the norm is not zero. The body angular accelerations are obtained from the quaternion accelerations by:

$$
\begin{equation*}
\underline{\ddot{\omega}}=2\left(\mathbf{q}^{c} \otimes \ddot{\mathbf{q}}+\dot{\mathbf{q}}^{c} \otimes \dot{\mathbf{q}}\right) \tag{C.35}
\end{equation*}
$$

## C. 7 Quaternion Logarithm and Exponential

The exponential of a quaternion takes a similar form to the exponential of complex numbers, $e^{i \phi}=$ $\cos (\phi)+i \sin (\phi)$, where in this case, the magnitude of the complex part is $|\underline{q}|$ :

$$
e^{q}=e^{q_{0}}\left[\begin{array}{c}
\cos (|\underline{q}|)  \tag{C.36}\\
\underline{\underline{q}} \operatorname{|\underline {q}|} \sin (|\underline{q}|)
\end{array}\right]
$$

Note how this is very similar to the complex exponential, but with the rotation axis giving three dimensions to the complex part.

The quaternion logarithm is the inverse of the exponential, using an "angle" between real and imaginary parts, $\phi$. For quaternions, this is the angle on the imaginary plane with coordinates ( $q_{0},|\underline{q}|$ ):

$$
\begin{equation*}
\phi=\arctan \left(\frac{\underline{\underline{q} \mid}}{q_{0}}\right) \tag{C.37}
\end{equation*}
$$

The logarithm is computed by:

$$
\ln (\mathbf{q})=\left[\begin{array}{c}
\ln (|\mathbf{q}|)  \tag{C.38}\\
\frac{q}{|\underline{q}|} \phi
\end{array}\right]
$$

When considering quaternions for attitude representation, these operations can be considered as a conversion between quaternion and axis-angle type representation, $\mathbf{e} \theta$. The exponential of a pure quaternion with the vector component $\mathbf{e} \theta / 2$ (halving a rotation vector representation) gives:

$$
\exp \left(\left[\begin{array}{c}
0  \tag{C.39}\\
\mathbf{e} \frac{\theta}{2}
\end{array}\right]\right)=\left[\begin{array}{c}
\cos \left(\frac{\theta}{2}\right) \\
\mathbf{e} \sin \left(\frac{\theta}{2}\right)
\end{array}\right]
$$

which is the quaternion representation for same attitude. The inverse can also be considered, by taking the $\log$ of a quaternion describing attitude, where $|\mathbf{q}|=1$, and $\phi=\arctan (\sin (\theta / 2) / \cos (\theta / 2))=\theta / 2$ :

$$
\ln (\mathbf{q})=\left[\begin{array}{c}
0  \tag{C.40}\\
\left.\frac{\underline{q}}{\mid \underline{ }} \right\rvert\, \\
\frac{\theta}{2}
\end{array}\right]=\left[\begin{array}{c}
0 \\
\mathbf{e} \frac{\theta}{2}
\end{array}\right]
$$

The result is half the axis angle representation of the attitude. For more details on these operations and use of them when $|\underline{q}|$ is near zero, refer to [45].

## C. 8 Quaternion Interpolation - SLERP

The evolution of quaternions representing attitude is not linear, hence linearly interpolating between two quaternions is not an accurate approach. Instead, the interpolation should be a progressive rotation from one orientation to the next. Consider an interpolation between $\mathbf{q}_{1}$ and $\mathbf{q}_{2}$. The quaternion defining the rotation between the two orientations is given by:

$$
\begin{equation*}
\Delta \mathbf{q}=\mathbf{q}_{1}^{c} \otimes \mathbf{q}_{2} \tag{C.41}
\end{equation*}
$$

Note that we are considering rotations between quaternions in this case. The mathematics is the same, whether or not they are used for transformations or rotations, but care needs to be taken when considering the order of the quaternions rotations, and the frame in which the rotation is defined. For example, if we want to represent the change in attitude from a roll in the body-axis, this rotation comes after the quaternion that describes the current attitude, so it is about the current body $x$ axis.

The delta quaternion defines a rotation vector and angle to move from one orientation to the next; hence rotating by fractions of the angle interpolates the rotation. To rotate by fractions of the angle, the difference quaternion is transformed into the rotation vector form by the quaternion logarithm:

$$
\left[\begin{array}{c}
0  \tag{C.42}\\
\mathbf{v}
\end{array}\right]=\left[\begin{array}{c}
0 \\
\mathbf{e} \frac{\theta}{2}
\end{array}\right]=\ln (\Delta \mathbf{q})
$$

with the rotation angle $\theta$ being a multiplicative factor of the rotation vector $\mathbf{v}$, the rotation can be incremented by using $\tau \mathbf{v}$, where $\tau$ ranges from 0 to 1 . A given partial rotation then needs to be mapped back into a quaternion with the matrix exponential:

$$
\Delta \mathbf{q}(\tau)=\exp \left(\tau\left[\begin{array}{l}
0  \tag{C.43}\\
\mathbf{v}
\end{array}\right]\right)=\exp \left(\tau \ln \left(\mathbf{q}_{1}^{c} \otimes \mathbf{q}_{2}\right)\right)
$$

This partial rotation quaternion can then be applied to the starting quaternion to get the interpolated rotation, $\mathbf{q}(\tau)$ :

$$
\begin{equation*}
\mathbf{q}(\tau)=\mathbf{q}_{1} \otimes \Delta \mathbf{q}(\tau)=\mathbf{q}_{1} \otimes \exp \left(\tau \ln \left(\mathbf{q}_{1}^{c} \otimes \mathbf{q}_{2}\right)\right) \tag{C.44}
\end{equation*}
$$

This method of interpolation is called Spherical Linear Interpolation, or SLERP.
The right part of Eq. C. 44 can be conveniently represented by the quaternion power:

$$
\begin{equation*}
\mathbf{q}^{t}=\exp (t \ln (\mathbf{q})) \tag{C.45}
\end{equation*}
$$

This power is effectively combining the steps to perform partial rotations for a quaternion: rotating about the same rotation axis, but by a fraction or multiple of $\theta$, set by the value $t$.

## C. 9 Quaternion Finite Differencing

If it is desired to get the rotational rate from a sequence of quaternions (e.g. the attitude of a vehicle over time), there needs to be a numerical computation of the rates. However, because quaternions are not linear, doing a simple finite differencing:

$$
\begin{equation*}
\dot{\mathbf{q}}=\frac{\mathbf{q}_{2}-\mathbf{q}_{1}}{\Delta t} \tag{C.46}
\end{equation*}
$$

is not sufficient and will introduce errors. The same principles as used in SLERP need to be applied to get the rotational difference between two quaternions, from Eq. C.41.

Taking the logarithm of this rotational difference converts it into an axis-angle form where the rotation can then be divided by time to get what is the body rates:

$$
\begin{equation*}
\frac{1}{2} \underline{\omega}=\frac{\ln (\Delta \mathbf{q})}{\Delta t} \tag{C.47}
\end{equation*}
$$

This method of determining body rates assumes a constant rotation axis, and computes the rate of rotation about that axis. With the angular rate, the quaternion rates can be determined:

$$
\begin{equation*}
\dot{\mathbf{q}}=\frac{1}{2} \mathbf{q}_{1} \otimes \underline{\omega}=\frac{1}{\Delta t} \mathbf{q}_{1} \otimes \ln \left(\mathbf{q}_{1}^{c} \otimes \mathbf{q}_{2}\right) \tag{C.48}
\end{equation*}
$$

## C. 10 Quaternion Integration

To propagate forward dynamics, integrating from given a quaternion rate or body rates, difficulties arise when using Euler or Runge-Kutta integration as the quaternions will lose their unit quaternion property. This issue can be managed by normalising the quaternions on each iteration.

$$
\begin{equation*}
\mathbf{q}_{k+1}=\frac{\dot{\mathbf{q}}_{k} \Delta t+\mathbf{q}_{k}}{\left|\dot{\mathbf{q}}_{k} \Delta t+\mathbf{q}_{k}\right|} \tag{C.49}
\end{equation*}
$$

Alternatively, the body rate can be computed with Eq. C.33, and an approximation made to have a constant angular rate for the period of integration. Multiplying the body rate by the time period, $\Delta t$ gives the forward propagated rotation in rotation vector format. The quaternion exponential transforms this into quaternion form to apply the rotation:

$$
\begin{equation*}
\mathbf{q}_{k+1}=\exp \left(\frac{\omega \Delta t}{2}\right) \otimes \mathbf{q}_{k} \tag{C.50}
\end{equation*}
$$

With this method, the unit quaternion properties are maintained.


## Quaternion Derivation of Differential Flatness Transform

It is assumed that a trajectory is given for the differentially flat derivatives and their inputs: position $(x, y, z)$ up to the 4 th derivative, the quaternion for the z part of the vector, $q_{3}$ up to the 2 nd derivative. i.e. the variables planned are:

$$
\begin{gathered}
\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)} \\
q_{3}, \dot{q_{3}}, \ddot{q_{3}}
\end{gathered}
$$

where the quaternion convention is defined as outlined in Appendix C.
It is assumed in this derivation that a North-East-Up reference frame is used (important for defining the sign of the acceleration due to gravity, $\tilde{g}$ ).

## D. 1 Compute the Thrust Vector

As in previous techniques, the net acceleration, along with gravity gives the thrust, which gives the $z$ body-axis.

First, the thrust vector is given by:

$$
\begin{equation*}
\mathbf{T}=m\left(\ddot{\mathbf{x}}+\tilde{g} \mathbf{z}_{g}\right) \tag{D.1}
\end{equation*}
$$

where $m$ is the mass, and $\mathbf{z}_{g}$ the $z$ global vector $\left([0,0,1]^{T}\right)$. The magnitude of thrust is:

$$
\begin{equation*}
T=m\left\|\ddot{\mathbf{x}}+\tilde{g} \mathbf{z}_{g}\right\| \tag{D.2}
\end{equation*}
$$

## D. 2 Coupling Thrust with Attitude

The thrust is coupled with the attitude, as thrust can only be directed upward in the $z$ body-axis. Hence:

$$
\begin{equation*}
\mathbf{R} \mathbf{z}_{b}=\frac{m}{T}\left(\ddot{\mathbf{x}}+\tilde{g} \mathbf{z}_{g}\right) \tag{D.3}
\end{equation*}
$$

where $\mathbf{R}$ is the rotation matrix describing the orientation. This coupling only reduces two of the degrees of freedom (with the equation only using the third column of the rotation matrix). FOr quaternions, the third column in the rotation matrix (Eq. C.30) results in the following equations:

$$
\begin{align*}
2\left(q_{0} q_{2}+q_{1} q_{3}\right) & =\frac{m}{T} \ddot{x}_{1}  \tag{D.4}\\
2\left(q_{2} q_{3}-q_{0} q_{1}\right) & =\frac{m}{T} \ddot{x}_{2}  \tag{D.5}\\
\left(q_{0}^{2}-q_{1}^{2}\right)-\left(q_{2}^{2}-q_{3}^{2}\right) & =\frac{m}{T}\left(\ddot{x}_{3}+\tilde{g}\right) \tag{D.6}
\end{align*}
$$

This gives us three equations to find the three unknowns, $q_{0}, q_{1}, q_{2}$, but the quaternions are also coupled by requiring a norm of 1 , giving us a fourth equation:

$$
\begin{equation*}
q_{0}^{2}+q_{1}^{2}+q_{2}^{2}+q_{3}^{2}=1 \tag{D.8}
\end{equation*}
$$

Equation D.8, combined with equation D. 6 gives either of the following equations:

$$
\begin{align*}
2 q_{0}^{2}+2 q_{3}^{2}-1 & =\frac{m}{T}\left(\ddot{x}_{3}+g\right)  \tag{D.9}\\
1-2 q_{1}^{2}-2 q_{2}^{2} & =\frac{m}{T}\left(\ddot{x}_{3}+g\right) \tag{D.10}
\end{align*}
$$

Equations D.4, D.5, D. 9 and D. 10 will be used to derive expressions for $q_{0}$, as a function of $q_{3}$, as well as $q_{1}$, and $q_{2}$ as a function of $q_{0}$ and $q_{3}$.

First we take equation D. 4 and rearrange to make $q_{2}$ the subject, then do the same for $q_{1}$, using equation D.5:

$$
\begin{align*}
& q_{2}=\frac{1}{q_{0}}\left(\frac{m}{2 T} \ddot{x}_{1}-q_{1} q_{3}\right)  \tag{D.11}\\
& q_{1}=\frac{1}{q_{0}}\left(-\frac{m}{2 T} \ddot{x}_{2}+q_{2} q_{3}\right) \tag{D.12}
\end{align*}
$$

Substituting one into the other and rearranging to isolate variables, we get expressions for $q_{1}$ and $q_{2}$ in terms of $q_{3}$ and $q_{0}:$

$$
\begin{align*}
q_{1} & =\left(\frac{m}{2 T}\right)\left(\frac{1}{\left(q_{0}^{2}+q_{3}^{2}\right)}\right)\left(q_{3} \ddot{x}_{1}-q_{0} \ddot{x}_{2}\right)  \tag{D.13}\\
q_{2} & =\left(\frac{m}{2 T}\right)\left(\frac{1}{\left(q_{0}^{2}+q_{3}^{2}\right)}\right)\left(q_{0} \ddot{x}_{1}+q_{3} \ddot{x}_{2}\right) \tag{D.14}
\end{align*}
$$

There are two methods to get $q_{0}$. First, using equation D.9, it can simply be rearranged to give $q_{0}$ as a function of $q_{3}$.

$$
\begin{equation*}
\left.q_{0}=\sqrt{[ }\right] \frac{1}{2}\left(1+\frac{m}{T}\left(\ddot{x}_{3}+g\right)\right)-q_{3}^{2} \tag{D.15}
\end{equation*}
$$

The second derivation uses equation D.10, and substitutes the equations for $q_{1}$ and $q_{2}$, giving:

$$
\begin{equation*}
\left.q_{0}=\sqrt{[ }\right] \frac{\ddot{x}_{1}^{2}+\ddot{x}_{2}^{2}}{2\left(\frac{T}{m}\right)\left[\frac{T}{m}-\left(\ddot{x}_{3}+g\right)\right]} \tag{D.16}
\end{equation*}
$$

A choice is made to select the positive square root in each, taking the option of a positive constant term of the two equivalent quaternion representations $(\mathbf{q}=-\mathbf{q})$.

With each of the quaternion terms, the full orientation is known, and the derivation of the remaining states is very similar to the standard derivation, except for the use of quaternion rates, rather than yaw rates (details are not presented here).

## D. 3 Singularities

It is important to note the singularities inherent in these formulations:

1. In equations D. 13 and D. 14 if $q_{0}^{2}+q_{3}^{2}=0$. Which would be the case for pure rotations about the $x$ or $y$ axes of 180 degrees.
2. For the different $q_{0}$ options:
a) In equation D.15, and D. 16 when $q_{3}$ is large, and the terms under the square root become negative.
b) In equation D. 16 when there is no horizontal acceleration $\left(\frac{T}{m}-\left(\ddot{x}_{3}+g\right)=0\right)$

An approach to address each of these singularities is to plan both $q_{0}$ and $q_{3}$, yet this has difficulties in describing the desired orientation.

## D. 4 Angular Rates

The process of extracting the angular rates is very similar to the process outlined by [26, 140], using jerk. Refer to their papers for more detail on the derivation. It is repeated here for completeness. First, the quaternion is used to get the rotation matrix (using equation C.30), where each column gives the body-axes:

$$
\mathbf{R}=\left[\begin{array}{lll}
\mathbf{x}_{b} & \mathbf{y}_{b} & \mathbf{z}_{b} \tag{D.17}
\end{array}\right]
$$

Here $\mathbf{x}_{b}$ is referring to the $x$ body-axis. The pitch and roll rates ( $\omega_{1}$ and $\omega_{2}$ ) can then be extracted using:

$$
\begin{align*}
\mathbf{h}_{w} & =\frac{m}{T}\left(\mathbf{x}^{(3)}-\left(\mathbf{z}_{b} \cdot \mathbf{x}^{(3)}\right) \mathbf{z}_{b}\right)  \tag{D.18}\\
\omega_{1} & =-\mathbf{h}_{w} \cdot \mathbf{y}_{b}  \tag{D.19}\\
\omega_{2} & =\mathbf{h}_{w} \cdot \mathbf{x}_{b} \tag{D.20}
\end{align*}
$$

The extraction of the yaw rate differs from [26, 139], as it is the rate of $q_{3}$ that is planned. The quaternion rates are given by the quaternion product, Eq. C.31. Taking the last row of the rates equation, and rearranging to make $\omega_{3}$ the subject gives:

$$
\begin{equation*}
\omega_{3}=\frac{1}{q_{0}}\left(2 \dot{q}_{3}+\omega_{1} q_{2}-\omega_{2} q_{1}\right) \tag{D.21}
\end{equation*}
$$

This equation does have a singularity when $q_{0}=0$ which would occur when there are 180 -degree rotations.

## D. 5 Angular Acceleration

Angular acceleration proceeds similarly to angular velocity, using the snap. Following [26, 139], first the utility vector $\mathbf{h}_{\alpha}$ is computed:

$$
\begin{align*}
\mathbf{h}_{\alpha}= & \frac{m}{T}\left[\boldsymbol{x}^{(4)}-\left(\mathbf{z}_{b} \cdot \boldsymbol{x}^{(4)}\right) \mathbf{z}_{b}\right. \\
& +\frac{T}{m}\left(\mathbf{z}_{b} \cdot\left(\omega \times \omega \times \mathbf{z}_{b}\right)\right) \mathbf{z}_{b}  \tag{D.22}\\
& -\frac{T}{m} \omega \times \omega \times \mathbf{z}_{b} \\
& \left.\left.-2 \omega \times\left(\mathbf{z}_{b} \cdot \boldsymbol{x}^{(3)}\right) \mathbf{z}_{b}\right)\right]
\end{align*}
$$

Which then gives the first two rotational accelerations:

$$
\begin{align*}
& \dot{\omega}_{1}=-\mathbf{h}_{\alpha} \cdot \mathbf{y}_{b}  \tag{D.23}\\
& \dot{\omega}_{2}=\mathbf{h}_{\alpha} \cdot \boldsymbol{x}_{b} \tag{D.24}
\end{align*}
$$

Deriving the third angular acceleration uses the quaternion acceleration:

$$
\begin{equation*}
\ddot{\mathbf{q}}=\frac{1}{2} \mathbf{q} \otimes \dot{\omega}-\mathbf{q} \otimes \dot{\mathbf{q}}^{c} \otimes \dot{\mathbf{q}}=\frac{1}{2} \mathbf{q} \otimes \dot{\omega}-\left(\dot{\mathbf{q}}^{T} \otimes \dot{\mathbf{q}}\right) \otimes \mathbf{q} \tag{D.25}
\end{equation*}
$$

We take the row corresponding to the yaw acceleration, and rearrange to give:

$$
\begin{equation*}
\dot{\omega}=\frac{1}{q_{0}}\left[2 \ddot{q}_{3}-q_{1} \dot{\omega}_{2}+q_{2} \dot{\omega}_{1}+2\left(\dot{\mathbf{q}}^{T} \otimes \dot{\mathbf{q}}\right) q_{3}\right] \tag{D.26}
\end{equation*}
$$

The quaternion rates, $\dot{\mathbf{q}}$ can be computed using equation C.31. Note that the same singularity exists here if $q_{0}=0$.

With the rotational acceleration, the torques and motor RPM are extracted as for all other methods, as described in Section 5.1.1.4.

## List of Acronyms and Abbreviations

| Acronym | Meaning |
| :--- | :--- |
| ASTRO | Admissible Subspace Trajectory Optimiser |
| ASTRO-C | ASTRO with corridor constraints |
| ASTRO-CE | ASTRO with corridor constraints and ESDF feasibility checks |
| ASTRO-E | ASTRO with ESDF constraints |
| BA | Bundle Adjustment |
| BCM | Bayesian Committee Machine |
| CRM | Confidence Rich Mapping |
| DSP | Digital Signal Processor |
| EKF | Extended Kalman Filter |
| ESC | Electronic Speed Controller |
| ESDF | Euclidean Signed Distance Field |
| GCS | Ground Control Station |
| GP | Gaussian Process |
| GPIS | Gaussian Process Implicit Surfaces |
| GPOM | Gaussian Process Occupancy Maps |
| GPS | Global Positioning System |
| ICP | Iterative Closest Point |
| IMU | Inertial Measurement Unit |
| LQR | Linear Quadratic Regulator |
| MEKF | Multiplicative Extended Kalman Filter |
| MIT | Massachusetts Institute of Technology |
| NURBS | Non-Uniform Rational B-Splines |
| NURBSLAM | Non-Uniform Rational B-Splines Localisation And Mapping |
| ORB-SLAM | Oriented FAST, Rotated BRIEF Simultaneous Localisation And Mapping |
| PCL | Point Cloud Library |
| PDFF | Proportional, Derivative, Feed-Forward controller |
| PID | Proportional, Integral, Derivative controller |
| PnP | Perspective from n Points |
| RANSAC | Random Sample Consensus |
| RDP | Ramer-Douglas-Peucker algorithm |
| RGBD | Red, Green, Blue, Depth (colour and depth images) |
| RMS/RMSE | Root-Mean-Square/Root-Mean-Square Error |
| ROS | Robotic Operating System |
| RPM | Revolutions Per Minute |
| RRT | Randomly-exploring Random Trees |
| SLAM | Simultaneous Localisation And Mapping |
| SLAM | Simultaneous Localisation And Mapping |
| SPHERES | Synchronized Position Hold, Engage, Reorient Experimental Satellites |


| Acronym | Meaning |
| :--- | :--- |
| SSL | Space Systems Laboratory |
| TACO | Tube And Cube constrained Optimiser |
| TAN | Terrain Aided Navigation |
| TSDF | Truncated Signed Distance Field |
| UAV | Unmanned Aerial Vehicle |
| UNCO | UNConstrained Optimiser |
| V-SLAM | Visual Simultaneous Localisation And Mapping |
| VIO | Visual Inertial Odometry |
| VO | Visual Odometry |

## List of Symbols

| Term | Meaning | Area |
| :---: | :---: | :---: |
| A | Shape matrix for ellipsoids and prisms | Traj. Opt. |
| $a$ | Time scaling factor for ASTRO | Traj. Opt. |
| $\alpha$ | Optimisation step size | Traj. Opt. |
| $A_{\text {max }}$ | Maximum acceleration for TACO | Traj. Opt. |
| $a_{u}$ | Acceleration uncertainty for dynamic obstacles | Traj. Opt. |
| $\beta$ | Angular rotation for pitch and roll differential flatness transformation | Quad. Control |
| $B_{i}^{r}$ | $i$ th basis blending function for NURBS of degree $r$ | SLAM + 3D Map. |
| $\boldsymbol{B}$ | Matrix of basis blending function for NURBS | SLAM + 3D Map. |
| $\overline{\mathbf{C}}$ | Matrix of coefficients for one polynomial segment in all dimensions | Traj. Opt. |
| $\boldsymbol{C}_{\perp}$ | Component of polynomial coefficients with no influence on boundary conditions | Traj. Opt. |
| $\mathbf{C}_{i}$ | Stack of polynomial coefficients for the $i$ th dimension | Traj. Opt. |
| $\mathbf{C}_{i}^{\prime}$ | Coefficients fit to a trajectory for the $i$ th dimension | Traj. Opt. |
| $C_{i k}$ | Polynomial coefficient for the $i$ th dimension for the $k$ th degree basis polynomial | Traj. Opt. |
| $\boldsymbol{C}_{\\|}$ | Component of polynomial coefficients that does influence boundary conditions | Traj. Opt. |
| $\underline{C}_{i}$ | Stack of polynomial coefficients for multiple segments in the $i$ th dimension | Traj. Opt. |
| D | Data points for NURBS surface fitting | SLAM + 3D Map. |
| d | Derivatives at waypoints, for use in snap optimisation | Traj. Opt. |
| $\mathbf{D}_{h}$ | Propellor drag vector | Quad. Control |
| $\mathbf{D}_{p}$ | Parasitic drag vector | Quad. Control |
| $\Delta t$ | Timestep | SLAM + 3D Map. |
| e | Rotation axis unit vector | - |
| E | Cross product matrix for rotation axis $\mathbf{e}$ | Traj. Opt. |
| $\mathbf{e}_{q}$ | Attitude error vector | SLAM + 3D Map. |
| $\epsilon$ | Setting for path discretisation algorithm RDP | Traj, Opt. |
| $\eta$ | Scalar component of Rodrigues parameters | SLAM + 3D Map. |
| $f_{B C}$ | Boundary conditions function | Traj. Opt. |
| $f_{c_{j}}$ | Constraint cost function | Traj. Opt. |
| $f_{j}$ | $j$ th cost function | Traj. Opt. |
| $f_{s}$ | Trajectory cost function | Traj. Opt. |
| $\Gamma$ | Time rescaling factor | Traj. Opt. |
| $\gamma$ | Second angle for use in differential flatness transformations | Quad. Control |
| $\boldsymbol{H}$ | Approximate Hessian matrix | Traj. Opt. |
| $h$ | Time per segment for TACO | Traj. Opt. |
| $\mathbf{h}_{\alpha}$ | Working vector for differential flatness transformation | Quad. Control |


| Term | Meaning | Area |
| :---: | :---: | :---: |
| $\mathbf{h}_{\omega}$ | Working vector for differential flatness transformation | Quad. Control |
| I | Identity matrix | - |
| $\bar{I}$ | Moment of Inertia matrix | Quad. Control |
| J | Process model Jacobian for EKF | SLAM + 3D Map. |
| J | Augmented cost function in Traj. Opt. | Traj. Opt. |
| $\mathbf{J}_{h}$ | Observation model Jacobian for EKF | SLAM + 3D Map. |
| $J_{m}$ | Multiple segment augmented cost function | Traj. Opt. |
| $J_{t}$ | Time optimisation cost function | Traj. Opt. |
| K | Kalman gain | SLAM + 3D Map. |
| к | Weighting in GPIS kernel | SLAM + 3D Map. |
| $K_{d}$ | D gain on velocity error | Quad. Control |
| $k_{f}$ | Thrust coefficient for quadrotor dynamics | Quad. Control |
| $K_{f f}$ | Feed-forward gain on acceleration | Quad. Control |
| $K_{j}$ | Weighting for $j$ th cost function | Traj. Opt. |
| $K_{\omega}$ | D gain on angular velocity error | Quad. Control |
| $K_{p}$ | $P$ gain on position error | Quad. Control |
| $K_{q}$ | P gain on attitude error | Quad. Control |
| $L$ | Moment are for quadrotor dynamics | Traj. Opt. |
| $l$ | Spacing between waypoints for TACO | Traj. Opt. |
| $l_{\text {max }}$ | Size of free-space bounds | Traj. Opt. |
| $m, m_{s}, m_{t}$ | NURBS number of data points (in $s$ and $t$ parametric directions) | SLAM + 3D Map. |
| $\mu$ | Custom constraint weighting parameter | Traj. Opt. |
| $n, n_{s}, n_{t}$ | NURBS number of control points (in $s$ and $t$ parametric directions) | SLAM + 3D Map. |
| $n_{f}$ | number of samples for fitting a polynomial for replanning | Traj. Opt. |
| $N_{i}$ | Number of coefficients in the $i$ th dimension | SLAM + 3D Map. |
| $n_{0}$ | ASTRO number of constraints | Traj. Opt. |
| $n_{\text {seg }}$ | Number of segments | Traj. Opt. |
| $\mathbf{n}_{s t}$ | Surface normal | SLAM + 3D Map. |
| $v$ | Custom constraint weighting parameter | Traj. Opt. |
| $\otimes$ | quaternion product | - |
| $\omega$ | Rotational velocity | Quad. Control |
| $\Omega$ | Rotor rotational velocity | Quad. Control |
| $p$ | Degree of polynomials in NURBS | SLAM + 3D Map. |
| $\boldsymbol{P}_{B C}$ | Legendre polynomial matrix for boundary conditions | Traj. Opt. |
| $\overline{\boldsymbol{P}}_{B C}$ | Multiple-segment boundary-condition matrix of Legendre polynomial coefficients | Traj. Opt. |
| $\boldsymbol{P}_{C C}$ | Matrix of Legendre polynomials for continuity constraints | Traj. Opt. |
| $\underline{\boldsymbol{P}_{C}}$ | Combined boundary and continuity constraint matrix of Legendre polynomials for multiple segments and dimension $i$ | Traj. Opt. |
| $\phi$ | Angle of rotation for rotation vector attitude representation | - |


| Term | Meaning | Area |
| :--- | :--- | :--- |
| $\boldsymbol{P}_{\text {int }}$ | Matrix of Legendre polynomial integrals | Traj. Opt. |
| $P_{k}$ | Legendre Polynomial coefficients of degree $k$ | Traj. Opt. |
| $\boldsymbol{P}_{L}\left(t^{\prime}\right)$ | Matrix of basis polynomials across all derivatives for normalised | Traj. Opt. |
|  | time $t^{\prime}$ |  |
| $\psi$ | yaw angle | - |
| $\psi_{s p}$ | Yaw set-point | Quad. Control |
| $\mathbf{q}$ | Quaternion | - |
| $q$ | The highest derivative considered in trajectory optimisaiton | Traj. Opt. |
| $\mathbf{q}_{e}$ | Error quaternion | - |
| $\mathbf{Q}_{n}$ | Process noise in EKF | SLAM + 3D Map. |
| $\mathbf{q}_{s p}$ | Quaternion set-point | Quad. Control |
| $\mathbf{R}$ | Rotation matrix | - |
| $r$ | derivative iterator | Traj. Opt. |
| $r_{1}, r_{2}, r_{3}$ | Axes sizes for ellipsoids and prisms | Traj. Opt. |
| $\boldsymbol{r} \boldsymbol{h} \boldsymbol{o}$ | NURBS Control points | SLAM + 3D Map. |
| $\mathbf{R}_{n}$ | Observation uncertainty matrix | SLAM + 3D Map. |
| $s$ | NURBS parameter value | SLAM + 3D Map. |
| $\Sigma$ | Covariance matrix | - |
| $\sigma_{1}, \sigma_{2}, \sigma_{3}$ | Optimisation convergence criteria | Traj. Opt. |
| $\hat{\Sigma}$ | Predicted Covariance | SLAM + 3D Map. |
| $T$ | Thrust magnitude | Quad. Control |
| $\mathbf{T}$ | Thrust vector | Quad. Control |
| $t_{0}$ | Starting time for trajectory planning Opt. |  |
| $t$ | NURBS parameter value | Traj. Opt. |
| $t$ | time | SLAM + 3D Map. |
| $\boldsymbol{\tau}$ | Moment vector | Traj. Opt. |
| $t_{f}$ | Ending time for trajectory planning | Quad. Control |
| $t_{u}$ | Normalised time | Traj. Opt. |
| $\bar{t}_{\text {obs }}$ | Translational component of an observed transformation | Traj. Opt. |
| $\mathbf{T}_{s p}$ | Thrust set-point | SLAM + 3D Map. |
| $u_{k}$ | NURBS knot in s direction | Quad. Control |
| $\bar{u}$ | NURBS knot vector in s direction | SLAM + 3D Map. |
| $v_{k}$ | NURBS knot in t direction | SLAM + 3D Map. |
| $\bar{v}$ | NURBS knot vector in t direction | SLAM + 3D Map. |
| $\boldsymbol{v}_{c}$ | Velocity of obstacle | SLAM + 3D Map. |
| $\mathbf{v}_{e}$ | Error rotation vector | Traj. Opt. |
| $V_{\text {max }}$ | Maximum velocity for TACO | - |
| $\mathbf{v}_{s t}$ | Vector from nearest surface point to query point for NURBS signed | SLAM + 3D Map. |
|  | distance evaluation |  |


| Term | Meaning | Area |
| :---: | :---: | :---: |
| W | Cost function for snap optimisation | Traj. Opt. |
| $w_{0-5}$ | Tuning parameters for NURBS alignment observation noise matrices. | SLAM + 3D Map. |
| $w_{\text {io }}$ | In or out value for constraints | Traj. Opt. |
| $W_{j}$ | Constraint cost for $j$ th constraint | Traj. Opt. |
| $W_{t}$ | Time penalty in optimisation | Traj. Opt. |
| $\boldsymbol{x}$ | Position vector | - |
| $\dot{x}$ | Velocity vector | - |
| $\dot{\boldsymbol{x}}$ | Acceleration vector | - |
| $\boldsymbol{x}_{\text {b }}$ | x body axis | - |
| $\overline{\boldsymbol{x}}$ | Vector with all derivatives for $x, y, z$ | Traj. Opt. |
| $\boldsymbol{X}_{B C}$ | Boundary conditions vector | Traj. Opt. |
| $\overline{\boldsymbol{X}}_{B C}$ | Multiple-segment boundary conditions | Traj. Opt. |
| $\breve{x}, \breve{y}, \breve{z}$ | Cost components for rectangular prism constraint | Traj. Opt. |
| $\boldsymbol{x}_{c}$ | Desired heading, x axis | Quad. Control |
| $\boldsymbol{x}_{c_{s a}}$ | Desired heading, x axis, rotated by a second angle | Quad. Control |
| $\underline{\underline{\mathbf{x}_{C}}}$ | Combined boundary and continuity constraints for multiple segments and dimension $i$ | Traj. Opt. |
| $\mathbf{x}$ | EKF State | SLAM + 3D Map. |
| $\xi$ | Derivative of trajectory that is parameterised by Legendre Polynomials | Traj. Opt. |
| $x_{i}^{(p)}$ | The $p$ th derivative in the $i$ th dimension | Traj. Opt. |
| $\boldsymbol{x}_{s p}$ | Position set-point | Quad. Control |
| $\tilde{\boldsymbol{x}}$ | Position offset from the centre of a constraint | Traj. Opt. |
| $x_{u}$ | Position uncertainty for dynamic obstacles | Traj. Opt. |
| $\boldsymbol{Y}$ | Matrix of scaled data points for NURBS fitting | SLAM + 3D Map. |
| $\boldsymbol{y}_{b}$ | y body axis | - |
| $\boldsymbol{y}_{\text {c }}$ | Desired heading, y axis | Quad. Control |
| $\boldsymbol{z}_{b}$ | z body axis | - |
| $\zeta$ | Control points on a new surface | SLAM + 3D Map. |
| $\boldsymbol{z}_{g}$ | Global z vector | - |
| $\mathbf{z}_{k}$ | Observation vector at the $k$ th timestep | SLAM + 3D Map. |


| Superscript | Meaning | Area |
| :--- | :--- | :--- |
| + | Moore-Penrose Pseudo-Inverse | - |
| $(i)$ | The $i$ th derivative | Traj. Opt. |
| $f_{(i)}$ | The $i$ th integral | Traj. Opt. |
|  |  |  |
| Subscript | Meaning | Area |
| $b$ | body frame | - |
| $g$ | Global frame | - |
| $m$ | Multiple segment | Traj. Opt. |

## BIBLIOGRAPHY

[1] Abbeel, P., Coates, A., and NG, A. Y.
Autonomous helicopter aerobatics through apprenticeship learning.
The International Journal of Robotics Research 29, 13 (2010), 1608-1639.
[2] Abdou, E., Foweraker, N., Phillips, E., Zhang, X., Zochowski, Y., Coen, M., McHenry, N., Morrell, B., Chamitoff, G., and Wu, X.

Spacecraft: Endeavouring towards a satellite virtual reality for mission operation evaluation.
In 68th International Astronautical Congress (IAC) (Adelaide, Australia, Sept. 2017), no. IAC-17F1.2.3.
[3] Agha-Mohammadi, A.-A., Heiden, E., Hausman, K., and Sukhatme, G.
Confidence-rich grid mapping.
In Proc. Int. Symp. Robot. Res. (2017), pp. 1-19.
[4] Ait-Jellal, R., And Zell, A.
Outdoor obstacle avoidance based on hybrid visual stereo SLAM for an autonomous quadrotor mav.

In 2017 European Conference on Mobile Robots (ECMR) (Sept. 2017), pp. 1-8.
[5] Allen, R. E., and Pavone, M.
A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. PhD thesis, Stanford University, 2016.
[6] Alonso Ayuso, A., Escudero, L., and Martin Campo, F.
Collision Avoidance in Air Traffic Management: A Mixed-Integer Linear Optimization Approach.
Intelligent Transportation Systems, IEEE Transactions on 12, 1 (Mar. 2011), 47-57.
[7] Amenta, N., Choi, S., And Kolluri, R. K.
The power crust, unions of balls, and the medial axis transform.
Computational Geometry 19, 2-3 (2001), 127-153.
[8] Andersen, M. S., Dahl, J., and Vandenberghe, L.
Cvxopt: A python package for convex optimization, version 1.1. 6.
Available at cuxopt. org 54 (2013).
[9] ArmiJo, L.
Minimization of functions having lipschitz continuous first partial derivatives.
Pacific Journal of mathematics 16, 1 (1966), 1-3.
[10] Arthur Richards, Tom Schouwenaars, Jonathan P. How, and Eric Feron.
Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming.
Journal of Guidance, Control, and Dynamics 25, 4 (July 2002), 755-764.
[11] Baldini, F., Bandyopadhyay, S., Foust, R., Chung, S.-J., Rahmani, A., de la Croix, J.-P., Bacula, A., Chilan, C. M., and Hadaegh, F. Y.
Fast motion planning for agile space systems with multiple obstacles.
In AIAA /AAS Astrodynamics Specialist Conference (2016), p. 5683.
[12] Bangura, M.
Aerodynamics and Control of Quadrotors.
PhD thesis, The Australian National University, 2017.
[13] Barry, A. J., Florence, P. R., and Tedrake, R.
High-speed autonomous obstacle avoidance with pushbroom stereo.
Journal of Field Robotics 35, 1 (2018), 52-68.
[14] Bellingham, J. S., Tillerson, M., Alighanbari, M., and How, J. P.
Cooperative Path Planning for Multiple UAVs in Dynamic and Uncertain Environments.
In Decision and Control, 2002, Proceedings of the 41 ${ }^{\text {st }}$ IEEE Conference on (2002), vol. 3, IEEE, pp. 2816-2822.
[15] Belongie, S.
Rodrigues rotation formula.
From MathWorld-A Wolfram Web Resource, created by Eric W. Weisstein. http:/ /mathworld. wolfram. com / RodriguesRotationFormula. html (1999).
[16] Beul, M., Droeschel, D., Nieuwenhuisen, M., Quenzel, J., Houben, S., and Behnke, S.

Fast autonomous flight in warehouses for inventory applications.
IEEE Robotics and Automation Letters 3, 4 (2018), 3121-3128.
[17] Biasotti, S., Giorgi, D., Marini, S., Spagnuolo, M., and Falcidieno, B.
A comparison framework for 3d object classification methods.
In International Workshop on Multimedia Content Representation, Classification and Security (2006), Springer, pp. 314-321.
[18] Blackmore, L., Li, H., and Williams, B.
A Probabilistic Approach to Optimal Robust Path Planning with Obstacles.
In American Control Conference, 2006 (2006), IEEE, pp. 7-pp.
[19] Bloesch, M., Omari, S., Hutter, M., and Siegwart, R.
Robust visual inertial odometry using a direct EKF-based approach.
In Intelligent Robots and Systems (IROS), 2015 IEEE / RSJ International Conference on (2015), IEEE, pp. 298-304.
[20] Breger, L. S., and How, J. P.
Safe Trajectories for Autonomous Rendezvous Of Spacecraft.
Journal of Guidance, Control, and Dynamics 31, 5 (2008), 1478-1489.
[21] Brockers, R., Fragoso, A., and Matthies, L.
Stereo vision-based obstacle avoidance for micro air vehicles using an egocylindrical image space representation.
In Micro-and Nanotechnology Sensors, Systems, and Applications VIII (2016), vol. 9836, International Society for Optics and Photonics, p. 98361R.
[22] Brook, P., Ciocarlie, M., and Hsiao, K.
Collaborative grasp planning with multiple object representations.
In Robotics and Automation (ICRA), 2011 IEEE International Conference on (2011), IEEE, pp. 2851-2858.
[23] Brooks, A., and Bailey, T.
HybridSLAM: Combining FastSLAM and EKF-SLAM for Reliable Mapping.
In Algorithmic Foundation of Robotics VIII. Springer, 2009, pp. 647-661.
[24] Broyden, C. G.
The Convergence of a Class of Double-Rank Minimization Algorithms.
Journal of the Institute of Mathematics and its Applications 6 (1970), 76-90.
[25] Bruce, J., and Veloso, M.
Real-Time Randomized Path Planning for Robot Navigation.
In Intelligent Robots and Systems, 2002. IEEE / RSJ International Conference on (2002), vol. 3, IEEE, pp. 2383-2388.
[26] Bry, A., Richter, C., Bachrach, A., and Roy, N.
Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments.
The International Journal of Robotics Research 34, 7 (2015), 969-1002.
[27] Buch, A. G., Kraft, D., Kamarainen, J.-K., Petersen, H. G., and Krüger, n.
Pose estimation using local structure-specific shape and appearance context.
In Robotics and Automation (ICRA), 2013 IEEE International Conference on (2013), IEEE, pp. 2080-2087.
[28] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R.
The EuRoC micro aerial vehicle datasets.
The International Journal of Robotics Research 35, 10 (2016), 1157-1163.
[29] Byatt, D., Coope, I. D., and Price, C. J.
Effect of limited precision on the BFGS quasi-Newton algorithm.
ANZIAM Journal 45, 0 (2004), 283-295.
[30] Cafieri, S., and Durand, N.
Aircraft Deconfliction with Speed Regulation: New Models from Mixed-Integer Optimization. Journal of Global Optimization 58, 4 (2014), 613-629.
[31] Calonder, M.
EKF SLAM vs. FastSLAM A Comparison.
Tech. rep., 2006.
[32] Campos-Macías, L., Gómez-Gutiérrez, D., Aldana-López, R., de la Guardia, R., and Parra-Vilchis, J. I.
A hybrid method for online trajectory planning of mobile robots in cluttered environments.
IEEE Robotics and Automation Letters 2, 2 (2017), 935-942.
[33] Chamitoff, G. E.
Autonomous Guidance for the Recovery and Landing of a Remotely Piloted Vehicle.
In IFAC Aerospace Control Conference (Palo Alto, California, 1994).
[34] Chamitoff, G. E., Saenz Otero, A., Katz, J. G., and Ulrich, S.
Admissible Subspace TRajectory Optimizer (ASTRO) for Autonomous Robot Operations on the Space Station.
In AIAA Guidance, Navigation, and Control Conference (2014), AIAA Reston, VA, pp. 1-17.
[35] Chamitoff, G. E., Saenz-Otero, A., Katz, J. G., Ulrich, S., Morrell, B. J., and Gibbens, P. W.

Real-time maneuver optimization of space-based robots in a dynamic environment: Theory and on-orbit experiments.
Acta Astronautica 142 (2018), 170-183.
[36] Chang, D. E., Shadden, S. C., Marsden, J. E., and Olfati Saber, R.
Collision Avoidance for Multiple Agent Systems.
In Proceedings of the $42^{\text {nd }}$ IEEE Conference on Decision and Control (Dec. 2003), IEEE.
[37] Chang, W.
Surface reconstruction from points.
Department of Computer Science and Engineering, University of California, San Diego, 2008.
[38] Chang, Y.-C., Kao, J.-H., Pinilla, J., Dong, J., and Prinz, F.
Medial axis transform (MAT) of general 2D shapes and 3D polyhedra for engineering applications.
In Geometric Modelling. Springer, 2001, pp. 37-52.
[39] Chiodini, S., Reid, R., Hockman, B., Nesnas, I., Debei, S., and Pavone, M.
Robust visual localization for hopping rovers on small bodies.
In International Conference on Robotics and Automation (2018), IEEE.
[40] Clark, C. M.
Probabilistic Road Map Sampling Strategies for Multi-Robot Motion Planning.

Robotics and Autonomous Systems 53, 3 (2005), 244-264.
[41] Concha, A., and Civera, J.
DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence.
In Intelligent Robots and Systems (IROS), 2015 IEEE / RSJ International Conference on (2015), IEEE, pp. 5686-5693.
[42] Conte, G., And Doherty, P.
An integrated uav navigation system based on aerial image matching.
In Aerospace Conference, 2008 IEEE (2008), IEEE, pp. 1-10.
[43] Conway, B. A.
A survey of methods available for the numerical optimization of continuous dynamic systems.
Journal of Optimization Theory and Applications 152, 2 (2012), 271-306.
[44] CsÁkÁny, P., and Wallace, A. M.
Representation and classification of 3-d objects.
IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 33, 4 (2003), 638-647.
[45] Dantam, N.
Quaternion computation.
Georgia Institute of Technology, Institute for Robotics and Intelligent Machines (2014).
[46] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O.
MonoSLAM: Real-time single camera SLAM.
Pattern Analysis and Machine Intelligence, IEEE Transactions on 29, 6 (2007), 1052-1067.
[47] Dellaert, F.
Factor graphs and gtsam: A hands-on introduction.
Tech. rep., Georgia Institute of Technology, 2012.
[48] Delmerico, J., and Scaramuzza, D.
A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots.
Memory 10 (2018), 20.
[49] Diebel, J.
Representing attitude: Euler angles, unit quaternions, and rotation vectors.
Matrix 58, 15-16 (2006), 1-35.
[50] Doherty, K., Wang, J., and Englot, B.
Probabilistic map fusion for fast, incremental occupancy mapping with 3D hilbert maps.
In Robotics and Automation (ICRA), 2016 IEEE International Conference on (2016), IEEE, pp. 1011-1018.
[51] Dragiev, S., Toussaint, M., and Gienger, M.
Gaussian process implicit surfaces for shape estimation and grasping.

In Robotics and Automation (ICRA), 2011 IEEE International Conference on (2011), IEEE, pp. 2845-2850.
[52] Droeschel, D., Nieuwenhuisen, M., Beul, M., Holz, D., Stückler, J., and Behnke, S. Multilayered mapping and navigation for autonomous micro aerial vehicles.
Journal of Field Robotics 33, 4 (2016), 451-475.
[53] Eggert, D. W., Lorusso, A., and Fisher, R. B.
Estimating 3-D rigid body transformations: a comparison of four major algorithms.
Machine vision and applications 9, 5-6 (1997), 272-290.
[54] Ehang.
Drone formation flight, 2019.
[55] Engel, J., Koltun, V., and Cremers, D.
Direct sparse odometry.
IEEE transactions on pattern analysis and machine intelligence 40, 3 (2018), 611-625.
[56] Engel, J., Schöps, T., and Cremers, D.
LSD-SLAM: Large-scale direct monocular SLAM.
In European Conference on Computer Vision (2014), Springer, pp. 834-849.
[57] Engel, J., Stückler, J., And Cremers, D.
Large-scale direct SLAM with stereo cameras.
In Intelligent Robots and Systems (IROS), 2015 IEEE / RSJ International Conference on (2015), IEEE, pp. 1935-1942.
[58] Engel, J., Usenko, V., and Cremers, D.
A photometrically calibrated benchmark for monocular visual odometry.
arXiv preprint arXiv:1607.02555 (2016).
[59] Epic Games, Inc.
Unreal engine.
Online software, May 2018.
https://www.unrealengine.com/en-US/what-is-unreal-engine-4.
[60] Eren, U., Açikmese, B., and Scharf, D. P.
A Mixed Integer Convex Programming Approach to Constrained Attitude Guidance.
1120-1126.
[61] Faessler, M., Fontana, F., Forster, C., and Scaramuzza, D.
Automatic re-initialization and failure recovery for aggressive flight with a monocular visionbased quadrotor.
In Robotics and Automation (ICRA), 2015 IEEE International Conference on (2015), IEEE, pp. 1722-1729.
[62] Falanga, D., Mueggler, E., Faessler, M., and Scaramuzza, D.
Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision.
In Proc. of the IEEE International Conference on Robotics and Automation (ICRA) (2017).
[63] Fang, Z., Yang, S., Jain, S., Dubey, G., Roth, S., Maeta, S., Nuske, S., Zhang, Y., and Scherer, S.
Robust autonomous flight in constrained and visually degraded shipboard environments.
Journal of Field Robotics 34, 1 (2017), 25-52.
[64] Fiorini, P., and Shiller, Z.
Motion planning in dynamic environments using velocity obstacles.
The International Journal of Robotics Research 17, 7 (1998), 760-772.
[65] Fischler, M. A., and Bolles, R. C.
Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.
Commun. ACM 24, 6 (June 1981), 381-395.
[66] Fletcher, R.
A New Approach to Variable Metric Algorithms.
Computer Journal 13 (1970), 317-322.
[67] Forster, C., Pizzoli, M., and Scaramuzza, D.
SVO: Fast semi-direct monocular visual odometry.
In Robotics and Automation (ICRA), 2014 IEEE International Conference on (2014), IEEE, pp. 1522.
[68] Forster, C., Zhang, Z., Gassner, M., Werlberger, M., and Scaramuzza, D. SVO: Semidirect visual odometry for monocular and multicamera systems.
IEEE Transactions on Robotics 33, 2 (2017), 249-265.
[69] Fox, D., Burgard, W., and Thrun, S.
The dynamic window approach to collision avoidance.
IEEE Robotics \& Automation Magazine 4, 1 (1997), 23-33.
[70] Fraundorfer, F., and Scaramuzza, D.
Visual odometry: Part II: Matching, robustness, optimization, and applications.
IEEE Robotics \& Automation Magazine 19, 2 (2012), 78-90.
[71] Frazzoli, E., Dahleh, M. A., and Feron, E.
A hybrid control architecture for aggressive maneuvering of autonomous helicopters.
In Decision and Control, 1999. Proceedings of the 38 ${ }^{\text {th }}$ IEEE Conference on (1999), vol. 3, IEEE, pp. 2471-2476.
[72] Furrer, F., Burri, M., Achtelik, M., and Siegwart, R.
RotorsS, a modular gazebo mav simulator framework.
In Robot Operating System (ROS). Springer, 2016, pp. 595-625.
[73] Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R.
Towards internet-scale multi-view stereo.
In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on (2010), IEEE, pp. 1434-1441.
[74] Gavrilets, V., Mettler, B., and Feron, E.
Human-inspired control logic for automated maneuvering of miniature helicopter.
Journal of Guidance Control and Dynamics 27, 5 (2004), 752-759.
[75] Ge, S. S., and Cui, Y. J.
Dynamic motion planning for mobile robots using potential field method.
Autonomous robots 13, 3 (2002), 207-222.
[76] Geiger, A., Lenz, P., and Urtasun, R.
Are we ready for autonomous driving? The kitti vision benchmark suite.
In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on (2012), IEEE, pp. 3354-3361.
[77] Gerardo Castro, M. P., Peynot, T., and Ramos, F.
Laser-radar data fusion with gaussian process implicit surfaces.
In Field and Service Robotics (2015), Springer, pp. 289-302.
[78] Giesen, J., Miklos, B., Pauly, M., and Wormser, C.
The scale axis transform.
In Proceedings of the twenty-fifth annual symposium on Computational geometry (2009), ACM, pp. 106-115.
[79] Gil, A., Juliá, M., and Reinoso, Ó.
Occupancy grid based graph-slam using the distance transform, surf features and sgd.
Engineering Applications of Artificial Intelligence 40 (2015), 1-10.
[80] Goldfarb, D.
A Family of Variable Metric Updates Derived by Variational Means.
Mathematics of Computing 24 (1970), 23-26.
[81] Grossekatthofer, K., and Yoon, Z.
Introduction into quaternions for spacecraft attitude representation.
TU Berlin 16 (2012).
[82] Gschwandtner, M., Kwitt, R., Uhl, A., and Pree, W.
BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing.
vol. 6939 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2011, ch. 20, pp. 199-208.
[83] Guizilini, V., and Ramos, F.
Large-scale 3d scene reconstruction with hilbert maps.
In Intelligent Robots and Systems (IROS), 2016 IEEE / RSJ International Conference on (2016), IEEE, pp. 3247-3254.
[84] Guo, Y., Bennamoun, M., Sohel, F., Lu, M., Wan, J., and Kwok, N. M.
A comprehensive performance evaluation of 3D local feature descriptors.
International Journal of Computer Vision 116, 1 (2016), 66-89.
[85] HAIDU, A.
Urosbridge.
Online Software, May 2018.
https://github.com/robcog-iai/UROSBridge.
[86] Handa, A., Whelan, T., McDonald, J., and Davison, A. J.
A benchmark for rgb-d visual odometry, 3D reconstruction and SLAM.
In Robotics and automation (ICRA), 2014 IEEE international conference on (2014), IEEE, pp. 1524-1531.
[87] Hehn, M., and D'Andrea, R.
Real-time trajectory generation for quadrocopters.
IEEE Transactions on Robotics 31, 4 (2015), 877-892.
[88] Heiden, E., Hausman, K., Sukhatme, G. S., and Agha-mohammadi, A.-a.
Planning high-speed safe trajectories in confidence-rich maps.
In Intelligent Robots and Systems (IROS), 2017 IEEE / RSJ International Conference on (2017), IEEE, pp. 2880-2886.
[89] Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M.
Autonomous visual mapping and exploration with a micro aerial vehicle.
Journal of Field Robotics 31, 4 (2014), 654-675.
[90] Hershberger, J. E., and Snoeyink, J.
Speeding up the Douglas-Peucker line-simplification algorithm.
No. TR-92-07. University of British Columbia, Department of Computer Science, 1992.
[91] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W.
OctoMap: An efficient probabilistic 3D mapping framework based on octrees.
Autonomous Robots 34, 3 (2013), 189-206.
[92] Hugh Durrant Whyte, and Tim Bailey.
Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms.

Tech. rep., Australian Centre for Field Robotics, 2006.
[93] Ichter, B., Schmerling, E., And Pavone, M.
Group Marching Tree: Sampling-Based Approximately Optimal Motion Planning on GPUs.
In Robotic Computing (IRC), IEEE International Conference on (2017), IEEE, pp. 219-226.
[94] Ioan A. Sucan, Mark Moll, and Lydia E. Kavrak.
The Open Motion Planning Library.
IEEE Robotics \& Automation Magazine 19, 4 (Dec. 2012), 72-82.
http://ompl.kavrakilab.org.
[95] Janson, L., Schmerling, E., Clark, A., and Pavone, M.
Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions.
The International journal of robotics research 34, 7 (2015), 883-921.
[96] Johnson, A. E., And Hebert, M.
Using spin images for efficient object recognition in cluttered 3D scenes.
IEEE Transactions on pattern analysis and machine intelligence 21, 5 (1999), 433-449.
[97] Jolliffe, I.
Principal component analysis.
Wiley Online Library, 2002.
[98] Jones, E., Oliphant, T., Peterson, P., et al.
SciPy: Open source scientific tools for Python.
Online, 2001.
[http://www.scipy.org/].
[99] Jung, S., Cho, S., Lee, D., Lee, H., and Shim, D. H.
A direct visual servoing-based framework for the 2016 IROS autonomous drone racing challenge.
Journal of Field Robotics 35, 1 (2018), 146-166.
[100] Kam, H. R., Lee, S.-H., Park, T., and Kim, C.-H.
RViz: a toolkit for real domain data visualization.
Telecommunication Systems 60, 2 (2015), 337-345.
[101] Kaplan, K.
500 drones light night sky to set record.
iQ by Intel, November 2016.
[102] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S.
Anytime motion planning using the RRT*.
In Robotics and Automation (ICRA), 2011 IEEE International Conference on (2011), IEEE, pp. 1478-1483.
[103] Kaul, L., Zlot, R., and Bosse, M.
Continuous-time three-dimensional mapping for micro aerial vehicles with a passively actuated rotating laser scanner.
Journal of Field Robotics 33, 1 (2016), 103-132.
[104] Klein, G., and Murray, D.
Parallel tracking and mapping for small AR workspaces.
In Mixed and Augmented Reality, 2007. ISMAR 2007. $6^{\text {th }}$ IEEE and ACM International Symposium on (2007), IEEE, pp. 225-234.
[105] Kobilarov, M.
Discrete Geometric Motion Control of Autonomous Vehicles.
University of Southern California, 2008.
[106] Koenderink, J. J., and Van Doorn, A. J.
Affine structure from motion.
JOSA A 8, 2 (1991), 377-385.
[107] Koenig, N., and Howard, A.
Design and use paradigms for gazebo, an open-source multi-robot simulator.
In Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE / RSJ International Conference on (2004), vol. 3, IEEE, pp. 2149-2154.
[108] Kraft, E.
A quaternion-based unscented Kalman filter for orientation tracking.
In Proceedings of the Sixth International Conference of Information Fusion (2003), vol. 1, pp. 4754.
[109] Kuether, D. J., Morrell, B. J., Chamitoff, G. E., Bishop, M., Mortari, D., Gibbens, P. W., and Coen, M.

Cohesive Autonomous Navigation System.
In AIAA Guidance Navigation and Control Conference, AIAA SciTech (San Diego, California, USA, 2016), IEEE.
[110] Kuffner, J. J., and LaValle, S. M.
RRT-connect: An efficient approach to single-query path planning.
In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on (2000), vol. 2, IEEE, pp. 995-1001.
[111] Kumar, V., and Michael, N.
Opportunities and challenges with autonomous micro aerial vehicles.
The International Journal of Robotics Research 31, 11 (2012), 1279-1291.
[112] Kushleyev, A., Mellinger, D., Powers, C., and Kumar, V.
Towards a swarm of agile micro quadrotors.
Autonomous Robots 35, 4 (2013), 287-300.
[113] Lalish, E., Morgansen, K. A., and Tsukamaki, t.
Decentralized Reactive Collision Avoidance for Multiple Unicycle-Type Vehicles.
In American Control Conference, 2008 (2008), IEEE, pp. 5055-5061.
[114] Landry, B.
Planning and control for quadrotor flight through cluttered environments.
PhD thesis, Massachusetts Institute of Technology, 2015.
[115] Langelaan, J., and Rock, S.
Navigation of small uavs operating in forests.
In AIAA Guidance, Navigation, and Control Conference and Exhibit (2004), p. 5140.
[116] LaValle, S. M.
Planning algorithms.
Cambridge university press, 2006.
[117] Lavoie, P.
NURBS++. A C++ library for manipulating NURBS curves and surfaces.
Online, 2002.
https://github.com/chrisidefix/nurbs.
[118] Lee, T., Leoky, M., and McClamroch, N. H.
Geometric tracking control of a quadrotor UAV on se (3).
In Decision and Control (CDC), $201049^{\text {th }}$ IEEE Conference on (2010), IEEE, pp. 5420-5425.
[119] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P.
Keyframe-based visual-inertial odometry using nonlinear optimization.
The International Journal of Robotics Research 34, 3 (2015), 314-334.
[120] Li, A. Q., Coskun, A., Doherty, S. M., Ghasemlou, S., Jagtap, A. S., Modasshir, M., Rahman, S., Singh, A., Xanthidis, M., OKane, J., et al.
Experimental comparison of open source vision-based state estimation algorithms.
In International Symposium on Experimental Robotics (2016), Springer, pp. 775-786.
[121] Loianno, G., Brunner, C., McGrath, G., and Kumar, V.
Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU.
IEEE Robotics and Automation Letters 2, 2 (2017), 404-411.
[122] Lu, P., and Liu, X.
Autonomous Trajectory Planning for Rendezvous and Proximity Operations by Conic Optimization.
Journal of Guidance, Control, and Dynamics 36, 2 (2013), 375-389.
[123] Luna, R., Sucan, I., Moll, M., Kavraki, L. E., et al.
Anytime solution optimization for sampling-based motion planning.

In Robotics and Automation (ICRA), 2013 IEEE International Conference on (2013), IEEE, pp. 5068-5074.
[124] Luo, J., and Hauser, K.
An empirical study of optimal motion planning.
In Intelligent Robots and Systems (IROS 2014), 2014 IEEE /RSJ International Conference on (2014), IEEE, pp. 1761-1768.
[125] Luo, Y.-Z., Lei, Y.-J., and Tang, G.-J.
Optimal Multi-Objective Nonlinear Impulsive Rendezvous.
Journal of guidance, control, and dynamics 30, 4 (2007), 994-1002.
[126] Lupashin, S., Schöllig, A., Sherback, M., and D'Andrea, R.
A simple learning strategy for high-speed quadrocopter multi-flips.
In Robotics and Automation (ICRA), 2010 IEEE International Conference on (2010), IEEE, pp. 1642-1648.
[127] Ma, W., and Kruth, J.-P.
Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces.
Computer-Aided Design 27, 9 (1995), 663-675.
[128] Mahony, R., Kumar, V., and Corke, P.
Multirotor aerial vehicles.
IEEE Robotics and Automation magazine 20, 32 (2012).
[129] Marder-Eppstein, E.
Project Tango.
In ACM SIGGRAPH 2016 Real-Time Live! (New York, NY, USA, 2016), SIGGRAPH '16, ACM, pp. 40:25-40:25.
[130] Markley, F. L., and Crassidis, J. L.
Fundamentals of spacecraft attitude determination and control, vol. 33.
Springer, 2014.
[131] Masoud, S. A., and Masoud, A. A.
Motion Planning in the Presence of Directional and Regional Avoidance Constraints Using Nonlinear, Anisotropic, Harmonic Potential Fields: A Physical Metaphor.
Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 32, 6 (2002), 705-723.
[132] Mathworks.
isosurface.
Matlab function, 2015.
http://www.mathworks.com/help/matlab/ref/isosurface.html.

## [133] MATLAB.

version 9.3 (R2017b).
The MathWorks Inc., Natick, Massachusetts, 2017.
[134] Matthies, L., Brockers, R., Kuwata, Y., and Weiss, S.
Stereo vision-based obstacle avoidance for micro air vehicles using disparity space.
In Robotics and Automation (ICRA), 2014 IEEE International Conference on (2014), IEEE, pp. 3242-3249.
[135] McCamish, S., Romano, M., Nolet, S., Edwards, C., and Miller, D. W.
Testing of Multiple-Spacecraft Control on SPHERES During Close-Proximity Operations.
Journal of Spacecraft and Rockets 46, 6 (2009), 1202-1213.
[136] McHenry, N., Coen, M., Hogan, R., Morrell, B., and Chamitoff, G.
Virtual reality multi-user space systems mission design and simulation: Engaging the public through open-source collaboration.
In 68th International Astronautical Congress (IAC) (Adelaide, Australia, Sept. 2017), no. IAC-17E1.6.13.
[137] Meier, K., Chung, S.-J., and Hutchinson, S.
Visual-inertial curve simultaneous localization and mapping: Creating a sparse structured world without feature points.
Journal of Field Robotics (2017).
[138] Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision.
Autonomous Robots 33, 1-2 (2012), 21-39.
[139] Mellinger, D., and Kumar, V.
Minimum snap trajectory generation and control for quadrotors.
In Robotics and Automation (ICRA), 2011 IEEE International Conference on (2011), IEEE, pp. 2520-2525.
[140] Mellinger, D., Michael, N., and Kumar, V.
Trajectory generation and control for precise aggressive maneuvers with quadrotors.
The International Journal of Robotics Research 31, 5 (2012), 664-674.
[141] Mendes, E., Koch, P., and Lacroix, S.
ICP-based pose-graph SLAM.
In Safety, Security, and Rescue Robotics (SSRR), 2016 IEEE International Symposium on (2016), IEEE, pp. 195-200.
[142] Michael Milford, and Gordon Wyeth.
Mapping a Suburb With a Single Camera Using a Biologically Inspired SLAM System.
IEEE Transactions on Robotics 24, 5 (Oct. 2008), 1038-1053.

## [143] Michael Milford, and Gordon Wyeth.

Persistent Navigation and Mapping using a Biologically Inspired SLAM System.
The International Journal of Robotics Research 29, 9 (Aug. 2010), 1191-1153.
[144] Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K.
Automatic grasp planning using shape primitives.
In Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on (2003), vol. 2, IEEE, pp. 1824-1829.
[145] MIT.
SPHERES website.
Online, 2014.
http://ssl.mit.edu/spheres/index.html.
[146] MIT.
Zero Robotics website.
Online, 2018.
http://zerorobotics.mit.edu/.
[147] Моhta, K., Watterson, M., Mulgaonkar, Y., Liu, S., Qu, C., Makineni, A., Saulnier, K., Sun, K., Zhu, A., Delmerico, J., et al.

Fast, autonomous flight in GPS-denied and cluttered environments.
Journal of Field Robotics 35, 1 (2018), 101-120.
[148] Montemerlo, M., and Thrun, S.
Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM.
In Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on (2003), vol. 2, IEEE, pp. 1985-1991.
[149] Mordvintsev, A., and Abid, K.
Introduction to SURF (Speeded-Up Robust Features).
Website, 2013.
[150] Moreno-Noguer, F., Lepetit, V., and Fua, P.
Accurate non-iterative $\mathrm{O}(\mathrm{n})$ solution to the PnP problem.
In Computer vision, 2007. ICCV 2007. IEEE 11 th international conference on (2007), IEEE, pp. 1-8.
[151] Morrell, B., Gibbens, P., and Chamitoff, G.
Application of a Trajectory Optimisation Algorithm for Dynamic Obstacle Avoidance and Multiple Vehicle Coordination.
In Fourth Australasian Conference on Unmanned Systems (Dec. 2014).
[152] Morrell, B., Player, P., Best, F., Fazio-Nagy, J., Chamitoff, G., and Holmes, W.
High school students coding for space: The Zero Robotics competition.
In International Astronautical Congress (2017), no. IAC-17-E1.2.13.
[153] Morrell, B., Rigter, M., Merewether, G., Reid, R., Thakker, R., Tzanetos, T., Rajur, V., and Chamitoff, G.

Differential flatness transformations for aggressive quadrotor flight.
In Robotics and Automation (ICRA), 2018 IEEE International Conference on Robotics and Automation (Brisbane, Australia, 2018), no. 1817, IEEE.
[154] Morrell, B., Thakker, R., Merewether, G., Reid, R., Rigter, M., Tzanetos, T., and Chamitoff, G.
Comparison of trajectory optimization algorithms for high-speed quadrotor flight near obstacles. IEEE Robotics and Automation Letters 3, 4 (2018), 4399-4406.
[155] Morrell, B. J., Chamitoff, G., and Gibbens, P.
Autonomous Operation of Multiple Free-Flying Robots on the International Space Station.
In 25th AAS /AIAA Spaceflight Mechanics Conference (Jan. 2015), vol. 155, pp. 2633-2650.
[156] Morrell, B. J., Chamitoff, G. E., Kuether, D. J., Coen, M., and Gibbens, P.
Integration of 3D SLAM, rigid body landmarks and 3D path planning.
In AIAA SPACE 2016. 2016, p. 5411.
[157] Mueller, M. W., Hehn, M., and D'Andrea, R.
A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification.
In Intelligent Robots and Systems (IROS), 2013 IEEE / RSJ International Conference on (2013), IEEE, pp. 3480-3486.
[158] Mueller, M. W., Hehn, M., and D'Andrea, R.
A computationally efficient motion primitive for quadrocopter trajectory generation.
IEEE Transactions on Robotics 31, 6 (2015), 1294-1310.
[159] Mukherjee, D., Wu, Q. J., and Wang, G.
A comparative experimental study of image feature detectors and descriptors.
Machine Vision and Applications 26, 4 (2015), 443-466.
[160] Munoz, J. D., and Fitz Coy, N. G.
Rapid Path-Planning Options for Autonomous Proximity Operations of Spacecraft.
In Proceedings of the AIAA/AAS Astrodynamics Specialist Conference, Toronto, Ortario Canada (2010).
[161] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D.
ORB-SLAM: a versatile and accurate monocular SLAM system.
IEEE Transactions on Robotics 31, 5 (2015), 1147-1163.
[162] Mur-Artal, R., and Tardós, J. D.
Fast relocalisation and loop closing in keyframe-based SLAM.
In Robotics and Automation (ICRA), 2014 IEEE International Conference on (2014), IEEE, pp. 846-853.
[163] Mur-Artal, R., and Tardós, J. D.
ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras.
IEEE Transactions on Robotics 33, 5 (2017), 1255-1262.
[164] Murray, R. M., Rathinam, M., and Sluis, W.
Differential flatness of mechanical control systems: A catalog of prototype systems.
In ASME international mechanical engineering congress and exposition (1995).
[165] Neunert, M., de Crousaz, C., Furrer, F., Kamel, M., Farshidian, F., Siegwart, R., and Buchli, J.
Fast nonlinear model predictive control for unified trajectory optimization and tracking.
In Robotics and Automation (ICRA), 2016 IEEE International Conference on (2016), IEEE, pp. 1398-1404.
[166] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J.
Dtam: Dense tracking and mapping in real-time.
In Computer Vision (ICCV), 2011 IEEE International Conference on (2011), IEEE, pp. 2320-2327.
[167] Niessner, M., Zollhöfer, M., Izadi, S., and Stamminger, M.
Real-time 3D reconstruction at scale using voxel hashing.
ACM Transactions on Graphics (ToG) 32, 6 (2013), 169.
[168] Nüchter, A., Lingemann, K., Hertzberg, J., and Surmann, H.
6D slam- 3D mapping outdoor environments.
In IEEE international workshop on safety, security, and rescue robotics (SSRR) (2006).
[169] OCallaghan, S. T., and Ramos, F. T.
Gaussian process Occupancy Maps.
The International Journal of Robotics Research 31, 1 (2012), 42-62.
[170] Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., and Galceran, E.
Continuous-time trajectory optimization for online UAV replanning.
In 2016 IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS) (Oct. 2016), pp. 5332-5339.
[171] Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., and Nieto, J.
Voxblox: Incremental 3D Euclidean signed distance fields for on-board mav planning.
In IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS) (2017).
[172] Oleynikova, H., Taylor, Z., Siegwart, R., and Nieto, J.
Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. IEEE Robotics and Automation Letters 3, 3 (2018), 1474-1481.
[173] Omer, J., and Farges, J.-L.
Hybridization of Nonlinear and Mixed-Integer Linear Programming for Aircraft Separation With Trajectory Recovery.

Intelligent Transportation Systems, IEEE Transactions on 14, 3 (Sept. 2013), 1218-1230.
[174] Ousingsawat, J., and Campbell, M. E.
On-Line Estimation and Path Planning For Multiple Vehicles in an Uncertain Environment.
International Journal of Robust and Nonlinear Control 14, 8 (2004), 741-766.
[175] Pallottino, L., Feron, E. M., and Bicchi, A.
Conflict Resolution Problems for Air Traffic Management Systems Solved with Mixed Integer Programming.
Intelligent Transportation Systems, IEEE Transactions on 3, 1 (2002), 3-11.
[176] Paranjape, A. A., Meier, K. C., Shi, X., Chung, S.-J., and Hutchinson, S.
Motion primitives and 3D path planning for fast flight through a forest.
The International Journal of Robotics Research 34, 3 (2015), 357-377.
[177] Perez-Grau, F. J., Ragel, R., Caballero, F., Viguria, A., and Ollero, A.
An architecture for robust UAV navigation in GPS-denied areas.
Journal of Field Robotics (2017), n/a-n/a.
[178] Petersen, K. B., Pedersen, M. S., et al.
The matrix cookbook.
Technical University of Denmark 7, 15 (2008), 510.
[179] Piegl, L.
Modifying the shape of rational B-splines. part 2: surfaces.
Computer-Aided Design 21, 9 (1989), 538-546.
[180] Piegl, L.
On NURBS: a survey.
IEEE Computer Graphics and Applications 11, 1 (1991), 55-71.
[181] Piegl, L., and Tiller, W.
The NURBS book.
Springer Science \& Business Media, 2012.
[182] Pizzoli, M., Forster, C., and Scaramuzza, D.
REMODE: Probabilistic, monocular dense reconstruction in real time.
In Robotics and Automation (ICRA), 2014 IEEE International Conference on (2014), IEEE, pp. 2609-2616.
[183] Qin, T., Li, P., AND SHEN, S.
Vins-mono: A robust and versatile monocular visual-inertial state estimator.
arXiv preprint arXiv:1708.03852 (2017).
[184] Qiu, W., AND Yuille, A.
UnrealCV: Connecting computer vision to unreal engine.
In European Conference on Computer Vision (2016), Springer, pp. 909-916.
[185] QuALCOMM.
Snapdragon flight 801 processor, 2017.
https://developer.qualcomm.com/hardware/snapdragon-flight.
[186] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and NG, A. Y.
ROS: an open-source robot operating system.
In ICRA workshop on open source software (2009), vol. 3, Kobe, Japan, p. 5.
[187] Ramos, F., and Ott, L.
Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent.
The International Journal of Robotics Research 35, 14 (2016), 1717-1730.
[188] RaO, A. V.
A survey of numerical methods for optimal control.
Advances in the Astronautical Sciences 135, 1 (2009), 497-528.
[189] Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolin, C., Sanders, I., and Huntington, G. T.
Algorithm 902: GPOPS, a Matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method.
ACM Transactions on Mathematical Software (TOMS) 37, 2 (2010), 22.
[190] Rao, D., Chung, S.-J., and Hutchinson, S.
CurveSLAM: An approach for vision-based navigation without point features.
In Intelligent Robots and Systems (IROS), 2012 IEEE / RSJ International Conference on (2012), IEEE, pp. 4198-4204.
[191] Rasmussen, C. E.
Gaussian processes for machine learning.
[192] Reid, R., Merewether, G., Tzanetos, T., Morrell, B., Rigter, M., and Matthies, L.
A high-speed autonomous quadrotor system for vision-based teach \& repeat.
Journal of Field Robotics (2018).
In submission process as of May 2018.
[193] Reveliotis, S. A., and Roszkowska, E.
Conflict Resolution in Free-Ranging Multivehicle Systems: A Resource Allocation Paradigm. Robotics, IEEE Transactions on 27, 2 (2011), 283-296.
[194] Richards, A., and How, J.
Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming. In American Control Conference, 2002. Proceedings of the 2002 (2002), vol. 3, pp. 1936-1941vol.3.
[195] Rigter, M.
Replanning strategies to improve real-time performance of trajectory optimisation.

Aero3711: Aerospace engineering project 2 report, The University of Sydney, Nov. 2016.
[196] Rimon, E., and Koditschek, D. E.
Exact Robot Navigation Using Artificial Potential Functions.
Robotics and Automation, IEEE Transactions on 8, 5 (1992), 501-518.
[197] Ross, I. M., and Karpenko, M.
A review of pseudospectral optimal control: from theory to flight.
Annual Reviews in Control 36, 2 (2012), 182-197.
[198] Rusu, R. B., Blodow, N., and Beetz, M.
Fast point feature histograms (FPFH) for 3D registration.
In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on (2009), IEEE, pp. 3212-3217.
[199] Rusu, R. B., Blodow, N., Marton, Z. C., and Beetz, M.
Aligning point cloud views using persistent feature histograms.
In Intelligent Robots and Systems, 2008. IROS 2008. IEEE / RSJ International Conference on (2008), IEEE, pp. 3384-3391.
[200] Rusu, R. B., and Cousins, S.
3D is here: Point Cloud Library (PCL).
In IEEE International Conference on Robotics and Automation (ICRA) (Shanghai, China, May 2011).
[201] Rusu, R. B., Holzbach, A., Blodow, N., and Beetz, M.
Fast geometric point labeling using conditional random fields.
In Intelligent Robots and Systems, 2009. IROS 2009. IEEE / RSJ International Conference on (2009), IEEE, pp. 7-12.
[202] Saenz Otero, A.
Design Principles for the Development of Space Technology Maturation Laboratories Aboard the International Space Station.
Doctor of philosophy, Massachusetts Institute of Technology, June 2005.
[203] Santos, J. M., Portugal, D., and Rocha, R. P.
An evaluation of 2D SLAM techniques available in robot operating system.
In Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on (2013), IEEE, pp. 1-6.
[204] Sayre-McCord, T., Antonini, A., Arneberg, J., Brown, A., Cavalheiro, G., Fang, Y., Gorodetsky, A., Guerra, W., McCoy, D., Quilter, S., Riether, F., Tal, E., Terzioglu, Y., Carlone, L., and Karaman, S.

FlightGoggles: Visual-inertial-odometry flight with photorealistic camera simulation in the loop. In International Conference on Robotics and Automation (ICRA) (Brisbane, 2018), IEEE.
[205] Scaramuzza, D., and Fraundorfer, F.
Visual odometry: Part I: The first 30 years and fundamentals.
IEEE robotics \& automation magazine 18, 4 (2011), 80-92.
[206] Schneider, T., Dymczyk, M. T., Fehr, M., Egger, K., Lynen, S., Gilitschenski, I., and SiEgWART, R.
maplab: An open framework for research in visual-inertial mapping and localization.
IEEE Robotics and Automation Letters (2018).
[207] Shanno, D. F.
Conditioning of Quasi-Newton Methods for Function Minimization.
Mathematics of Computing 24 (1970), 647-656.
[208] Smith, M., Baldwin, I., Churchill, W., Paul, R., and Newman, P.
The new college vision and laser data set.
The International Journal of Robotics Research 28, 5 (2009), 595-599.
[209] Sola, J., Vidal-Calleja, T., Civera, J., and Montiel, J. M. M.
Impact of landmark parametrization on monocular ekf-slam with points and lines.
International journal of computer vision 97, 3 (2012), 339-368.
[210] Spedicato, S., and Notarstefano, G.
Computing minimum-time trajectories for quadrotors via transverse coordinates.
In Decision and Control (CDC), 2016 IEEE 55 ${ }^{\text {th }}$ Conference on (2016), IEEE, pp. 239-244.
[211] Steder, B., Rusu, R. B., Konolige, K., and Burgard, W.
Point feature extraction on 3D range scans taking into account object boundaries.
In Robotics and automation (icra), 2011 ieee international conference on (2011), IEEE, pp. 26012608.
[212] Sturm, J., Magnenat, S., Engelhard, N., Pomerleau, F., Colas, F., Burgard, W., Cremers, D., AND Siegwart, R.
Towards a benchmark for RGB-D SLAM evaluation.
In Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS) (Los Angeles, USA, June 2011).
[213] Sun, K., Mohta, K., Pfrommer, B., Watterson, M., Liu, S., Mulgaonkar, Y., Taylor, C. J., and Kumar, V.

Robust stereo visual inertial odometry for fast autonomous flight.
IEEE Robotics and Automation Letters 3, 2 (2018), 965-972.
[214] Svestka, P., and Overmars, M. H.
Coordinated Path Planning for Multiple Robots.
Robotics and Autonomous Systems 23, 3 (1998), 125-152.
[215] Thiery, J.-M., Guy, É., and Boubekeur, T.
Sphere-Meshes: shape approximation using spherical quadric error metrics.
ACM Transactions on Graphics (TOG) 32, 6 (2013), 178.
[216] Thomas, J., Pope, M., Loianno, G., Hawkes, E. W., Estrada, M. A., Jiang, H., Cutkosky, M. R., and Kumar, V.

Aggressive flight with quadrotors for perching on inclined surfaces.
Journal of Mechanisms and Robotics 8, 5 (2016), 051007.
[217] Tombari, F., Salti, S., and Di Stefano, L.
Unique signatures of histograms for local surface description.
In European conference on computer vision (2010), Springer, pp. 356-369.
[218] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W.
Bundle adjustment, a modern synthesis.
In International workshop on vision algorithms (1999), Springer, pp. 298-372.
[219] Van Nieuwstadt, M., and Murray, R. M.
Real time trajectory generation for differentially flat systems.
IFAC Proceedings Volumes 29, 1 (1996), 2301-2306.
[220] Vásárhelyi, G., Virágh, C., Somorjai, G., Nepusz, T., Eiben, A. E., and Vicsek, T.
Optimized flocking of autonomous drones in confined environments.
Science Robotics 3, 20 (2018), eaat3536.
[221] Volkova, A., and Gibbens, P. W.
More robust features for adaptive visual navigation of UAVs in mixed environments.
Journal of Intelligent \& Robotic Systems 90, 1-2 (2018), 171-187.
[222] Wan, E., Van Der Merwe, R., et al.
The unscented Kalman filter for nonlinear estimation.
In Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000 (2000), IEEE, pp. 153-158.
[223] Wang, J., and Englot, B.
Fast, accurate Gaussian Process Occupancy Maps via test-data octrees and nested Bayesian fusion.
In Robotics and Automation (ICRA), 2016 IEEE International Conference on (2016), IEEE, pp. 1003-1010.
[224] Watterson, M., and Kumar, V.
Control of quadrotors using the Hopf fibration on SO(3).
In Proceedings of the 2017 International Symposium on Robotics Research (2018).
[225] Williams, D.
Edge Feature and Optical Flow Terrain Aid for GNSS-Denied Airborne Visual Navigation.

Phd thesis, The University of Sydney, School of Aeronautical, Mechanical and Mechatronic Engineering, 2017.
[226] Williams, O., and Fitzgibbon, A.
Gaussian Process Implicit Surfaces.
Gaussian Proc. in Practice (2007).
[227] Xue, Y., Lee, B., and Han, D.
Automatic Collision Avoidance of Ships.
Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment 223, 1 (2009), 33-46.
[228] ZHANG, Z.
Iterative point matching for registration of free-form curves and surfaces.
International Journal of Computer Vision 13, 2 (Oct. 1994), 119-152.


[^0]:    ${ }^{1}$ Running an algorithm online means that it is doing the computations while the robot is moving and making new observations.

[^1]:    ${ }^{2}$ More specific details on these contributions are given in Chapter 4.

[^2]:    ${ }^{1}$ See Section 2.2.1.2 for more details on occupancy grids.

[^3]:    ${ }^{2}$ The sensors are also active, projecting radiation into the environment, which may not be desired in particular application domains.

[^4]:    * Visual features are used both for detection and description unless otherwise stated.

[^5]:    ${ }^{1}$ Using the isosurface function from Matlab ${ }^{\circledR}$ [132].

[^6]:    ${ }^{2}$ Further developments could include angular velocity to update the error covariance, if gyroscope information was available.
    ${ }^{3}$ The simplicity of the observation function and Jacobian is the reason that an EKF was selected over an UKF, as one of the main advantages of the UKF is the Jacobians do not need to be computed.

[^7]:    ${ }^{1}$ This formulation has changed slightly from previous formulations [35] which had the $f_{s}$ and $f_{c}$ terms squared. The costs are not squared here to take advantage of the quadratic nature of cost functions, as described in Section 4.3.6.1

[^8]:    ${ }^{2}$ The randomised seeding work is predominantly by Marc Rigter [195], with whom this author collaborated, and is presented here for a complete presentation of ASTRO.

[^9]:    ${ }^{3}$ This section is work in collaboration with Marc Rigter [195] and is presented for a complete description of ASTRO.

[^10]:    ${ }^{4}$ The variable $k$ is being reused here to represent iterations through segments, and will represent this for the rest of this section

[^11]:    ${ }^{5}$ SciPy optimisation toolbox [98].

[^12]:    ${ }^{6}$ The integral of snap squared is a cost function that helps to produce dynamically feasible trajectories, as elaborated in Chapter 5

[^13]:    ${ }^{1}$ In a controller, the previous attitude set-point is used as the orientation to compare the axes-sets. Alternatively the current attitude could be used; however, this was found to be inferior in flight tests.
    ${ }^{2}$ The result from using $\boldsymbol{x}_{c}$ and $\boldsymbol{y}_{c}$ as well as the negative $\boldsymbol{x}_{b}$ and $\boldsymbol{y}_{b}$ of their results.

[^14]:    ${ }^{3} \mathrm{~A}$ line simplification algorithm produces the reduced waypoints [90].

[^15]:    ${ }^{1}$ Snapdragon 801, Quad-core 2.26 GHz
    ${ }^{2}$ Dedicated Apps DSP (QDSP6 V5A - $801 \mathrm{MHz}+256 \mathrm{KL} 2$ )
    ${ }^{3}$ a system developed by Google [129].

[^16]:    ${ }^{4}$ The PD controller is a simplification from [118] that removes the feedforward acceleration and gyroscopic compensation terms. This simplification is done because of the difficulty in accurately characterising the desired feed-forward terms.

[^17]:    ${ }^{5}$ Such as the Asus Zenphone AR

[^18]:    ${ }^{6}$ The code for the GCS is open sourced and available on GitHub: https: //github. com/genemerewether/torq.

[^19]:    ${ }^{1}$ Variations in trajectories are due to the random perturbations technique in ASTRO.

[^20]:    ${ }^{1}$ As a general rule: a transformation from frame $A$ to frame $B$ is equivalent to placing the vector in $A$ straight into $B$ (the same values) and rotating it by the required steps to align $B$ with $A$.

