

Motion planning algorithms for autonomous navigation for a  
rotary-wing UAV

by

Coenraad Johannes Beyers

*Thesis presented in partial fulfilment of the requirements for the degree of  
Master of Science in Engineering  
at Stellenbosch University*



Supervisors:

Dr C.E. van Daalen   Mr J.A.A. Engelbrecht  
Department Electrical and Electronic Engineering

March 2013

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2013

# Abstract

A typical autopilot system is capable of steering an Unmanned Aerial Vehicle (UAV) along a set of defined waypoints. The defined waypoints are static, and if change is necessary (e.g. due to environmental changes) a human operator has to intervene. In order to fully achieve autonomous navigation the vehicle must be able to act on changing situations, and to this end motion planning and conflict detection is employed.

This project concerns motion planning for a rotary wing UAV, where vehicle controllers are already in place, and map data is readily available to a collision detection module. In broad terms, the goal of the motion planning algorithm is to provide a safe (i.e. obstacle free) flight path between an initial- and goal waypoint. This project looks at two specific motion planning algorithms, the Rapidly Exploring Random Tree (or RRT\*), and the Probabilistic Roadmap Method (or PRM).

The primary focus of this project is learning how these algorithms behave in specific environments and an in depth analysis is done on their differences. A secondary focus is the execution of planned paths via a Simulink simulation and lastly, this project also looks at the effect of path replanning.

The work done in this project enables a rotary wing UAV to autonomously navigate an uncertain, dynamic and cluttered environment. The work also provides insight into the choice of an algorithm for a given environment: knowing which algorithm performs better can save valuable processing time and will make the entire system more responsive.

# Uittreksel

'n Tipiese vliegstuurotomat is daartoe in staat om 'n onbemande lugvaartvoertuig (UAV) so te stuur dat 'n stel gedefinieerde punte gevolg word. Die punte moet egter vooraf beplan word, en indien enige verandering nodig is (bv. as gevolg van veranderinge in die omgewing) is dit nodig dat 'n menslike operateur betrokke moet raak. Vir voertuie om ten volle outonoom te kan navigeer, moet die voertuig in staat wees om te kan reageer op veranderende situasies. Vir hierdie doel word kinodinamiese beplanningsalgoritmes en konfliktdeteksie-metodes gebruik.

Hierdie projek behels kinodinamiese beplanningsalgoritmes vir 'n onbemande helikopter, waar die beheerders vir die voertuig reeds in plek is, en omgewingsdata beskikbaar is vir 'n konfliktdeteksie-module. In breë terme is die doel van die kinodinamiese beplanningsalgoritme om 'n veilige (d.w.s 'n konflikvrye) vlugpad tussen 'n begin- en eindpunt te vind. Hierdie projek kyk na twee spesifieke kinodinamiese beplanningsalgoritmes, die "Rapidly exploring Random Tree\*" (of RRT\*), en die "Probabilistic Roadmap Method" (of PRM).

Die primêre fokus van hierdie projek is om die gedrag van hierdie algoritmes in spesifieke omgewings te analiseer en 'n volledige analise te doen op hul verskille. 'n Sekondêre fokus is die uitvoering van 'n beplande vlugpad d.m.v 'n Simulink-simulasie, en laastens kyk hierdie projek ook na die effek van padherbeplanning.

Die werk wat gedoen is in hierdie projek stel 'n onbemande helikopter in staat om outonoom te navigeer in 'n onsekere, dinamiese en besige omgewing. Die werk bied ook insig in die keuse van 'n algoritme vir 'n gegewe omgewing: om te weet watter algoritme beter uitvoertye het kan waardevolle verwerkingstyd bespaar, en verseker dat die hele stelsel vinniger kan reageer.

# Contents

|   |            |
|---|------------|
| <b>Abstract</b>   | <b>iii</b> |
| <b>Uittreksel</b>   | <b>iv</b>  |
| <b>List of Figures</b>  | <b>ix</b>  |
| <b>List of Tables</b>   | <b>xii</b> |
| <b>Nomenclature</b>   | <b>xiv</b> |
| <b>Acknowledgements</b>   | <b>xvi</b> |
| <b>1 Introduction</b>   | <b>1</b>   |
| 1.1 The relevance of Autonomous Navigation . . . . .                    | 1          |
| 1.2 System architecture . . . . .                                       | 2          |
| 1.3 Thesis overview . . . . .   | 3          |
| 1.4 Conclusion . . . . .  | 4          |
| <b>2 Vehicle Model and Controllers</b>                                  | <b>5</b>   |
| 2.1 Axis System Definition . . . . .                                    | 5          |
| 2.1.1 Earth Reference Frame . . . . .                                   | 5          |
| 2.1.2 Body Reference Frame . . . . .                                    | 5          |
| 2.1.3 Reference Frame Relationship . . . . .                            | 6          |
| 2.2 The Vehicle . . . . .   | 7          |
| 2.2.1 Vehicle choice . . . . .  | 7          |
| 2.2.2 Vehicle actuators . . . . .                                       | 8          |
| 2.2.3 Vehicle Model . . . . .   | 9          |
| 2.2.4 Vehicle controllers . . . . .                                     | 12         |
| 2.2.4.1 Heading angle controller . . . . .                              | 12         |
| 2.2.4.2 Heave position controller . . . . .                             | 13         |
| 2.2.4.3 Lateral and Longitudinal position controllers . . . . .         | 14         |
| 2.2.4.4 Vehicle model and controller simulation with Simulink . . . . . | 15         |
| 2.2.5 Vehicle Estimator . . . . .                                       | 15         |
| 2.3 Conclusion . . . . .  | 16         |
| <b>3 Motion planning</b>  | <b>17</b>  |
| 3.1 Motion planning problem statement . . . . .                         | 17         |
| 3.2 Available motion planning algorithms . . . . .                      | 18         |
| 3.2.1 Complete motion planning . . . . .                                | 18         |
| 3.2.2 Almost complete motion planners . . . . .                         | 18         |
| 3.2.2.1 Grid-based search . . . . .                                     | 18         |
| 3.2.2.2 Potential fields . . . . .                                      | 19         |

|          |   |           |
|----------|---|-----------|
| 3.2.2.3  | Sampling based  | 19        |
| 3.2.2.4  | Conclusion  | 19        |
| 3.3      | Sampling based motion planning  | 20        |
| 3.3.1    | The Probabilistic Roadmap Method  | 20        |
| 3.3.2    | The Rapidly exploring Random Tree   | 21        |
| 3.4      | Problem statement requirements  | 22        |
| 3.4.1    | Kinodynamic and nonholonomic motion constraints   | 22        |
| 3.4.2    | Manoeuvres  | 23        |
| 3.4.3    | Conflict detection  | 24        |
| 3.4.4    | Cost function   | 24        |
| <b>4</b> | <b>Algorithm implementation</b>   | <b>26</b> |
| 4.1      | Introduction  | 26        |
| 4.2      | Hardware and software specifications  | 26        |
| 4.3      | Generic sampling based algorithm  | 26        |
| 4.4      | Sampling milestones   | 27        |
| 4.5      | The manoeuvre library   | 27        |
| 4.6      | Data structures   | 30        |
| 4.6.1    | The milestone data structure  | 30        |
| 4.6.2    | The node linked list data structure   | 31        |
| 4.7      | Conflict detection  | 32        |
| 4.8      | The Probabilistic Roadmap Method - PRM  | 33        |
| 4.8.1    | The Local Planning Method - LPM   | 34        |
| 4.8.2    | The Extend method   | 34        |
| 4.8.3    | The Path Planner  | 36        |
| 4.9      | The Rapidly exploring Random Tree* - RRT*   | 37        |
| 4.9.1    | The Steer method  | 38        |
| 4.9.2    | The Extend method   | 39        |
| 4.9.3    | Path planner  | 43        |
| 4.10     | Conclusion  | 44        |
| <b>5</b> | <b>Algorithm Analysis</b>   | <b>45</b> |
| 5.1      | Introduction  | 45        |
| 5.2      | PRM analysis  | 46        |
| 5.2.1    | Key concepts necessary to determine the theoretical upper bound                               | 46        |
| 5.2.1.1  | Definition of the $\beta$ -lookout( $\mathcal{S}$ )   | 50        |
| 5.2.1.2  | Definition of $\alpha$  | 51        |
| 5.2.2    | Theoretical performance bound for finding a path with a PRM                                   | 51        |
| 5.2.3    | Determining $\alpha$ and the $\beta$ -lookout( $\mathcal{S}$ ) for the PRM                    | 55        |
| 5.2.3.1  | Probability with relative frequency   | 55        |
| 5.2.3.2  | Determining $\alpha$ and the $\beta$ -lookout( $\mathcal{S}$ ) using relative frequency       | 56        |
| 5.2.4    | Relating $\alpha$ and $\beta$ on a computer for a specific environment and a specific vehicle | 57        |
| 5.2.5    | Conclusion  | 60        |
| 5.3      | RRT* Analysis   | 60        |
| 5.3.1    | Theoretical Upper bound for finding a path with a RRT*  | 61        |
| 5.3.1.1  | The Attraction sequence   | 63        |
| 5.3.1.2  | Probability of sampling a milestone in each attraction set                                    | 64        |
| 5.3.1.3  | Determining the location of the attraction sequence   | 64        |
| 5.3.1.4  | Determining the volume of the subsets of the attraction sequence                              | 65        |
| 5.3.2    | Conclusion  | 67        |
| 5.4      | Histogram analysis  | 67        |

|          |  |            |
|----------|--|------------|
| 5.4.1    | Iteration count . . . . .                                    | 67         |
| 5.4.1.1  | Theoretical versus Practical iteration requirement . . . . . | 69         |
| 5.4.2    | Milestone count . . . . .                                    | 69         |
| 5.4.2.1  | Theoretical versus Practical milestone requirement . . . . . | 71         |
| 5.4.3    | CPU-time . . . . .   | 71         |
| 5.4.3.1  | CPU time of theoretical bound . . . . .                      | 73         |
| 5.4.4    | Path cost . . . . .  | 73         |
| 5.5      | Conclusion . . . . .   | 75         |
| <b>6</b> | <b>Path Replanning</b>                                       | <b>76</b>  |
| 6.1      | Introduction . . . . .                                       | 76         |
| 6.2      | Implementation . . . . .                                     | 77         |
| 6.3      | Simulation example . . . . .                                 | 77         |
| 6.3.1    | Initial path . . . . .                                       | 77         |
| 6.3.2    | Improving the path . . . . .                                 | 78         |
| 6.3.3    | Path with new environment information . . . . .              | 79         |
| 6.4      | Conclusion . . . . .   | 80         |
| <b>7</b> | <b>Software simulation</b>                                   | <b>82</b>  |
| 7.1      | Introduction . . . . .                                       | 82         |
| 7.2      | Simulink model . . . . .                                     | 82         |
| 7.2.1    | Path planner block . . . . .                                 | 86         |
| 7.2.1.1  | Necessary path information . . . . .                         | 86         |
| 7.2.1.2  | Manoeuvre controllers . . . . .                              | 87         |
| 7.3      | Results . . . . .  | 89         |
| 7.3.1    | Path following . . . . .                                     | 89         |
| 7.3.2    | Cost function . . . . .                                      | 91         |
| 7.4      | Conclusion . . . . .   | 91         |
| <b>8</b> | <b>Environments</b>  | <b>93</b>  |
| 8.1      | Environment one . . . . .                                    | 93         |
| 8.1.1    | Performance of the RRT* vs the PRM . . . . .                 | 93         |
| 8.2      | Environment two . . . . .                                    | 95         |
| 8.2.1    | Performance of the RRT* vs the PRM . . . . .                 | 95         |
| 8.2.2    | Summary . . . . .  | 99         |
| 8.3      | Environment three . . . . .                                  | 99         |
| 8.3.1    | Performance of the RRT* vs the PRM . . . . .                 | 100        |
| 8.3.2    | Summary . . . . .  | 103        |
| <b>9</b> | <b>Conclusion</b>  | <b>105</b> |
| 9.1      | Project Scope . . . . .                                      | 105        |
| 9.2      | Motion planning algorithms . . . . .                         | 105        |
| 9.3      | Algorithm implementation . . . . .                           | 106        |
| 9.3.1    | Probabilistic Roadmap Method . . . . .                       | 106        |
| 9.3.2    | Rapidly exploring Random Tree* . . . . .                     | 106        |
| 9.4      | Analysis . . . . .   | 106        |
| 9.4.1    | Theoretical . . . . .  | 106        |
| 9.4.2    | Histogram . . . . .  | 106        |
| 9.4.2.1  | CPU time . . . . .   | 107        |
| 9.4.2.2  | Path cost . . . . .  | 107        |
| 9.5      | Path Replanning . . . . .                                    | 107        |
| 9.6      | Software simulation . . . . .                                | 107        |

|  |             |
|--|-------------|
| <i>CONTENTS</i>                        | <b>viii</b> |
| <b>A MATLAB Code to execute a Path</b> | <b>108</b>  |
| <b>Bibliography</b>                    | <b>110</b>  |



# List of Figures

|      |  |    |
|------|--|----|
| 1.1  | Shows the architecture of the entire autonomous navigation project. Source: C.E. van Daalen . . . . .  | 3  |
| 2.1  | Earth Axis System Definition. . . . .  | 6  |
| 2.2  | Aircraft Body Axis System Definition [1]. . . . .  | 6  |
| 2.3  | The X-Cell during autonomous hover [1]. . . . .  | 8  |
| 2.4  | On the left a hinged rotor hub is shown, and on the right a rigid rotor hub is shown [1]. . . . .  | 8  |
| 2.5  | The longitudinal model of the vehicle [7]. . . . .   | 11 |
| 2.6  | The lateral model of the vehicle [7]. . . . .  | 11 |
| 2.7  | The heave model of the vehicle [7]. . . . .  | 11 |
| 2.8  | The heading model of the vehicle [7]. . . . .  | 11 |
| 2.9  | Successive-loop-closure for the heading plant [7]. . . . .   | 12 |
| 2.10 | Heading angle step response [7]. The amplitude is given in radians. . . . .  | 13 |
| 2.11 | Successive-loop-closure for the heave plant [7]. . . . .   | 13 |
| 2.12 | Heave position step response [7]. The amplitude is given in meters. . . . .  | 14 |
| 2.13 | Successive-loop-closure for the lateral plant [7]. . . . .   | 14 |
| 2.14 | Successive-loop-closure for the longitudinal plant [7]. . . . .  | 15 |
| 2.15 | Longitudinal position step response [7]. The amplitude is given in meters. . . . .   | 15 |
| 2.16 | The kinematic state estimator structure by Hough [2]. . . . .  | 16 |
| 3.1  | This figure shows the PRM algorithm with several milestones in its tree. . . . .   | 20 |
| 3.2  | This figure shows the classic RRT algorithm after 100 iterations. . . . .  | 21 |
| 3.3  | A plot showing the cost function used to determine costs. The numbers on the vertical axis represent sample averages of the measured (during simulation) force of the main and tail rotors. The measurements are dimensionless and only provide information relative to each other. . . . .                          | 25 |
| 4.1  | A LPM manoeuvre consisting of a left turn, straight line, and a right turn. . . . .  | 29 |
| 4.2  | A Steer manoeuvre consisting of a left turn and a straight line. . . . .   | 29 |
| 4.3  | The execution time for adding 300 milestones to a tree is shown here for both the classic and updated PRM Extend algorithms. . . . .   | 36 |
| 4.4  | This figure shows a recently added milestone in blue, a ball (shown as a circle in 2 dimensions) of radius $\eta$ in black, and a path between the start point and goal in green. The milestones shown in red are in the <code>list<sub>rrt*</sub></code> list but do not form part of the path to the goal. . . . . | 42 |
| 4.5  | This figure shows the improved path of Figure 4.4. . . . .   | 43 |
| 5.1  | LPM reachability for a point $p$ . . . . .   | 47 |

|      |  |    |
|------|--|----|
| 5.2  | In blue the $\beta$ -lookout of $\mathcal{S}$ is shown for a <b>small</b> (close to zero) value of $\beta$ . The red area shows the LPM reachability set $\mathcal{R}_{lpm}(p_1)\setminus\mathcal{S}$ and the green area shows the LPM reachability set $\mathcal{R}_{lpm}(p_0)\setminus\mathcal{S}$ . . . . . | 48 |
| 5.3  | $\beta$ -lookout of $\mathcal{S}$ for a <b>large</b> (close to one) value of $\beta$ . . . . .   | 49 |
| 5.4  | In blue, the $\beta$ -lookout( $\mathcal{S}_1$ ), in red the $\beta$ -lookout( $\mathcal{S}_2$ ), and in green the $\beta$ -lookout( $\mathcal{S}_3$ ). . . . .  | 50 |
| 5.5  | Introducing Cases A and B. . . . .   | 52 |
| 5.6  | This figure illustrates a large intersection between the endgame region and $\mathcal{R}_{lpm}(PRM_{tree})$ , as well as a milestone in the intersection. For this case a path is formed between the initial milestone and goal milestones. . . . .  | 53 |
| 5.7  | The environment wherein the PRM and RRT* algorithms are analysed. . . . .  | 58 |
| 5.8  | Plot of $\alpha$ vs $\beta$ in the environment shown in Figure 5.7. . . . .  | 59 |
| 5.9  | Upper-bound plot of the number of milestones necessary for a 99.99% probability that the PRM will not require more than $r$ milestones to find a path vs $\beta$ for the environment shown in Figure 5.7. The smallest value of $r = 1328$ , with $\beta = 0.03075$ and $\alpha = 0.4408$ . . . . .            | 60 |
| 5.10 | Steer reachability of $p$ . . . . .  | 61 |
| 5.11 | Attraction set of $\mathcal{A}_1$ . . . . .  | 62 |
| 5.12 | Attraction set of $\mathcal{A}_1$ . . . . .  | 63 |
| 5.13 | Six subsets ( $k = 5$ ) are shown. A path between the initial and goal milestones is formed by sampling a milestones in the subsets 1 to 4. . . . .  | 65 |
| 5.14 | This figure shows an example attraction sequence as determined by the approach in Subsection 5.3.1.4. . . . .  | 66 |
| 5.15 | Histogram plot of iterations required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. . . . .  | 68 |
| 5.16 | Histogram plot of iterations required by the RRT* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. The measurements show that 0 times out of 100000 runs the algorithm required more than 7174 iterations. . . . .   | 69 |
| 5.17 | Histogram plot of number of milestones required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. The measurements showed that 0 time out of 100000 runs the algorithm required more than 1328 milestones. . . . .                                     | 70 |
| 5.18 | Histogram plot of number of milestones required by the RRT* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. . . . .   | 71 |
| 5.19 | Histogram plot of CPU-time required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. . . . .  | 72 |
| 5.20 | Histogram plot of cpu time required by the RRT* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. . . . .   | 73 |
| 5.21 | Histogram plot of path costs for the PRM in the environment shown in Figures 8.1(a) and 8.1(b) for 100000 runs. . . . .  | 74 |
| 5.22 | Histogram plot of path costs for the RRT* in the environment shown in Figures 8.1(a) and 8.1(b) for 100000 runs. . . . .   | 74 |
| 6.1  | A plot showing the top and side view plots of the first path the RRT* path planner found and the <b>initial</b> path the replanner will execute. . . . .   | 78 |
| 6.2  | A plot showing the top and side view plots of an <b>improved</b> path (cost: 7887) the RRT* path planner found in blue and the old path in red. . . . .  | 78 |
| 6.3  | A plot showing the top and side view plots of an <b>improved</b> path (cost: 7556) the RRT* path planner found in blue and the old path in red. . . . .  | 79 |
| 6.4  | A plot showing the top and side view plots of an <b>improved</b> path (cost: 7227) the RRT* path planner found in blue and the old path in red. . . . .  | 79 |
| 6.5  | A plot showing the top and side view plots of the path the RRT* path planner found with the <b>updated</b> environment information. . . . .  | 80 |

|      |   |     |
|------|---|-----|
| 6.6  | A plot showing the top and side view plots of the path (cost: 6789) the vehicle traversed. . . . .  | 80  |
| 7.1  | This figure shows the high level Simulink simulation. The simulation consists of three parts, the helicopter model (the block shown in blue), the autopilot (the block shown in red) and the data viewer or logger (the block shown in orange). . . . . | 83  |
| 7.2  | This figure shows an overview of the Autopilot in the Simulink simulation. . . . .  | 84  |
| 7.3  | This figure shows the inner loop controllers. . . . .   | 85  |
| 7.4  | This figure shows the path planner block and its interaction with the inner loop controllers. . . . .   | 85  |
| 7.5  | This figure shows a planned path superimposed on the simulated path of the vehicle. . . . .   | 90  |
| 8.1  | Path generated by the RRT*. . . . .   | 94  |
| 8.2  | Path generated by the PRM. . . . .  | 94  |
| 8.3  | Path generated by the RRT*. . . . .   | 95  |
| 8.4  | Path generated by the PRM. . . . .  | 96  |
| 8.5  | Iteration histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 97  |
| 8.6  | Milestone histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 97  |
| 8.7  | CPU time histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .  | 98  |
| 8.8  | Path cost histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 98  |
| 8.9  | Path generated by the RRT*. . . . .   | 100 |
| 8.10 | Path generated by the PRM. . . . .  | 100 |
| 8.11 | Iteration histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 101 |
| 8.12 | Milestone histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 102 |
| 8.13 | CPU time histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .  | 102 |
| 8.14 | Path cost histograms of 100000 runs for both the PRM and RRT* algorithms. . . . .   | 103 |

# List of Tables

|     |  |     |
|-----|--|-----|
| 2.1 | Summary of the vehicle states included in its model. . . . .   | 9   |
| 4.1 | The <b>milestone</b> data structure. . . . .   | 31  |
| 4.2 | The <b>milestone node</b> data structure. . . . .  | 32  |
| 4.3 | The <b>milestone node list</b> data structure. . . . .   | 32  |
| 7.1 | Maximum, mean and standard deviation errors in x,y,z and time dimension<br>(along-track) between the planned and travelled path. . . . . | 90  |
| 7.2 | Maximum, mean and standard deviation errors in x,y,z dimension (cross-track)<br>between the planned and travelled path. . . . .          | 91  |
| 7.3 | Difference in cost between Simulink simulation and that determined by the cost<br>function. . . . .                                      | 91  |
| 8.1 | Summary of the iteration, milestone, CPU time, and path cost histograms from<br>Subsection 8.2.1. . . . .                                | 99  |
| 8.2 | Summary of the iteration, milestone, CPU time, and path cost histograms from<br>Section 8.3. . . . .                                     | 104 |

# List of Algorithms

|   |   |    |
|---|---|----|
| 1 | A generic sampling based algorithm. . . . . | 27 |
| 2 | The conflict detection algorithm. . . . .   | 33 |
| 3 | The Local planning algorithm. . . . .       | 34 |
| 4 | The PRM Extend algorithm. . . . .           | 35 |
| 5 | The PRM path planner algorithm. . . . .     | 37 |
| 6 | The RRT* steer algorithm. . . . .           | 38 |
| 7 | The RRT* extend algorithm. . . . .          | 41 |
| 8 | The RRT* path planner algorithm. . . . .    | 44 |

# Nomenclature

## Abbreviations and Acronyms

|                   |  |
|-------------------|--|
| PRM               | Probabilistic Roadmap Method.            |
| RRT(*)            | Rapidly Exploring Random Tree (star).    |
| LPM or <i>lpm</i> | Local planning method.                   |
| UAV               | Unmanned Aerial Vehicle.                 |
| EC                | Energy Cost.                             |
| CF                | Cost fraction, fraction of a total cost. |
| CCPM              | Cyclic-, Collective Pitch Mixing.        |
| TTP               | Tip path plane.                          |
| GPS               | Global Positioning System.               |
| DCM               | Direction Cosine Matrix.                 |

## Notational convention

|                        |  |
|------------------------|--|
| N                      | is used to indicate North, and is used interchangeably with the x dimension.         |
| E                      | is used to indicate East, and is used interchangeably with the y dimension.          |
| D                      | is used to indicate Down, and is used interchangeably with the negative z dimension. |
| $\eta$                 | Limit on the length of a RRT* <i>steer</i> manoeuvre.                                |
| $m_b$                  | denotes the start/initial space $\times$ time (or milestone) configuration.          |
| $m_{rand}$             | denotes a random milestone.  |
| $m_g$                  | denotes the end/stop/goal milestone.   |
| $\mathbf{m}_{newest}$  | denotes the milestone most recently added to an algorithm's data structure.          |
| $\mathbf{m}_{nearest}$ | denotes the closest milestone (in euclidean distance) to another milestone.          |
| $\mathcal{S}$          | denotes a <i>state space</i> .   |
| $\mathcal{T}$          | denotes a <i>time space</i> .  |
| $\mathcal{F}$          | denotes the set of all collision-free points.  |

$\mathcal{F} \subset \mathcal{S} \times \mathcal{T}$  denotes a *free space*.

$\mathcal{R}(\mathcal{S})$  denotes the set of *reachable points*<sup>1</sup> from the set  $\mathcal{S}$ :

$$\mathcal{R}(\mathcal{S}) = \bigcup_{p \in \mathcal{S}} \mathcal{R}(p)$$

where  $p$  refers to different points in  $\mathcal{S}$ .

$\mathcal{R}_{lpm}(\mathcal{S})$  denotes the set of reachable points from the set  $\mathcal{S}$  through executing a single *lpm* run:

$$\mathcal{R}_{lpm}(\mathcal{S}) = \bigcup_{p \in \mathcal{S}} \mathcal{R}_{lpm}(p)$$

where  $p$  refers to different points in  $\mathcal{S}$ .

$\mathcal{X} = \mathcal{R}(m_b)$  denotes the set of all points reachable from  $m_b$ .

**tree**<sub>pr<sub>m</sub></sub> denotes a tree data structure used by the PRM where each node in the tree has parent and child relationships except the root node.

**tree**<sub>rrt\*</sub> denotes a tree data structure used by the RRT\* where each node in the tree has parent and child relationships except the root node.

$\rho(a, b)$  is the euclidean distance between the points  $a$  and  $b$  in  $\mathcal{S}$ .

$mp_*$  denotes a manoeuvre primitive, where  $*$  can be *lt* for a left turn, *rt* for a right turn or *fl* for a straight line

$man_*$  denotes a manoeuvre library entry, where  $*$  may be any combination of *l* for left turn, *r* for right turn, or *f* for forward ex.  $man_{lfr}$  is left forward right manoeuvre library entry.

---

<sup>1</sup>A reachable point is a point to which it is possible to connect to with multiple local planning method runs.

# Acknowledgements

I would like to thank the following people:

- Dr Corné van Daalen, for all his guidance and feedback.
- For all the support and motivation of my family.
- All my friends in the ESL.
- My wife to be, Pascalle, for her love, support and motivation.
- My mother for proofreading this thesis.



# Chapter 1

## Introduction

### 1.1 The relevance of Autonomous Navigation

The use of Unmanned Aerial Vehicles (UAVs) in both commercial and military applications are becoming ever more significant. An autonomous UAV<sup>1</sup> does not require any human intervention<sup>2</sup> after assigning a goal to it; that is, after the vehicle is assigned a job to do, it is capable of completing the navigational aspects of its job without any human intervention.

Applications of autonomous navigation range from military purposes, to the surveying of crops, and includes unmanned space exploration.

- UAVs have been mostly used by the military because of their ability to operate in dangerous locations without endangering their human operators. Autonomous navigation seeks to remove human interaction even further, perhaps even to a point where certain military tasks (e.g. reconnaissance) are performed fully autonomously, by an UAV.
- Surveying of crops is currently a relatively costly task; it requires hiring a pilot, an aeroplane, and surveillance equipment, as opposed to an autonomous UAV being a once-off investment. Furthermore, an autonomous UAV would be able to survey crops (almost) on-demand.
- For space exploration, real time control of a rover is not possible<sup>3</sup> at great distances, hence necessitating some form of autonomous navigation for the rover.

The intended application of this thesis is autonomous navigation of rotary-wing UAVs. Autonomous rotary-wing UAVs, as opposed to fixed-wing UAVs, have a number of applications, e.g. where:

- a surveillance stream is required of an agile target,
- vertical take-off and landing is required, or
- manoeuvring space is limited.

---

<sup>1</sup>This project specifically focuses on an Unmanned Aerial Vehicle, however the aspects relating to autonomous navigation in this document are also applicable to other vehicles, including but not limited to, manned, unmanned, aerial, water-based or land-based vehicles.

<sup>2</sup>Regarding navigation.

<sup>3</sup>E.g. communication with rovers on Mars takes between 10 and 20 minutes [3], and the Deep Space Network (or DSN) is used by other projects, which renders real time control of the vehicle impossible.

This chapter discusses an autonomous navigation system architecture (Section 1.2) and the necessary modules (or building blocks) required to build such a system. The chapter then identifies the modules in the system architecture which forms the focus of this project. Chapter 2 looks at the vehicle for which this system is being built, as well as the work already completed on the vehicle.

## 1.2 System architecture

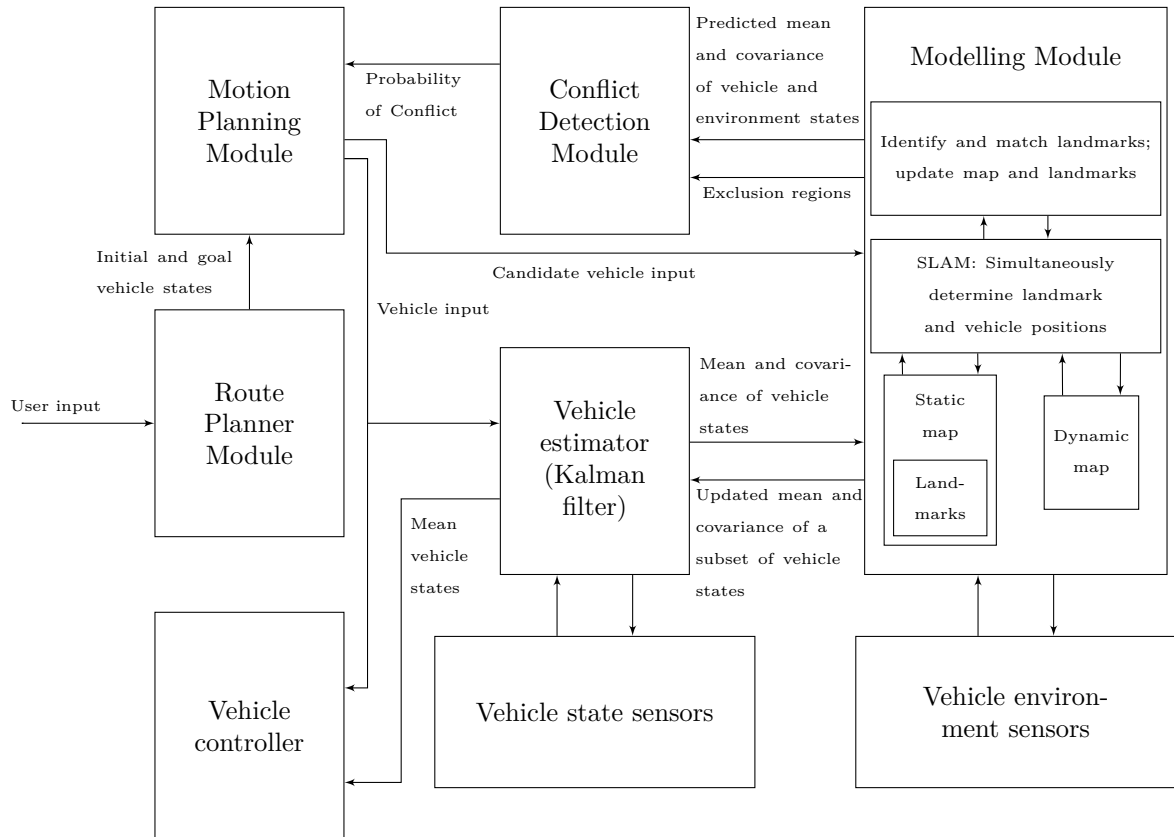
In order to achieve autonomous navigation, a couple of modules are required, namely: vehicle controllers, sensors, converting sensor data into map information, testing flight paths for conflict, and motion planning. The focus of this project is on the motion planning required to autonomously navigate from one waypoint to another.

The architecture required for autonomous navigation is shown in Figure 1.1. The **route planner module** takes input from an user, and accordingly determines the initial and goal vehicle states, which are passed to the **motion planning module**. The motion planning module plans the path between the initial and goal states, and it uses the **conflict detection module** to obtain the probability of conflict along any path segment. If the accumulated probability of conflict along the entire path is below a chosen threshold, then the motion planning module passes the necessary input to the **vehicle's controllers, estimator, and modelling module**. The vehicle's controllers are responsible for commanding the vehicle's actuators, and the vehicle's estimator is responsible for approximating the vehicle's states through the **vehicle's state sensors**. The modelling module contains models of the vehicle, as well as the environment. The purpose of the modelling module is to:

1. provide the conflict detection module with predicted mean and covariance states of the vehicle and the environment,
2. provide the conflict detection module with the exclusion regions<sup>4</sup> of the vehicle and the environment,
3. update the environment model through the vehicle's **environmental sensors**, and
4. update the vehicle's model through the vehicle's estimator.

---

<sup>4</sup>These are regions which the vehicle may not enter.



**Figure 1.1** – Shows the architecture of the entire autonomous navigation project. Source: C.E. van Daalen

This project's focus concerns the algorithms that will solve the problems presented by the **motion planning module** in Figure 1.1. These specific problems are solved by a class of algorithms referred to as motion planning algorithms, and a discussion on the different algorithms is presented in Chapter 3.

The work in this project uses a simplified deterministic conflict detection module, which doesn't calculate probabilities of conflict as done by C.E. van Daalen [4]. It is seen in Figure 1.1 that a different conflict detection module (provided that it returns results in a similar manner) can be used without changing the motion planning module.

A secondary focus of this project is the execution of planned paths using vehicle controllers. Vehicle controllers are already built in previous projects, and are therefore only briefly discussed in the next chapter (Chapter 2). In the next section an overview of this thesis is presented.

### 1.3 Thesis overview

**Chapter 2** The vehicle for which the autonomous navigation system is being built is presented, along with the axis system definition used throughout this document.

**Chapter 3** This project's problem statement is presented, and the motion planning approaches used in this project are introduced.

**Chapter 4** The implementations of the PRM and RRT\* motion planning algorithms are discussed.

**Chapter 5** Theoretical analyses of the PRM and RRT\* motion planning algorithms are conducted, and compared to practical measurements.

**Chapter 6** A discussion on the effect of introducing path replanning for the RRT\* algorithm is presented.

**Chapter 7** The Simulink simulation used to verify the validity of several theoretical aspects of this project is presented, along with several results.

**Chapter 8** Different example environments are presented, along with practical measurements of the performance of the PRM and RRT\* algorithms.

**Chapter 9** The conclusion of this document, wherein the achievements of this thesis are briefly presented.

## 1.4 Conclusion

In this chapter the architecture of an autonomous navigation system is presented together with the modules necessary to build the entire system. The chapter then highlights that this project's aim is to solve the problems presented by the motion planning module. These problems will be solved using a class of algorithms called motion planning algorithms, and in Chapter 3, a discussion of these algorithms is presented. The next chapter (Chapter 2) concerns the work already completed on the vehicle, along with this document's axis system definition.

## Chapter 2

# Vehicle Model and Controllers

In the previous chapter the relevance of autonomous UAVs are discussed, and a system architecture for achieving autonomous navigation is presented. The vehicle, for which the autonomous navigation system is being built will now be introduced, together with previous work already completed on the vehicle.

This chapter starts by defining the axis reference frame (Section 2.1) used in this document, then continues by presenting the chosen vehicle (Section 2.2). For the chosen vehicle, the available actuators (Subsection 2.2.2), the model of the vehicle (Subsection 2.2.3), the developed controllers of the vehicle (Subsection 2.2.4), and the vehicle estimator (Subsection 2.2.5) are presented.

### 2.1 Axis System Definition

#### 2.1.1 Earth Reference Frame

The origin (denoted by  $O$  in Figure 2.1) is chosen to coincide with the starting position of the vehicle, and remains earth-fixed. The axis components for the earth reference frame is defined as follows: the  $OX_E$ -axis points to the North pole and is tangential to the earth's surface. The  $OY_E$ -axis points to the East, also tangential to the earth's surface. The  $OZ_E$ -axis is perpendicular to both the  $OX_E$  and  $OY_E$  axes, and points down towards the centre of the earth. The position offset components are defined as North, East, and Down (NED displacement). This is shown in Figure 2.1.

Throughout this document, the  $OX_E$ -axis is also referred to as the x-axis, the  $OY_E$ -axis referred to as the y-axis, and the  $OZ_E$ -axis referred to as the negative of the z-axis.

#### 2.1.2 Body Reference Frame

The reference frame of the body of the vehicle is fixed to the body of the vehicle and has its origin at the centre of gravity (CG) of the vehicle. Figure 2.2 shows the reference frame, as well as definitions used throughout this document. The axis components are defined as follows: the  $OX_B$ -axis points forwards towards the nose of the aircraft, the  $OY_B$ -axis is perpendicular to the  $OX_B$ -axis and points to the right of the aircraft, and the  $OZ_B$ -axis is perpendicular to the  $X_B Y_B$ -plane and points downwards.

Furthermore, the vehicle **rolls** around the  $OX_B$ -axis, **pitches** around the  $OY_B$ -axis, and **yaws** around the  $OZ_B$ -axis, where the respective angles relative to the earth reference frame are introduced in the next subsection. The position ( $x$ ,  $y$ , and  $z$ ) place the vehicle CG at a

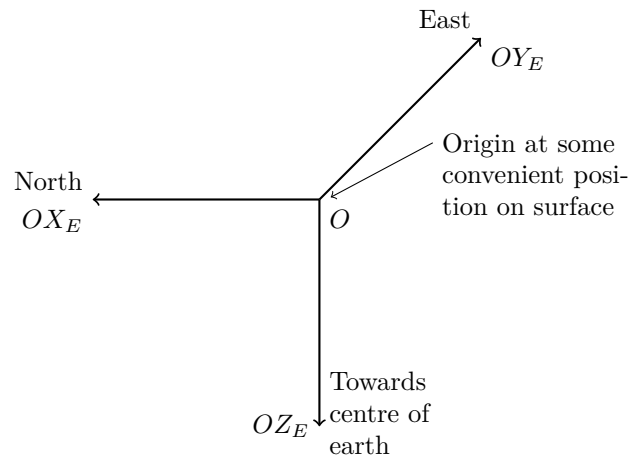


Figure 2.1 – Earth Axis System Definition.

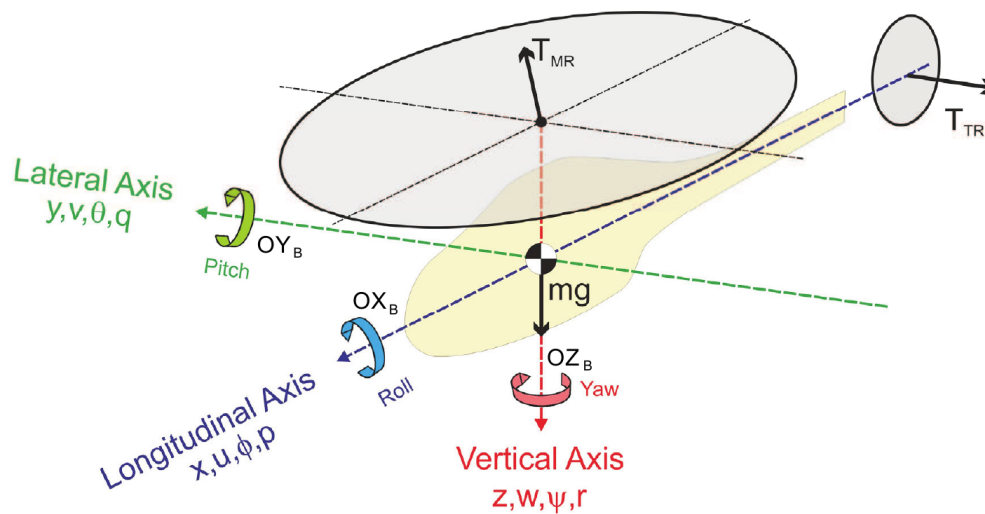


Figure 2.2 – Aircraft Body Axis System Definition [1].

specific location relative to the earth reference frame. The velocity ( $u$ ,  $v$ , and  $w$ ) describe the motion of the CG in body coordinates. The angular rates ( $p$ ,  $q$ , and  $r$ ) are defined in body coordinates.

The vector  $\mathbf{T}_{MR}$  shows the force generated by the main rotor of the vehicle, and  $\mathbf{T}_{TR}$  shows the force generated by the tail rotor.

### 2.1.3 Reference Frame Relationship

This subsection discusses the relationship between the earth and body reference frames. The body reference frame is defined to be fixed to the body of the vehicle, and earth reference frame is defined to be fixed to the earth. To determine the vehicle's position and attitude within the earth reference frame, it is important to know how the body reference frame is spatially oriented relative to the earth reference frame. Euler angles are used in this document to describe this orientation, and the transformation from one reference frame to another is achieved through three rotations.

This document uses the Euler 3-2-1 order of rotation, meaning that the transformation from the body to the earth reference frame is performed by:

1. first rotating the body reference frame about the  $OZ_E$ -axis, through the yaw angle  $\psi$ ,
2. then about the  $OY_E$ -axis, through the pitch angle  $\theta$ , and
3. lastly about the  $OX_E$ -axis, through the roll angle  $\phi$ .

For a vector  $\mathbf{V}_B$  in the body reference frame and a Direct Cosine Matrix<sup>1</sup> (DCM)  $\mathbf{T}(\phi, \theta, \psi)$ , the same vector located in the earth reference frame  $\mathbf{V}_E$  is related by the following:

$$\mathbf{V}_B = \mathbf{T}(\phi, \theta, \psi) \cdot \mathbf{V}_E, \quad (2.1.1)$$

$$\mathbf{V}_E = \mathbf{T}^T(\phi, \theta, \psi) \cdot \mathbf{V}_B. \quad (2.1.2)$$

The Euler 3-2-1 DCM is defined as:

$$\mathbf{T}(\phi, \theta, \psi) = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin(\phi\sin(\theta\cos\psi - \cos\phi\sin\psi) & \sin(\phi\sin(\theta\sin\psi + \cos\phi\cos\psi) & \sin(\phi\cos\theta \\ \cos\phi\sin(\theta\cos\psi + \sin(\phi\sin\psi) & \cos\phi\sin(\theta\sin\psi - \sin(\phi\cos\psi) & \cos\phi\cos\theta \end{bmatrix} \quad (2.1.3)$$

## 2.2 The Vehicle

This section provides an overview of the vehicle upon which the research for this project is conducted. Firstly, the vehicle is presented, and its available actuators and their limitations are introduced. Thereafter, the model of the vehicle is presented together with the states present in the model, as well as an introduction to the model used during simulation. Lastly, the vehicle's available controllers, and its estimator are presented.

### 2.2.1 Vehicle choice

In a previous project at Stellenbosch University, research was done by S. Groenewald [1] to choose a RC helicopter to be used in subsequent projects at the university. The vehicle chosen by Groenewald is the **X-Cell Fury** .60 Expert (fitted with a .70 size engine), and the work done in this project will be based on it. The X-Cell helicopter is a methanol powered aircraft with a .70 size glow<sup>2</sup> engine, capable of lifting a maximum payload of 2kg<sup>3</sup>. Throughout this document the X-Cell will be referred to as the vehicle. A photograph of the vehicle during autonomous hover is shown in Figure 2.3.

<sup>1</sup>For a complete derivation of the DCM see [5].

<sup>2</sup>Glow fuel consists of Methanol as base, and is usually mixed with Synthetic Oil and Nitro Methane.

<sup>3</sup>With a set of 680mm rotor diameter blades.

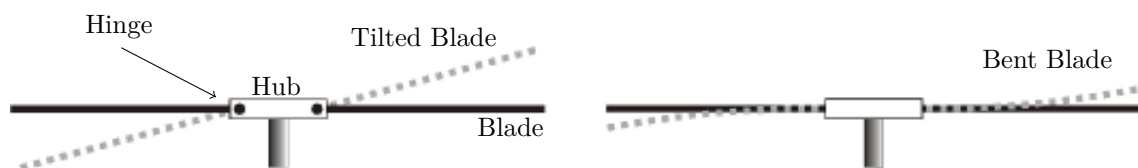


**Figure 2.3** – The X-Cell during autonomous hover [1].

The next subsection presents the actuators of the vehicle.

### 2.2.2 Vehicle actuators

To control the vehicle, actuators are used to influence the force vectors ( $\mathbf{T}_{MR}$  and  $\mathbf{T}_{TR}$ ) generated by the main and tail rotors, as shown in Figure 2.2. To understand how this works, it is first necessary to look at how the rotor blades of a helicopter are connected to the helicopter itself. Large helicopters usually have a hinged rotor hub, as shown on the left of Figure 2.4, where the rotor blade is connected to the hub of the helicopter via a hinge [6]. The hinges allow the tip path plane<sup>4</sup> (TPP) to change its orientation, and therefore change the direction of the force vector produced at the hub. For small helicopters (such as RC helicopters) the blade and hub provides a similar function, except that the hub typically does not have hinges.



**Figure 2.4** – On the left a hinged rotor hub is shown, and on the right a rigid rotor hub is shown [1].

The angle that the tip of a blade makes with the hub, along the lateral or longitudinal axis, is called the flapping angle<sup>5</sup>, and is denoted by  $a_1$  and  $b_1$  respectively. The control system

<sup>4</sup>The tip path plane is the path the tip of the rotor blade traces as it rotates around the hub.

<sup>5</sup>As noted by S. Groenewald [1],  $a_1$  and  $b_1$  actually denotes the first harmonics of the blade flapping angles.



of the vehicle is able to control these angles with the cyclic input commands,  $\delta_a$  and  $\delta_b$ . The collective angle of attack of the blades can be varied by  $\delta_c$ , which controls the magnitude of the thrust. These control inputs are transferred to the main rotor by using cyclic collective pitch mixing (CCPM). Typically, three servos are used to control the swashplate, which forces the TPP to trace the required path. The last control input is the collective pitch of the tail rotor, which adjusts the thrust of the tail rotor, and is denoted by  $\delta_r$ . These inputs control all the degrees of freedom of the vehicle.

The next subsection provides details about the model of the vehicle.

### 2.2.3 Vehicle Model

A nonlinear hover model for low advance ratio flight was developed by E. Rossouw [7]. The vehicle states in the model are described in Table 2.1:

| State          | Description   |
|----------------|---|
| $u, v, w$      | Velocity components in the x,y and z-directions of the body frame |
| $p, q, r$      | Roll-, pitch-, and yaw rates of the body reference frame          |
| $\phi, \theta$ | Roll-, and pitch angles relative to the earth reference frame     |
| $a_1, b_1$     | First harmonics of the blade flapping angles                      |

**Table 2.1** – Summary of the vehicle states included in its model.

#### Relation of actuators and states

Now that the states of the vehicle model are known, it is possible to qualitatively describe the influence the actuators have on the states of the vehicle model.

The input  $\delta_a$  controls the **lateral** flapping angle  $a_1$ , which changes the force vector  $\mathbf{T}_{MR}$  about the **longitudinal-axis** of the body frame, and results in acceleration to the **left or right** of the vehicle. Intuitively, the input  $\delta_a$  influences the lateral velocity  $v$ , the roll angle  $\phi$ , and the roll rate  $p$ .

The input  $\delta_b$  controls the **longitudinal** flapping angle  $b_1$ , which changes the force vector  $\mathbf{T}_{MR}$  about the **lateral-axis** of the body frame, and results in acceleration to the **front or rear** of the vehicle. Intuitively, the input  $\delta_b$  influences the longitudinal velocity  $u$ , the pitch angle  $\theta$ , and the pitch rate  $q$ .

The input  $\delta_c$  controls the collective angle of attack of the main rotor, and therefore changes the magnitude of the force vector  $\mathbf{T}_{MR}$ <sup>6</sup>. The states that  $\delta_c$  effects will depend on  $a_1$  and  $b_1$ , however it can potentially influence  $v, u, \phi, \theta, p$ , and  $q$ . In addition to the states dependant on  $a_1$  and  $b_1$ ,  $\delta_c$  will mainly influence the vertical velocity  $w$ , and the yaw rate  $r$ .

The input  $\delta_r$  controls the angle of attack of the tail rotor, which changes the magnitude of the force vector  $\mathbf{T}_{TR}$ . Intuitively, the input  $\delta_r$  influences the lateral velocity  $v$ , and the yaw rate  $r$ .

#### Definition of the non-linear model

The non-linear model of the vehicle is defined by Rossouw as follows:

<sup>6</sup>Note that a governor is used to keep the rotor at a fixed speed.

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u}), \quad (2.2.1)$$

with:

$$\mathbf{x} = [u \ q \ \theta \ a_1 \ v \ p \ \phi \ b_1 \ w \ r]^T, \quad (2.2.2)$$

$$\mathbf{u} = [\delta_a \ \delta_b \ \delta_c \ \delta_r]^T. \quad (2.2.3)$$

To develop the feedback controllers for the vehicle, Rossouw linearised the non-linear model around hover trim conditions using MATLAB.

### Linearised hover model

The linearised hover model is defined as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad (2.2.4)$$

where the linearised  $A$  and  $B$  matrices are:

$$A = \begin{bmatrix} -0.036 & 0 & -g & -9.55 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.001 & 0 & 0 & 203.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.002 & -1 & 0 & -8.39 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.13 & 0 & g & 9.55 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.16 & 0 & 0 & 383.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.002 & -1 & 0 & -8.39 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.11 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -23.37 & 0 \end{bmatrix}, \quad (2.2.5)$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 35.31 & 0 & 0 \\ 0 & 0 & 1.28 & -6.77 \\ 0 & 0 & 11.23 & -20.15 \\ 0 & 0 & 0 & 0 \\ 35.31 & 0 & 0 & 0 \\ 0 & 0 & -164.5 & 0 \\ 0 & 0 & 0 & 147.3 \end{bmatrix}. \quad (2.2.6)$$

In the B-matrix four cross-coupling terms are identified: 1.28, 11.23,  $-6.77$ , and  $-20.15$ , which are on the  $\delta_c$  and  $\delta_r$  input commands. The rudder creates a side force ( $-6.77$ ), however the roll cyclic cancels this quickly through feedback. The rudder could also cause a roll moment ( $-20.15$ ) if it is offset from the body x-axis, however this offset is small enough for the effect to be negligible. Side force from the collective (1.28) arises due to the roll angle the helicopter hangs at during hover. Similarly, a roll moment (11.23) will result if the tail rotor is offset such that the main rotor needs to provide a steady state moment in trim<sup>7</sup>.

The above mentioned effects can be reasoned as small, therefore Rossouw reasoned that the cross coupling terms may be neglected and thereby the full helicopter model could

<sup>7</sup>The above insights regarding the cross coupling terms are provided by Dr. Iain K. Peddle.

be decoupled into four separate models, which is advantageous since a separate feedback controller could be designed for each model.

The helicopter model is decoupled into separate longitudinal (Figure 2.5), lateral (Figure 2.6), heave (Figure 2.7), and heading (Figure 2.8) models.

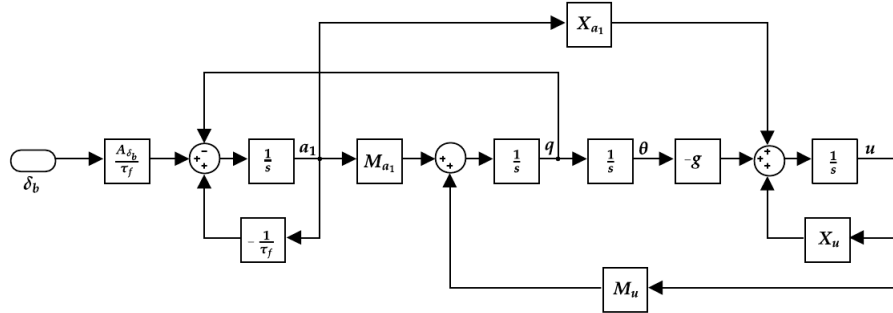


Figure 2.5 – The longitudinal model of the vehicle [7].

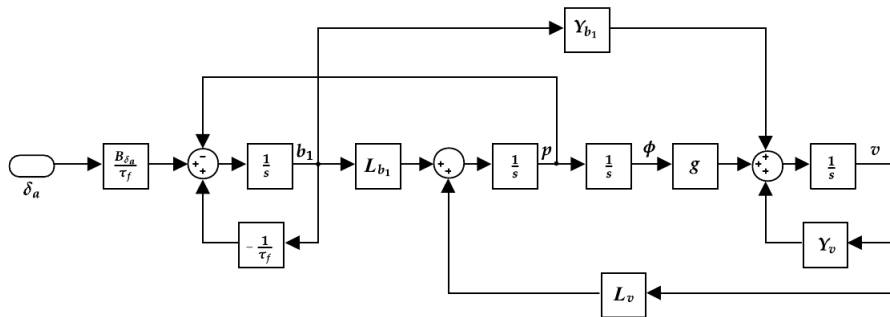


Figure 2.6 – The lateral model of the vehicle [7].

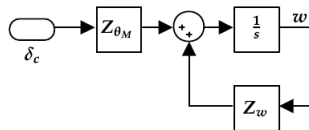


Figure 2.7 – The heave model of the vehicle [7].

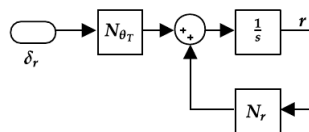


Figure 2.8 – The heading model of the vehicle [7].

The next subsection presents the controllers developed for the vehicle.

## 2.2.4 Vehicle controllers

The control method used by Rossouw is successive-loop-closure, which has been widely used to automate unmanned helicopters. The method works by first controlling the fast modes of motion of the system by closing control loops with the dominant states that influence these modes. The slower dynamics are then controlled by successive outer loops.

### 2.2.4.1 Heading angle controller

The first controller designed by Rossouw is the heading angle controller. It should be noted that the gyro of the vehicle is operated in **normal** mode, not heading hold mode. The design specifications of the heading angle controller are:

- rise time of less than 3s,
- overshoot of less than 20%, and
- zero steady state errors.

Since zero steady state errors are required, proportional-integral (PI) control is used by Rossouw. Figure 2.9 shows the successive-loop-closure block diagram of the heading controller. A heading angle step response is shown in Figure 2.10.

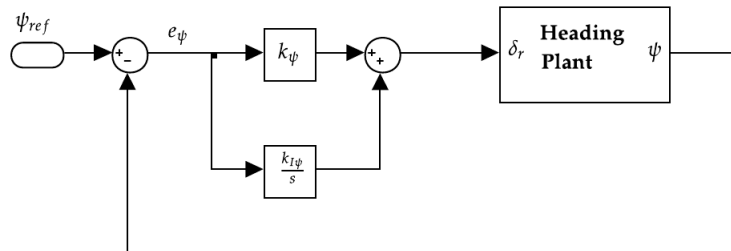


Figure 2.9 – Successive-loop-closure for the heading plant [7].

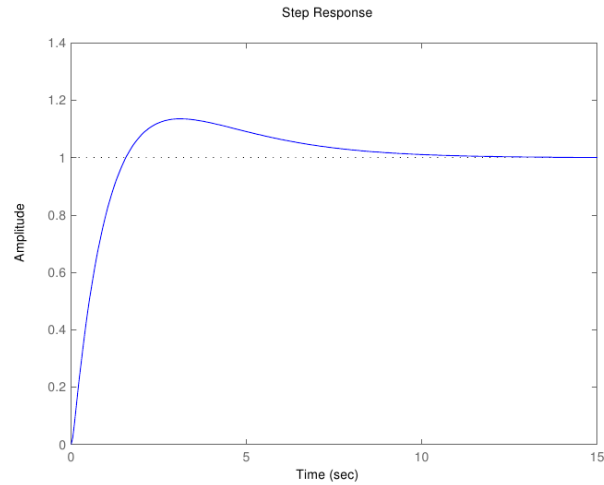


Figure 2.10 – Heading angle step response [7]. The amplitude is given in radians.

### 2.2.4.2 Heave position controller

Next, the heave position controller is designed by Rossouw. The design specifications of the heave position controller are:

- rise time of less than 3s,
- overshoot of less than 20%, and
- zero steady state errors.

Once again, PI control is used by Rossouw to achieve zero steady state errors. Figure 2.9 shows the successive-loop-closure block diagram of the heave controller. A heave position step response is shown in Figure 2.12.

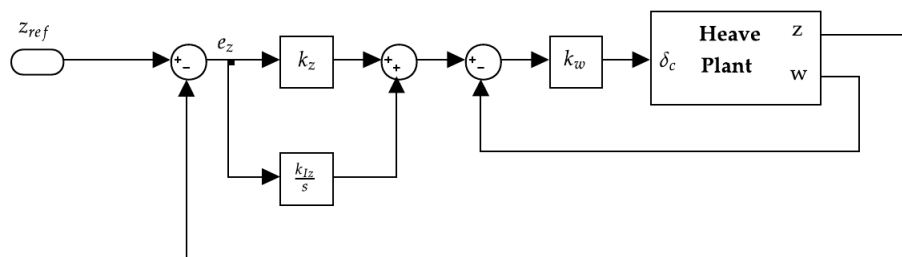


Figure 2.11 – Successive-loop-closure for the heave plant [7].

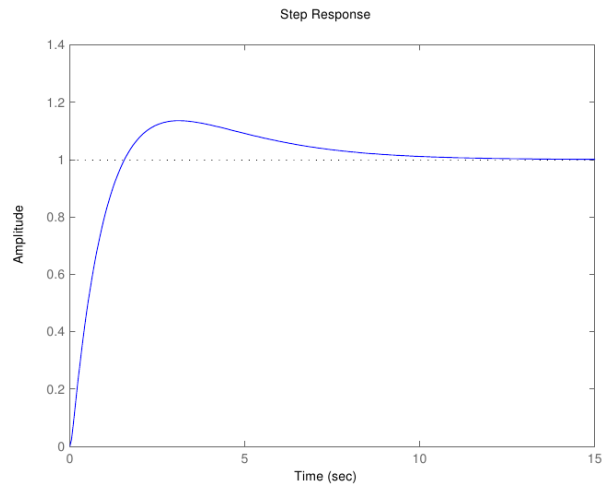


Figure 2.12 – Heave position step response [7]. The amplitude is given in meters.

### 2.2.4.3 Lateral and Longitudinal position controllers

Lastly, the lateral and longitudinal position controllers are designed by Rossouw. The specification for the horizontal position controllers are:

- rise time of less than 5s,
- overshoot of less than 10%, and
- zero steady state errors.

PI control, together with successive-loop-closure methods, are once again employed, and a resultant longitudinal position step response is shown in Figure 2.15. Figures 2.13 and 2.14 shows the successive-loop-closure block diagrams of the lateral and longitudinal controllers.

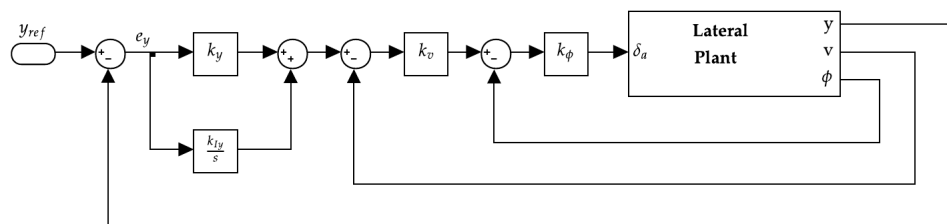


Figure 2.13 – Successive-loop-closure for the lateral plant [7].

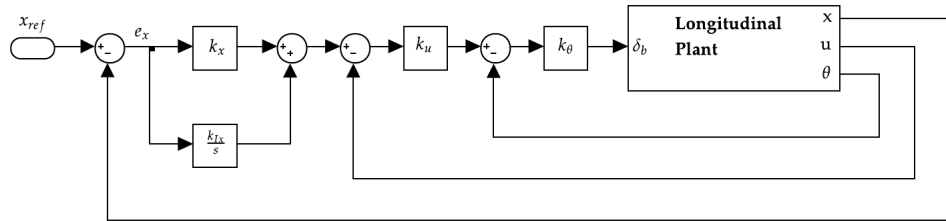


Figure 2.14 – Successive-loop-closure for the longitudinal plant [7].

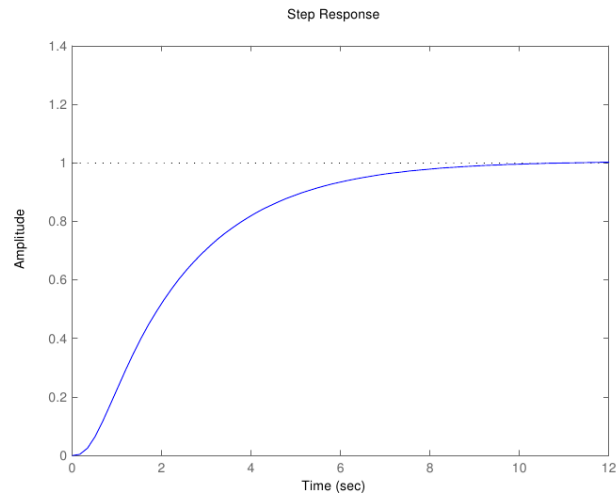


Figure 2.15 – Longitudinal position step response [7]. The amplitude is given in meters.

#### 2.2.4.4 Vehicle model and controller simulation with Simulink

The full non-linear equations for the vehicle, as developed by V. Gavrillets, B. Mettler, and E. Feron [8], were used by Medellín-Colombia [9] to create a Simulink model of the vehicle in MATLAB. The model parameters were tailored to this project's vehicle specifications, as summarised by Groenewald [1].

The Simulink model is covered in Chapter 7; Figure 7.3 shows the inner loop controllers in a Simulink diagram. Hardware implementations of these controllers have undergone flight tests, and the software implementation of them in Simulink is deemed to accurately represent their real world behaviour.

#### 2.2.5 Vehicle Estimator

A state estimator is used to obtain the translational and rotational states of a vehicle, even if the states are not directly measurable. This is achieved by combining measurements from a number of sensors and making use of the kinematic relationships between the states and measurements.

A full state kinematic estimator was designed in the ESL by W.J. Hough [2]. The estimator uses measurements from strapped down gyroscopes to determine the attitude of the vehicle. Once the attitude is known, strapped down accelerometer measurements can be

coordinated in an inertial reference frame. After gravitational accelerations is compensated for, the measurements are integrated twice to determine the platform's inertial frame, and its coordinated velocity and position.

A problem is that the integrated position, velocity, and attitude states will lose accuracy over time since an approximated integration process is used, as well as the inertial sensors having biases. However, it is possible to correct this by using measurements from the GPS and magnetometer to update the integrated states. The structure, as designed by Hough, is shown in Figure 2.16, and is implemented in the Simulink simulation.

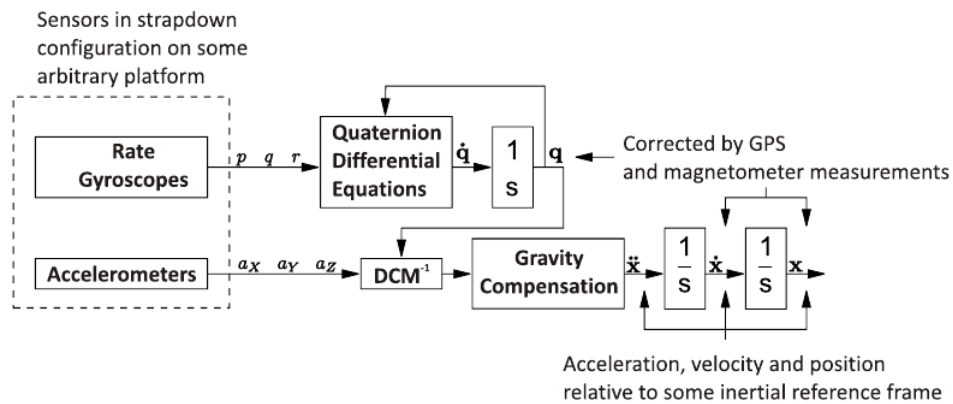


Figure 2.16 – The kinematic state estimator structure by Hough [2].

## 2.3 Conclusion

Several key aspects of the vehicle, for which the autonomous navigation system is being built, are introduced in this chapter. The chapter starts by introducing the earth and body reference frames, and accordingly the axis conventions used throughout this document. Next, this chapter introduces the vehicle used in this project as the X-Cell Fury, as well as its available actuators, and its identified model. Lastly, the available controllers and their characteristics are presented, and the vehicle's estimator is discussed.

In the next chapter (Chapter 3), the problem statement for this project is presented, along with a discussion on different motion planning algorithms, and the requirements of the motion planning module for this project.



## Chapter 3

# Motion planning

The focus of this project is on motion planning. In general, motion planning finds a path that a vehicle can execute to move from one waypoint to another. To place the motion planning module in context of the entire autonomous navigation system, Section 1.2 presents a system architecture wherein the role of the motion planning module is illustrated.

This chapter starts by defining the problem statement for this thesis (Section 3.1), after which a discussion on the different motion planning algorithms (Section 3.2) is presented.

### 3.1 Motion planning problem statement

This project concerns motion planning for a rotary wing UAV, where vehicle controllers are already in place, and map data is readily available to a conflict detection module. The basic motion planning problem is to determine how a vehicle should move so that it can reach a goal position, while avoiding obstacles. The problem statement for this thesis is listed below:

1. For given initial and goal milestones, it is necessary to find an executable and conflict free path between the initial and goal milestones.
2. This path must satisfy nonholonomic<sup>1</sup> motion, as well as kinodynamic constraints of the chosen vehicle.
3. Given that a conflict free path exists, it is necessary to guarantee that such a path will be found.
4. The environment in which the vehicle operates is uncertain, dynamic and cluttered.
5. The algorithm must be able to improve already existing paths.
6. The algorithm must be usable on practical systems, where almost real time motion planning is required.

The aforementioned problems are all solved using the motion planning algorithms that are implemented in Chapter 4. The subsequent chapters of this thesis focuses on specific points of the problem statement, and aim to provide an in depth understanding as to how the algorithms solve the above problems. Chapter 5 performs an in depth analysis on two

---

<sup>1</sup>Subsection 3.4.1 explains nonholonomic and kinodynamic constraints, and why they are relevant to this project.

motion planning algorithms, and shows that the algorithms are usable on practical systems. Chapter 6 focuses on path replanning, and shows that the algorithm is able to improve an existing path, and that the environment in which the vehicle operates may be uncertain. Chapter 7 performs a Simulink simulation, wherein it shows that the path provided by the motion planning algorithm satisfies the nonholonomic motion and kinodynamic constraints of a vehicle. Chapter 8 shows that the motion planning algorithm finds a conflict free path between given initial and goal milestones, even in cluttered environments.

The rest of this chapter (Chapter 3) discusses different motion planning algorithms, and concludes with choosing two motion planning algorithms capable of solving the above problems.

## 3.2 Available motion planning algorithms

As previously discussed, this project forms part of a larger goal to achieve autonomous flight of an unmanned rotary wing aircraft. Many building blocks are required to achieve such a goal, and for this project the motion planning, conflict resolution, and vehicle controller blocks form its focus. These blocks are presented in the architecture diagram shown in Figure 1.1. The motion planning and conflict detection blocks are responsible for solving most of the problems in Section 3.1. In this section algorithms are introduced that are suitable for solving these problems; these algorithms are known as **motion planning** algorithms.

### 3.2.1 Complete motion planning

An algorithm that solves the motion planning problem is said to be complete if it finds a path in any given environment (given that a path exists), and for environments where no path exists, it is able to determine that no path exists.

Early works by T. Lozano-Perez, M. A. Wesley [10], and J. T. Schwartz, M. Sharir [11], focused on complete motion planning algorithms for polygonal robots and obstacles using algebraic approaches. However, work by J.H. Reif [12] proved that even a basic motion planning problem, called the Piano Movers Problem, is polynomial space hard. This strongly suggests that complete motion planning algorithms are execution time expensive due to their computational complexity, which means that they are unsuitable for practical applications.

Since this project requires motion planning in (close to) real time, other solutions have to be considered. In the next subsection, algorithms that are not strictly complete are presented along with a discussion on the path finding guarantees they provide.

### 3.2.2 Almost complete motion planners

In the previous subsection it is concluded that complete planners are computationally too expensive for practical considerations. By relaxing the strict completeness constraint, it is possible to consider motion planning algorithms that still offer some form of completeness.

#### 3.2.2.1 Grid-based search

One example of an algorithm that does not provide strict completeness is the **grid-based** search algorithm, which guarantees **resolution completeness**, that is, with a fine enough grid resolution, a solution to the motion planning problem will be found, if one exists [13].

Grid-based search algorithms overlay a grid on the configuration space and associate robot or vehicle configurations to the points on the grid. Vehicles can move to adjacent grid points only if the line connecting the two points is entirely conflict free.

Discretising the configuration space means that for rough resolutions of the grid, a path through narrow parts of the free configuration space might not be found. This is a problem, because for very fine resolutions, or high dimensions of the configuration space, this algorithm becomes computationally very expensive to execute. This is due to the exponential growth of points in the grid that is necessary to cover the entire configuration space.

### 3.2.2.2 Potential fields

One approach used in **potential field** algorithms is to create a potential field that attracts towards the goal, and repulses from obstacles. The vehicle is then a point in this potential field, which is attracted towards the goal and repulsed from obstacles.

The main problem with potential field based algorithms is getting stuck in a local minima, and not being able to escape [13]. However, recent advances to the potential field algorithm was made by S. Ansari, K. Ok, B. Gallagher and W. Sica [14], where Voronoi vertexes are employed to solve the local minima problem. The improved algorithm is said to be complete, however formal proof thereof is not shown by Ansari et al. Furthermore, the Voronoi vertex generation is computationally very expensive and in Table 1 by Ansari et al., execution times exceeding 9 seconds are necessary with many (in this case 500) obstacles present in the environment.

### 3.2.2.3 Sampling based

**Sampling based** motion planning algorithms arguably present the most influential recent advances, as is seen from the amount of research done by E. Frazzoli, M.A. Daleh, and E. Feron [15], D. Hsu, R. Kindel, J.C Latombe, and S. Rock [16], S. Karaman, and E. Frazzoli [17], L. Kavraki, and J. Latombe [18], L.E. Kavraki, P. Svestka, J Latombe, and M.H. Overmars [19], S.M. LaValle, and J.J. Kuffner [20], J. vd. Berg, D. Ferguson, and J.J. Kuffner [21], and more. These algorithms typically generate uniform samples (or milestones) over the entire configuration space, and then try to add these milestones to a tree of milestones which is rooted to the initial milestone, or start state of the vehicle.

The Rapidly Exploring Random Tree (RRT) and Probabilistic Roadmap Method (PRM) are examples of sampling based motion planning algorithms, and formal proofs of **probabilistic completeness** exist for both the RRT and PRM algorithms. Generally speaking, probabilistic completeness means that the probability of finding a path, given that a path exists, will approach certainty by generating more and more milestones.

The RRT and PRM motion planning algorithms are widely used, and real time motion planning is achievable for both algorithms. Overviews of the histories, and major contributions to the PRM and RRT algorithms are presented in Subsections 3.3.1, and 3.3.2.

### 3.2.2.4 Conclusion

Following this brief discussion, it is decided that the PRM and RRT\*<sup>2</sup> algorithms will be used in this project. In the next section an overview of the research done on the PRM and RRT\* algorithms is presented.

---

<sup>2</sup>A recent paper [17] made huge improvements to the classic RRT. The improved algorithm is now referred to as the RRT\*.

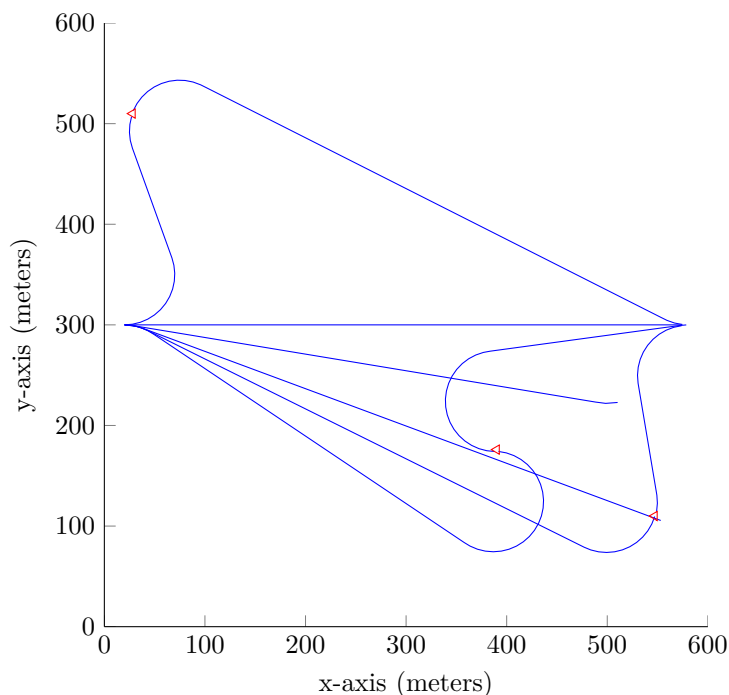
### 3.3 Sampling based motion planning

In this section, developments regarding the PRM and RRT\* sampling based algorithms are presented.

#### 3.3.1 The Probabilistic Roadmap Method

The Probabilistic Roadmap Method (PRM), is introduced in a paper by L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars [19] in 1996. The PRM works in two phases: a learning and a query phase. In the learning phase, a probabilistic roadmap is constructed of conflict free vehicle configurations, where the edges of the roadmap corresponds to feasible paths between the configurations. Next, the roadmap is queried for a path between an initial and goal configuration, and a path is returned that connects the initial and goal vehicle configurations. Perhaps the biggest advantage the PRM offered, when compared to previous planners, is that the time required by the PRM to find a solution does not increase exponentially in high dimensional configurations spaces.

Since its conception, the PRM has seen a multitude of papers published suggesting differing strategies for connecting vehicle configurations, as well as several sampling strategies<sup>3</sup>. One major difference between the classical PRM and the PRM implemented for this paper is the fusing of the learning and query phases. The implementation of the PRM used in this paper is presented in Section 4.8. Figure 3.1 shows a typical PRM with several milestones in its roadmap.



**Figure 3.1** – This figure shows the PRM algorithm with several milestones in its tree.

<sup>3</sup>The PRM uses a sampling strategy while it is constructing its roadmap.

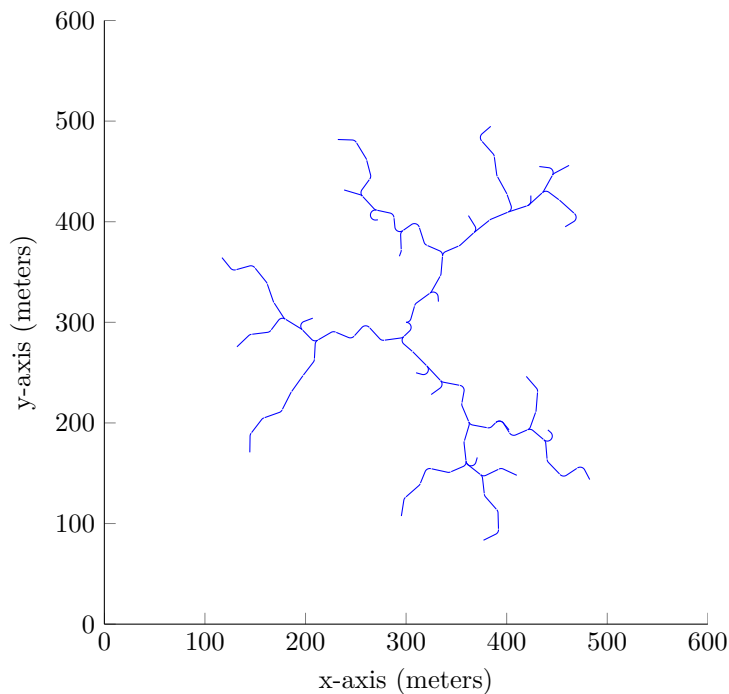
A theorem presented by D. Hsu, R. Kindel, J.C. Latombe, and S. Rock [16] proves that the PRM is guaranteed to find a solution (given that one exists) with increasing probability, as the number of known milestones<sup>4</sup> increases. This is referred to as probabilistic completeness, and is arguably the single most important contribution to the PRM. Building on this, Section 5.2.2 uses parts of the proof of probabilistic completeness to obtain insight into the workings of the PRM in a specific environment.

In the next subsection, a second sampling based motion planning algorithm is introduced, the Rapidly exploring Random Tree.

### 3.3.2 The Rapidly exploring Random Tree

The Rapidly exploring Random Tree (RRT) algorithm was first introduced by S.M. LaValle in 1998 [22]. The RRT was originally designed to handle nonholonomic, dynamic constraints in search spaces with high degrees of freedom. The RRT algorithm has been widely used for practical applications and a variant thereof was proposed for use on the Mars rovers by J. Kwak [23].

The classic RRT algorithm expands by iteratively applying control inputs that drive a system only slightly in the direction of randomly selected points, whereas the PRM requires point to point convergence. S.M. LaValle notes that at the introduction of the RRT, it was already capable of dealing with holonomic, nonholonomic, and kinodynamic planning problems of up to twelve degrees of freedom. The classic RRT is shown in Figure 3.2, with the difference that manoeuvres (discussed in Subsection 3.4.2), instead of control inputs, are used to drive the system in the direction of points, or milestones.



**Figure 3.2** – This figure shows the classic RRT algorithm after 100 iterations.

---

<sup>4</sup>For the definition of a milestone, refer to Subsection 4.6.1. A milestone is essentially a waypoint with some added data members.

In 2001, S.M. LaValle and J.J. Kuffner [20] presented a mathematical proof that the RRT guarantees probabilistic completeness. This is arguably the single most important feature of the RRT, as it now guarantees the probability of finding a path, if one exists, strives to certainty, as the number of iterations increases. Parts of the proof of probabilistic completeness is used to gain insight into the workings of the RRT in a specific environment, in Section 5.3.1.

In 2010, S. Karaman and E. Frazzoli [17] made significant improvements to the RRT. Karaman et al. addresses issues of the RRT such as the quality of the path it finds. The first contribution from the authors is mathematical proof that the cost of the best path in the RRT converges almost surely to a non-optimal value. This is a negative result for the RRT as it means that when a RRT is used for motion planning the path, it finds will almost surely be non optimal, and better paths might be found with another algorithm.

The second contribution of Karaman et al. is a new algorithm called the Rapidly-exploring Random Graph (RRG), for which it is proven that the cost of the best path in the RRG converges to the optimum almost surely. Next, the authors introduce a tree version of the RRG, referred to as the RRT\*. The RRT\* maintains the asymptotic optimality of the RRG, while maintaining a tree structure like the RRT. Lastly, Karaman et al. also shows that the computational complexity of the RRG and RRT\* algorithms are asymptotically within a constant factor of the RRT.

The RRT\* algorithm by Karaman et al. does not deal with several requirements of this project's problem statement; therefore the next section discusses these requirements.

### 3.4 Problem statement requirements

The problem statement for this project requires motion planning in uncertain, dynamic environments for vehicles with nonholonomic and kinodynamic constraints. In the previous sections the available motion planning options are briefly discussed, and their respective weaknesses and strengths are presented. Accordingly, it is decided that sampling based algorithms are best suited for this project, specifically the PRM and RRT\* motion planning algorithms.

The PRM algorithm has been implemented with nonholonomic and kinodynamic constraints in previous projects, however the RRT\* algorithm proposed by Karaman et al. [17] use straight lines to connect milestones, thus not taking nonholonomic nor kinodynamic constraints into account. In this section, various requirements of the problem statement are discussed, and the approaches used in this project to adhere to these requirements are introduced.

#### 3.4.1 Kinodynamic and nonholonomic motion constraints

##### Kinodynamic

The kinodynamic planning problem refers to robotic (or vehicle) motion that is subject to simultaneous kinematic and dynamic constraints, as well as avoiding obstacles. Constraints on vehicle dynamics includes bounds on velocity, acceleration and force [24]. Since this project concerns planning for a real world vehicle, it is important to adhere to the aforementioned constraints. Furthermore, the control inputs of the vehicle are via actuators, which also impose kinodynamic constraints.

Obstacle avoidance is achieved via the conflict detection algorithm outlined in Subsection 3.4.3. The vehicle's dynamic constraints are adhered to by placing bounds on the allowable vehicle velocity during planning.

### Nonholonomic

A nonholonomic system, in this project's context, refers to a system where the **controllable** degrees of freedom are less than the **total** degrees of freedom of the vehicle. The controllable degrees of freedom of the vehicle are  $\mathbf{u} = [\delta_a \ \delta_b \ \delta_c \ \delta_r]^T$ , as opposed to the states in the model  $\mathbf{x} = [u \ q \ \theta \ a_1 \ v \ p \ \phi \ b_1 \ w \ r]^T$ . This means that the system is path dependent: the heading at which the vehicle arrives at any point is also the heading at which the vehicle must leave the point. These constraints are adhered to in this project through using manoeuvres, outlined in Subsection 3.4.2.

In Chapter 7, a Simulink simulation is performed using simulated vehicle controllers, which have been used during real world flight of the vehicle. The simulation simulates the vehicle executing a path generated by the motion planning algorithm. It is seen that the vehicle is able to execute a planned path while staying in its envelope<sup>5</sup> of operation.

### 3.4.2 Manoeuvres

From the previous subsection it is apparent that the motion planning required in this project should adhere to nonholonomic bounds. This requirement is addressed by using manoeuvres to determine a flight path of a vehicle. Informally, a manoeuvre consists of multiple trajectories (or manoeuvre primitives), where each of these manoeuvres are predetermined to be executable by a vehicle.

Take for example a manoeuvre that consists of two manoeuvre primitives, a left turn and a straight line. The left turn manoeuvre primitive is constructed by determining the minimum turning circle for the vehicle (at a specified forward velocity), and then executing such a turn. Next, it is necessary to determine the vehicle manoeuvre primitive for transitioning between a left turn and straight line. It is now possible to connect the left turn manoeuvre primitive to the straight line manoeuvre primitive by prepending this transition manoeuvre primitive to the straight line manoeuvre primitive. Together, this forms a single manoeuvre in the manoeuvre library, which is the set of all executable manoeuvres by the vehicle. The reader is referred to Section 4.5 where the implementation of manoeuvres is discussed.

An example where manoeuvres are used is by E. Frazzoli, M.A. Dahleh, and E. Feron [25], where control of a small helicopter was developed though using manoeuvres. Frazzoli et al. notes that through the quantisation of system dynamics, the computational complexity of the motion planning problem for non-linear, high dimensional systems is reduced in the sense that the feasible nominal trajectories are restricted to a family of time parametrised curves (a **manoeuvre library**) which is constructed by the interconnection of appropriately defined primitives to form manoeuvres. These primitives will then constitute multiple manoeuvres from which the nominal trajectories may be constructed, and these nominal trajectories are then stored in a manoeuvre library. Instead of solving an optimal control problem over a high-dimensional, continuous space, the problem is reduced to a mixed integer programming problem, over a much smaller space.

Formally, a manoeuvre is defined in this project as a finite time transition from the vehicle's hover (zero translation in the horizontal and vertical directions) state to one of its manoeuvre library<sup>6</sup> entries. While the manoeuvre is executed, the vehicle maintains an average velocity

<sup>5</sup>The envelope of operation refers to the inherent constraints a vehicle has while moving. In particular, the kinodynamic and nonholonomic motion constraints are referred to in this context.

<sup>6</sup>The reader is referred to Section 4.5.

as required by a time constraint.

### 3.4.3 Conflict detection

In the problem statement it is required that a planned path is conflict free. To achieve this, information about the environment is necessary. The only information the motion planning algorithms have about the environment is provided by the conflict detection module. The conflict detection module is therefore the only part of the motion planning algorithm that requires a description of the environment.

For this project, it is assumed that all information needed by a conflict detection module is readily available, and that there are no uncertainties regarding obstacle locations. This means that all information about the environment, i.e. future obstacle locations etc., must be known beforehand. C.E. van Daalen [4] developed a novel probabilistic conflict detection method that determines a tight upper bound on the probability of conflict, and the conflict detection method therein is shown to be executable in real time.

The implementation of a probabilistic conflict detection module is considered outside the scope of this project. However, since the conflict detection module is the only part of the motion planning algorithm that requires information about the environment, replacing it with a probabilistic conflict detection module will enable the motion planning algorithm to deal with uncertainties in the environment.

### 3.4.4 Cost function

A requirement for this project is that an algorithm must be able to improve already existing paths. To achieve this, it must be capable of comparing two different paths that reach the same point, and then determine which path is preferable. For this project, the path which is more energy effective is the preferable path. This project uses manoeuvres to connect subsequent milestones, and therefore it is necessary to determine for each manoeuvre primitive the energy cost of executing that particular manoeuvre.

The cost of executing a circle segment  $mp_{lt}$  (manoeuvre primitive left turn) or  $mp_{rt}$  (manoeuvre primitive right turn) is relatively easy to determine:

1. Determine energy cost ( $EC$ ) of executing a full circle turn.
2. Determine the fraction ( $CF$ ) the segments  $mp_{lt}$  or  $mp_{rt}$  is of a full circle.
3. Cost of executing  $mp_{lt}$  or  $mp_{rt}$  is  $EC \times CF$ .

Determining the cost of executing a manoeuvre (consisting of multiple primitives) is a bit more complicated. Since a manoeuvre will always consist of a circle segment, a line segment, and then optionally another circle segment, a circle segment is always executed before a line segment. Thus, in order to calculate the cost of executing a manoeuvre consisting of a circle segment and a line segment, the cost of transitioning between the circle and line segments may be added to the line execution cost. This realisation is important since the line segments are not flat in the xy plane, as opposed to the circle segments: they have heave angles in the z dimension. That is, executing line segments will have different execution costs, depending on their respective heave angle, because a steeper heave angle will require more energy. Transition costs between circle and line segments with steeper heave angles will also differ.

Let  $EC$  be the circle execution cost,  $TCLC(\text{heave angle})$  the transition from circle to line cost,  $LC(\text{heave angle})$  the line execution cost, and  $LTC(\text{heave angle})$  be the combined



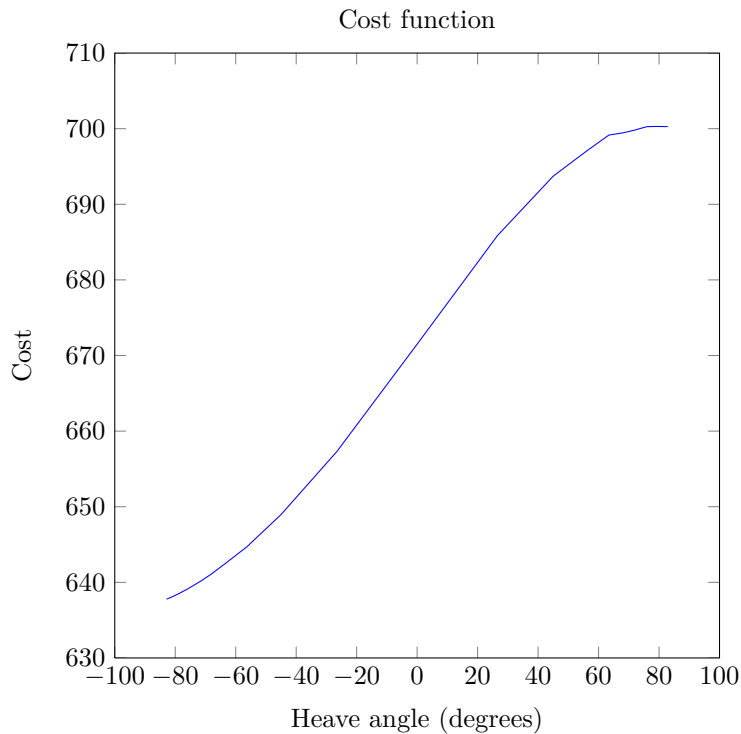
$TCLC(\text{heave angle})$  and  $LC(\text{heave angle})$  cost. It is then possible to formulate:

$$\text{Cost} = EC \times CF + TCLC(\text{heave angle}) + LC(\text{heave angle}), \quad (3.4.1)$$

$$= EC \times CF + LTC(\text{heave angle}). \quad (3.4.2)$$

As can be seen from Equation 3.4.2 both  $TCLC(\text{heave angle})$  and  $LC(\text{heave angle})$  are linearly dependant on the heave angle, which indicates that grouping the costs is possible.

A plot of energy cost vs different (line segment) heave angles is shown in Figure 3.3. The vertical axis shows the cost  $LTC(\text{heave angle})$ , and the horizontal axis shows the heave degrees. The different costs are determined through measurements of a simulation, presented in Chapter 7. For the measurements, circle and line segments with different heave angles are executed, and the forces exerted by the main and tail rotors of the simulated vehicle model are recorded. The velocity of the helicopter is kept constant during the measurements.



**Figure 3.3** – A plot showing the cost function used to determine costs. The numbers on the vertical axis represent sample averages of the measured (during simulation) force of the main and tail rotors. The measurements are dimensionless and only provide information relative to each other.

Calculating the cost of executing a manoeuvre is then achieved by using Equation 3.4.3:

$$\text{Cost} = CC \times CF + LTC(\text{heave angle}) \times LF, \quad (3.4.3)$$

where  $LF$  is the line fraction executed. It should be noted that this calculation is only valid at a specific velocity. In the implementation several samples at different velocities are used and interpolated to enable the calculation of manoeuvre costs at different velocities.

## Chapter 4

# Algorithm implementation

### 4.1 Introduction

For this project, the problem statement presented in Section 3.1 has to be solved. This chapter (Chapter 4) provides solutions to key parts of the problem statement. In the previous chapters, available options to solve the problems are discussed, and it is concluded therein that sampling based motion planning algorithms are to be used. It is also seen that there are two sampling based algorithms that are of particular interest since work already done on them closely aligns with solving the problem statement's problems, especially the PRM. The other algorithm, the RRT\*, does not take nonholonomic constraints into account during the planning process. The implementation by S. Karaman, and E. Frazzoli [17] does not use manoeuvres to connect subsequent milestones.

Section 4.3 provides an introduction to the working of a sampling based algorithm. In Section 4.4 the sampling strategy used by both the PRM and RRT\* is discussed, and in Section 4.5 the manoeuvre libraries used by the PRM and RRT\* are introduced. Next, Section 4.6 presents the data structure which is used by both the PRM and RRT\* algorithms. The chapter then continues with the conflict detection algorithm in Section 4.7, which is also used by both the PRM and RRT\*.

In Section 4.8 the PRM motion planning algorithm is presented, and in Section 4.9 the RRT\* is presented. The PRM and RRT\* algorithms are compared and analysed in Chapter 5.

### 4.2 Hardware and software specifications

All algorithms are implemented using GNU project C, following C99 standards, running in a single thread. All code is compiled with gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5). The code is executed on an AMD Athlon x4 630, capable of  $33.16/4 = 8.29$  GFLOPS per core.

### 4.3 Generic sampling based algorithm

This section presents the working of a sampling based algorithm. The algorithm is generic in that it does not contain any of the PRM or RRT\* specifics.

The sampling based class of algorithms take as input an **initial** and **goal** waypoint. The algorithm starts by using the initial waypoint as the root of a **tree**. The algorithm then **samples** a waypoint, at a random position, and connects the sample to the tree. After

the above steps have been completed, the algorithm has a tree with two vertices and one edge. The tree serves as the storage mechanism of the algorithm, each vertex in the tree corresponds to a waypoint, and each edge corresponds to a path between two vertices. Algorithm 1 shows the above steps, with the addition of repeating the sampling and connecting steps until the tree contains a vertex that can connect to the goal waypoint.

**Input:** Initial waypoint, goal waypoint  
**Output:** Path between the initial and goal waypoints  
 create tree with initial waypoint as root;  
**while** *Path to goal not found* **do**  
   | sample;  
   | connect sample to tree;  
**end**  
**return** *Path connecting the initial and goal waypoints;*

**Algorithm 1:** A generic sampling based algorithm.

The data structure used to store the tree, and all its information is presented in Section 4.6. The sampling mechanism is presented in Section 4.4, and Section 4.5 presents how an edge (or path) is constructed. The area where the PRM and RRT\* differ the most is how the two algorithms connect a waypoint to the tree data structure. Sections 4.8 and 4.9 provides implementation details on PRM and RRT\* specifics.

#### 4.4 Sampling milestones

A requirement of the problem statement is to guarantee<sup>1</sup> that a path will be found to the goal milestone<sup>2</sup>. There exists mathematical proofs for both the PRM and RRT which guarantees that a path will be found, however, a requirement of the proofs are that milestones are sampled randomly, uniformly distributed over the entire configuration space.

Milestones in this project are sampled uniformly distributed in the  $x$ ,  $y$ ,  $z$ ,  $time$  and  $heading$  dimensions, and it is assumed that the sampling range of these dimensions are provided. Each newly sampled milestone is tested for conflict, and if the conflict detection module indicates that conflict occurs between the sample and the environment, the sample is discarded. A new milestone is then sampled and tested for conflict, and this is repeated until a milestone is found that does not conflict with the environment.

#### 4.5 The manoeuvre library

The previous section discusses the sampling of milestones, and this section (Section 4.5) discusses how a vehicle moves from one milestone to a next, that is, the construction of the path the vehicle must travel to reach a milestone is presented in this section. In this project **manoeuvres** are used to construct the flight path of a vehicle. Manoeuvres<sup>3</sup> are used since they will ensure that the flight path is executable by a vehicle with kinodynamic and nonholonomic constraints.

**Definition 4.5.1** *A single manoeuvre consists of various manoeuvre primitives (or building blocks), and for this project very simple manoeuvre primitives are chosen: left turn, right turn, or forward.*

<sup>1</sup>This guarantee is conditional on the fact that a path does exist. Throughout this thesis this condition is always implied.

<sup>2</sup>The goal milestone is defined in Table 4.1

<sup>3</sup>The reader is referred to Subsection 3.4.2 for a discussion thereof.

This means that any manoeuvre is executed by executing several of these manoeuvre primitives in a specific order. It is necessary to define a structure (or library) in which a manoeuvre and the order of its manoeuvre primitives are contained:

**Definition 4.5.2** *A manoeuvre library contains all the possible manoeuvres which the vehicle may execute, that is, each entry in the library contains one manoeuvre and the order of its corresponding manoeuvre primitives.*

This project requires two manoeuvre libraries, one for the LPM and one for the Steer method. The reader is referred to Subsections 4.8.1 and 4.9.1 for discussions on both methods (or algorithms).

The manoeuvre library chosen for the LPM is listed below:

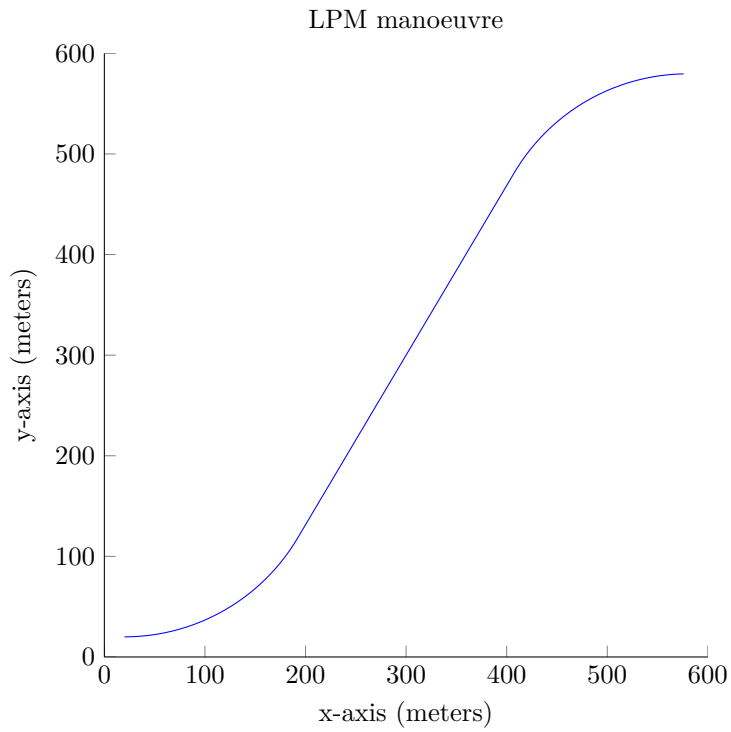
1. left turn; forward; left turn
2. left turn; forward; right turn
3. right turn; forward; left turn
4. right turn; forward; right turn

where the length of any manoeuvre primitive may be zero, and the turn radius is constant. An example is presented in Figure 4.1.

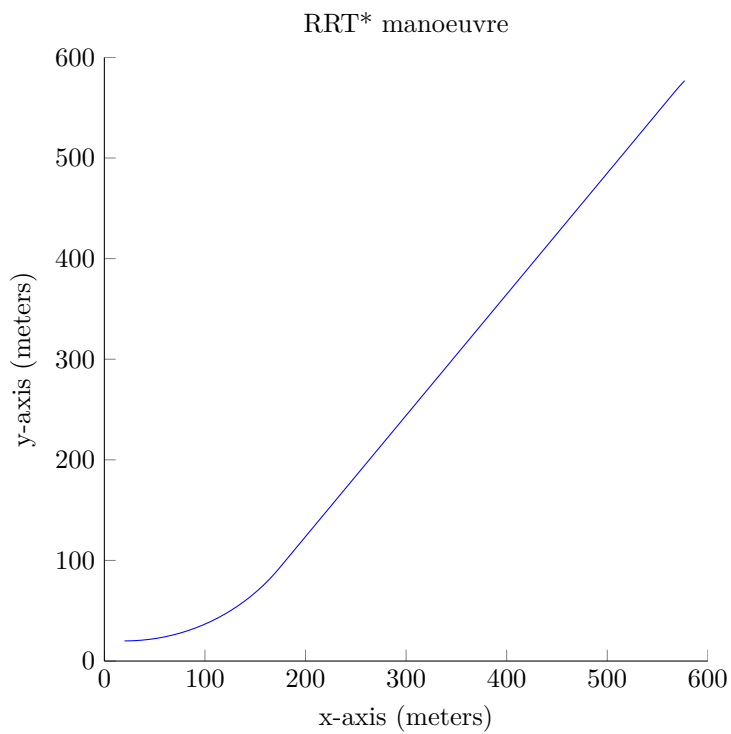
The Steer manoeuvre library consists of:

1. left turn; forward
2. right turn; forward

where the length of any manoeuvre primitive may be zero, and the turn radius is constant. The reason for this difference is discussed in Subsection 4.9.1. An example is presented in Figure 4.2.



**Figure 4.1** – A LPM manoeuvre consisting of a left turn, straight line, and a right turn.



**Figure 4.2** – A Steer manoeuvre consisting of a left turn and a straight line.

During the planning, several constraints need to be placed on the manoeuvres. This is to ensure the vehicle is not operated outside its envelope of operation. Another consideration is the limitations of the Simulink model used later in this thesis during simulation. Taking the above considerations into account, a turning radius of 50m, and a maximum allowable velocity of 1.2m/s is used during planning.

Using manoeuvres provides various benefits, however using manoeuvres also limit the manoeuvrability of the vehicle. For example, the manoeuvre library of the LPM only allows a set sequence (i.e. turn-line-turn), and executing three consecutive turns, or line-turn-line, is not allowed.

In the previous section, the sampling method used by the PRM and RRT\* algorithms is discussed, and in this section the manoeuvres used to construct an executable path between two milestones is presented. The next section (Section 4.6) discusses how all of the aforementioned information is stored during algorithm execution.

## 4.6 Data structures

A key part of both the PRM and RRT\* algorithms is growing a tree of reachable points across the entire search space. This tree is grown until it contains the goal milestone, or after a path to the goal is found, the algorithms can improve the path by continuing to grow their respective tree. During every algorithm iteration the milestones in the PRM and RRT\* trees are accessed multiple times. If the data structure does not efficiently handle the mass of information it contains, it will directly influence algorithm performance. The way the data structure is implemented is therefore very important.

### 4.6.1 The milestone data structure

The milestone data structure uses a tree type structure of which the vehicle starting point is used as the root milestone. The tree is then built by adding milestones as children to the root milestone. All milestones, except for the root, have parents. The significance of the normal parent-child relationship is that child milestones are reached via its parent, and building on this, a child milestone only knows how to reach itself from its parent. That is, at each milestone, only the path between it and its parent is stored. If the path from the starting point to a specific milestone  $\mathbf{m}$  is required, the path is built (in reverse) starting at  $\mathbf{m}$ . The path is built with calls to  $\mathbf{m}$ 's parent, the parent's parent, and so forth; each time their respective paths are added (in reverse) to that of  $\mathbf{m}$  until the starting milestone is reached.

| Variable             | Type           | Description                            |
|----------------------|----------------|--|
| x, y, z              | Double         | Position of milestone                  |
| time                 | Double         | Time reachable from initial milestone  |
| psi                  | Double         | Heading of milestone                   |
| theta                | Double         | Heave angle                            |
| cost                 | Double         | Cost from initial milestone            |
| *path                | Double pointer | Array of coordinates, forms trajectory |
| turn1_length         | Double         | Length of turn 1                       |
| turn2_length         | Double         | Length of turn 2                       |
| line_length          | Double         | Length of the line segment             |
| manoeuvre_primitive1 | Integer        | First manoeuvre primitive type         |
| manoeuvre_primitive2 | Integer        | Second manoeuvre primitive type        |
| manoeuvre_primitive3 | Integer        | Third manoeuvre primitive type         |

Table 4.1 – The **milestone** data structure.

The data structure in Table 4.1 is used to represent different points in the state space to where a vehicle can travel. The **x**, **y** and **z** members are used to store where the milestone is located in the  $x, y$ , and  $z$  dimensions. The **time** member is used to store the time by which this milestone can be reached from the starting point. The **psi** member stores the heading at which the vehicle must arrive at this milestone and the **theta** member stores the heave angle<sup>4</sup> required during the execution of the line manoeuvre primitive. The **cost** member stores the cost to reach a milestone from the starting milestone.

The manoeuvre primitives that form the path are either segments of geometrically describable circles or a straight line; therefore, the only information of a path that is necessary to store, is the manoeuvre primitive types and lengths. The **turn1\_length**, **turn2\_length**, **line\_length** members are used to store the lengths of the respective manoeuvre primitives, and **manoeuvre\_primitive1**, **manoeuvre\_primitive2**, **manoeuvre\_primitive3** members are used to store the manoeuvre primitive types (i.e. left turn, right turn, or straight line) of the path.

A path is described by points on a trajectory, where each point on the trajectory is separated by a set time interval<sup>5</sup>. These points are stored as an array of  $x, y$ , and  $z$  points in **\*path**. Since this can use a lot of memory, the **\*path** member is unused initially and is only allocated once it is required to contain a path, e.g. when the path to the goal must be returned.

Each milestone has a reference to its parent milestone as well as a **linked list** of nodes (defined in the next Subsection 4.6.2) containing references to children of this milestone. It is often required to traverse milestones in a sorted order, which is why the **node** and linked list data structures are introduced in the next Subsection 4.6.2.

#### 4.6.2 The node linked list data structure

The milestone data structure is used to keep track of all the points the vehicle can reach, and the inherent parent-child relationship thereof is beneficial for this purpose. The milestone data structure is therefore a good representation of the information it holds, however, it is much more difficult to sort a tree than a simple list. In addition, the parent-child relationships in the milestone data structure are not important when searching for a nearest neighbour; this is because it does not make sense to traverse points in a tree-like order for

<sup>4</sup>The angle between the  $x$ - $y$  plane and the  $z$  axis.

<sup>5</sup>This time interval can be changed.

this purpose. To address these issues, a linked list data structure is proposed to be used in addition to the tree data structure.

The node data structure contains references to its successor node, and therefore forms a linked list. Each node also contains a reference to a milestone, whereby it provides an easy way of traversing through milestones ordered by the respective node's successor relationships. This is especially helpful when a sorted list is used, or when a traversal of all known milestones is required. Table 4.2 shows the milestone node data structure, and Table 4.3 shows the milestone node list data structure.

| Variable  | Type           | Description                             |
|-----------|----------------|---|
| *data     | milestone      | Reference to a milestone                |
| *next     | milestone_node | Reference to a successor milestone node |
| temp_cost | Double         | Temporary cost                          |

**Table 4.2** – The **milestone node** data structure.

The **\*data** member points to a milestone and the **\*next** member points to the successor node in the list. The **temp\_cost** member is used when the list needs to be sorted by some value other than that of the **cost** member in a milestone.

| Variable | Type           | Description                           |
|----------|----------------|---------------------------------------|
| *head    | milestone_node | Reference to the first milestone node |
| *tail    | milestone_node | Reference to the last milestone node  |
| length   | Integer        | Number of nodes in the list           |

**Table 4.3** – The **milestone node list** data structure.

The **\*head** member points to the first node in the list, and this node contains a reference to the first milestone in the list, which is normally the starting point or initial milestone. The **\*tail** member points to the most recently added node. The **length** member stores the number of nodes in this list.

## 4.7 Conflict detection

A requirement of the problem statement for this project is that the path the vehicle executes is conflict free, and for this purpose a conflict detection algorithm was developed. The conflict detection algorithm used in this project is deterministic, and allows for dynamic environments. Even though the algorithm is currently implemented to work with only spheres and cubes, it can be extended to work with other shapes.

The conflict detection algorithm works by examining the points on a path between two milestones, and for each point the algorithm determines whether the point is outside of all the obstacles present in an environment. If the point is outside of all the obstacles, the point is deemed conflict free.

For static objects this is simple, but dynamic objects are at different locations at different time intervals. This project assumes that the conflict detection algorithm knows where the dynamic obstacles are at any point in time, that is, in the past, present and future. In order to detect path conflict with dynamic obstacles all points on the path need to have a x,y, and



z position, as well as a time associated with it. Note that the `.` operator is used to access a data member from a data structure, e.g.: `m1.time` accesses the time associated with `m1`.

**Input:** milestones `m1` and `m2`, and the **environment**

**Output:** Whether the path was conflict free or not.

`time_elapsed = m1.time;`

`static_shapes = get static shapes from environment;`

`dynamic_shapes = get dynamic shapes from environment;`

**for** *Points on the path between m<sub>1</sub> and m<sub>2</sub>* **do**

    determine the next *x*, *y*, and *z* point on the path, `pxyz`;

    test whether `pxyz` is inside one of the `static_shapes`;

**if** *conflict detected* **then**

        | **return** *true*;

**end**

`time_elapsed += one time step;`

    test whether `pxyz` is inside one of the `dynamic_shapes` at time `time_elapsed`;

**if** *conflict detected* **then**

        | **return** *true*;

**end**

**end**

**return** *false*;

**Algorithm 2:** The conflict detection algorithm.

Algorithm 2 shows that in order to test whether a path is conflict free every point on the path has to be tested. This is a major performance bottleneck in both the PRM and RRT\* motion planning algorithms.

To address this problem, the time steps used in this algorithm can be adjusted. If the time steps are too fine it will result in a very (CPU-time) expensive conflict detection process, but for course time steps a path might be detected as conflict free even though it is not. It should be noted that for a practical system a probabilistic conflict detection algorithm (as mentioned in Section 3.4.3) should be used.

## Conclusion

The basic building blocks used by both the PRM and RRT\* motion planning algorithms are described previously in this chapter. It is now possible to present both algorithms, the PRM in Section 4.8 and the RRT\* in 4.9.

## 4.8 The Probabilistic Roadmap Method - PRM

A major part of this project is about finding a path between a start and goal milestone. In the problem statement it is required to find not only a path, but this path has to be executable, and conflict free, and it is necessary to guarantee that a path will be found. This problem is referred to as the **motion planning** problem. Chapter 3 concludes that sampling based algorithms will be used to solve this problem, and that there are two main algorithms still actively researched. The PRM motion planning algorithm is presented in this section, and is loosely based on the PRM implemented by C.E. van Daalen [4].

The PRM algorithm uses the LPM algorithm (Subsection 4.8.1) to determine which manoeuvre to use to connect two milestones. It uses the Extend algorithm (Subsection 4.8.2) to determine the parent milestone of a newly sampled milestone, and the Path planner algorithm (Subsection 4.8.3) to keep track of everything.

### 4.8.1 The Local Planning Method - LPM

To get the manoeuvre that forms the path between two milestones, the PRM uses a method called the Local Planning Method (LPM). This method takes as parameters two milestones, where a position, time, and heading is associated with each. The LPM then executes all manoeuvres in its library and stores the path that minimises<sup>6</sup> the cost function in the end point milestone,  $\mathbf{m}_2$ . Note that the path the LPM returns is not necessarily conflict free as it does not consider any information about the environment.

**Input:** milestones  $\mathbf{m}_1$ , and  $\mathbf{m}_2$

**Output:** A path from  $\mathbf{m}_1$  to  $\mathbf{m}_2$ , stored in  $\mathbf{m}_2$

$\mathbf{m}_{2rfr}$  = copy of  $\mathbf{m}_2$ ;

$\mathbf{m}_{2rfl}$  = copy of  $\mathbf{m}_2$ ;

$\mathbf{m}_{2lfl}$  = copy of  $\mathbf{m}_2$ ;

$\mathbf{m}_{2lfr}$  = copy of  $\mathbf{m}_2$ ;

use right, forward, right manoeuvre primitives to form a path between  $\mathbf{m}_1$  and  $\mathbf{m}_{2rfr}$ ;

use right, forward, left manoeuvre primitives to form a path between  $\mathbf{m}_1$  and  $\mathbf{m}_{2rfl}$ ;

use left, forward, left manoeuvre primitives to form a path between  $\mathbf{m}_1$  and  $\mathbf{m}_{2lfl}$ ;

use left, forward, right manoeuvre primitives to form a path between  $\mathbf{m}_1$  and  $\mathbf{m}_{2lfr}$ ;

$\mathbf{m}_2$  = milestone with smallest cost ( $\mathbf{m}_{2rfr}$ ,  $\mathbf{m}_{2rfl}$ ,  $\mathbf{m}_{2lfl}$ ,  $\mathbf{m}_{2lfr}$ );

**Algorithm 3:** The Local planning algorithm.

Algorithm 3 only gives an overview of the tasks the LPM has to do. Most of the tasks are straight forward, and determining the characteristics<sup>7</sup> of the manoeuvre primitives are not of importance to this discussion. The left and right manoeuvre primitives are formed using trigonometric functions and thus form a segment of a geometrically describable circle. Subsection 4.5 presents a discussion on the choice of manoeuvre primitives.

The algorithm starts off by making copies of  $\mathbf{m}_2$ , and then uses a different entry in the manoeuvre library to connect to each copy, thereby forming multiple paths between  $\mathbf{m}_1$  and  $\mathbf{m}_2$ . The cost of executing each individual path is calculated and stored with its respective milestone copy. The path with the smallest cost is then deemed the most cost effective way to reach  $\mathbf{m}_2$  from  $\mathbf{m}_1$  (within the allowable manoeuvre library), and as such is stored in  $\mathbf{m}_2$ .

### Conclusion

The work in this subsection enables us to get the manoeuvre (or path) between two milestones. The next section (Section 4.8.2) continues the discussion of the PRM algorithm, with specific focus on which milestone a newly sampled milestone should be connected to, i.e. finding the best parent for a child milestone.

### 4.8.2 The Extend method

The PRM Extend method takes a newly sampled milestone,  $\mathbf{m}_{rand}$ , and tries to add it to the PRM-tree  $\mathbf{tree}_{prm}$ . The classic Extend method loops through the entire  $\mathbf{tree}_{prm}$ , running the LPM for every milestone in the  $\mathbf{tree}_{prm}$ . The classic Extend method does this to minimise the cost to get to  $\mathbf{m}_{rand}$  from the initial milestone. If the LPM can connect to the  $\mathbf{m}_{rand}$  from any milestone in the  $\mathbf{tree}_{prm}$  and conflict isn't detected along the manoeuvre, then the milestone is added to the  $\mathbf{tree}_{prm}$ .

The problem with the classic approach is that two (CPU-time) expensive methods<sup>8</sup> are called for each milestone in the  $\mathbf{tree}_{prm}$ . As the size of the  $\mathbf{tree}_{prm}$  grows the execution of

<sup>6</sup>Within the allowable manoeuvre library.

<sup>7</sup>I.e. the length of the right or left manoeuvre primitive.

<sup>8</sup>I.e. the LPM and conflict detection methods

the Extend method becomes very (CPU-time) expensive. This behaviour is unwanted as it means the time between adding new milestones to the tree is constantly increasing.

A proposed solution to this problem is to first add all the milestones in the  $\mathbf{tree}_{prm}$  into a list  $\mathbf{list}_{prm}$ , and then sort the list according to the distance a milestone therein is from the  $\mathbf{m}_{rand}$ .

**Input:** Tree of reached  $\mathbf{list}_{prm}$ , a pointer to the  $\mathbf{latest\_milestone}$ , the  $\mathbf{m}_{rand}$  to add to  $\mathbf{list}_{prm}$ , the  $\mathbf{goal\_milestone}$ , and the  $\mathbf{environment}$ .

**Output:** The algorithm adds the random milestone to its tree of milestones.

sort the  $\mathbf{list}_{prm}$  list by distance to the  $\mathbf{m}_{rand}$ ;

```

for the first 50 milestones in  $\mathbf{list}_{prm}$  do
    let  $\mathbf{m}$  be the next milestone in  $\mathbf{list}_{prm}$ ;
    connect the  $\mathbf{m}_{rand}$  to  $\mathbf{m}$  using the LPM();
    test the path between  $\mathbf{m}_{rand}$  and  $\mathbf{m}$  for conflict;
    if  $\mathbf{random\_milestone.path}$  is conflict free then
        add  $\mathbf{random\_milestone}$  to  $\mathbf{milestones}$ ;
         $\mathbf{random\_milestone.parent} = \mathbf{m}$ ;
         $\mathbf{latest\_milestone} = \mathbf{m}_{rand}$ ;
        return;
    end
end

```

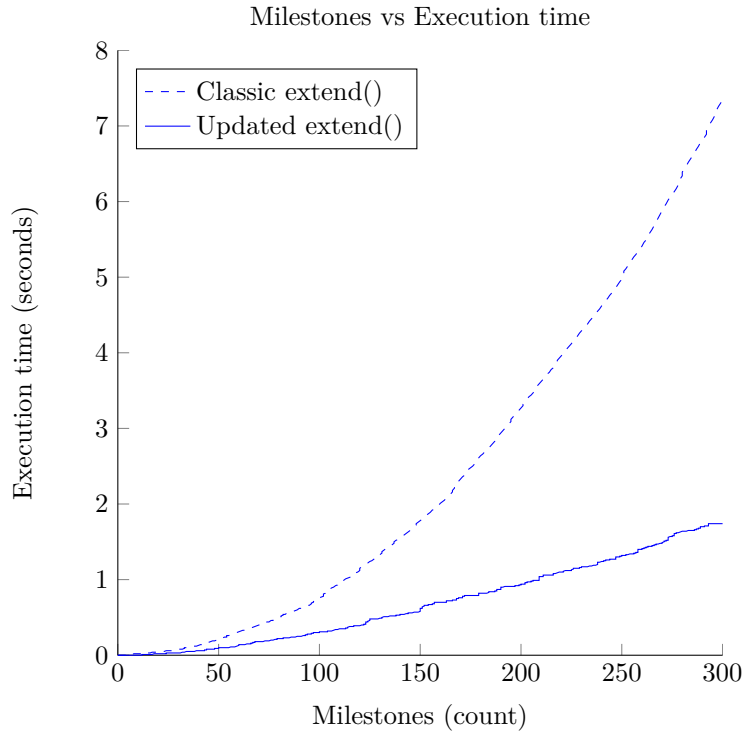
**end**

**Algorithm 4:** The PRM Extend algorithm.

In Algorithm 4 it is seen that by sorting  $\mathbf{list}_{prm}$  it is unnecessary to iterate through the entire  $\mathbf{tree}_{prm}$ , and the algorithm can stop as soon as a path is found. While this does no longer guarantee that the  $\mathbf{m}_{rand}$  is connected to the tree via the most cost efficient path, it reduces the execution time dramatically. Calculating the distance between the  $\mathbf{m}_{rand}$  and every other milestone in  $\mathbf{list}_{prm}$ , and then sorting the list is a relatively (CPU-time) inexpensive operation when compared to the execution of the LPM and the conflict detection methods for the same amount of milestones.

Another difference to the classic PRM Extend algorithm is limiting the number of milestones in the  $\mathbf{list}_{prm}$  from which the algorithm tries to connect to the  $\mathbf{m}_{rand}$ . The RRT\* discussed in Section 4.9 only tries to connect to the  $\mathbf{m}_{rand}$  from its nearest neighbour, and if that fails the  $\mathbf{m}_{rand}$  is discarded.

Sorting and limiting the number of milestones results in much less time between adding new milestones to the  $\mathbf{list}_{prm}$  list. In Figure 4.3 the execution times of the classic and updated Extend methods are compared. It should be noted that the path cost of the classic algorithm (10304 units) was better than that of the updated algorithm (12045 units); however, the classic algorithm took much longer for the same amount of milestones. For the same amount of time as the classic algorithm took to add all 300 milestones to its tree, the updated Extend algorithm managed to add 476 milestones in its tree, and thereby found a much better path (8617 units) to the goal.



**Figure 4.3** – The execution time for adding 300 milestones to a tree is shown here for both the classic and updated PRM Extend algorithms.

### 4.8.3 The Path Planner

The building blocks of the PRM algorithm are presented in the previous subsections, however, one more algorithm is necessary to bring everything together. The PRM path planner algorithm uses the LPM, Extend, milestone sampling, and conflict detection algorithm.

The **path planner** algorithm starts by trying to connect the newest milestone  $\mathbf{m}_{newest}$  in its tree  $\mathbf{tree}_{prm}$  (initially  $\mathbf{m}_{newest}$  is the initial milestone), to the goal milestone. If no conflict is detected, a path to the goal is found, and the path planner can stop. If conflict exists or the LPM cannot connect  $\mathbf{m}_{newest}$  to the goal milestone, a new random milestone  $\mathbf{m}_{rand}$  is sampled, and the PRM Extend function is used to try and add  $\mathbf{m}_{rand}$  to the  $\mathbf{tree}_{prm}$ . If  $\mathbf{m}_{rand}$  is successfully added to the tree,  $\mathbf{m}_{newest}$  now points to  $\mathbf{m}_{rand}$ , and the process

starts again by trying to connect  $\mathbf{m}_{newest}$  to the goal milestone.

**Input:** The **initial** milestone, the **goal** milestone, and the **environment**.

**Output:** The algorithm finds a path between the **initial** and **goal** milestone.

initialise  $\text{list}_{prm}$  tree;

**latest\_milestone** = **initial**;

**while** *algorithm should keep searching* **do**

    use the LPM to try and connect the **latest\_milestone** to the **goal**;

    test for conflict between the **latest\_milestone** and the **goal**;

**if** *the path between the latest\_milestone and the goal is conflict free* **then**

**if** *the cost to the goal is improved* **then**

            save the new path to the goal;

**end**

**if** *algorithm stop condition is satisfied* **then**

**return** path to the goal;

**end**

**else**

        generate new  $\mathbf{m}_{rand}$ ;

        use *extend\_prm()* to insert  $\mathbf{m}_{rand}$  into  $\text{list}_{prm}$ ;

**latest\_milestone** = returned by *extend\_prm()*;

**end**

**end**

**Algorithm 5:** The PRM path planner algorithm.

Algorithm 5 has several modes of operation: by altering the stop condition, it is possible to either limit the number of algorithm iterations, the number of milestones in the  $\text{tree}_{prm}$ , the time the algorithm has to solve the problem, or use the algorithm to search only while no path has been found to the goal.

The difference between limiting the iterations and milestones is subtle but important, the next section (Section 4.9) will elaborate further. Chapter 5 uses the proof of probabilistic completeness to calculate a limit on the number of milestones the PRM needs to guarantee a probability of finding a path.

## 4.9 The Rapidly exploring Random Tree\* - RRT\*

The PRM algorithm was presented in the previous section. It was presented as one of two possibilities for solving the motion planning part of this project's problem statement. In this section the RRT\* is presented as the second motion planning algorithm.

The classic RRT algorithm had a recent overhaul by S. Karaman and E. Frazzoli [17]. The improvements presented by the authors are related to the cost of the path the classic RRT finds. First, the authors show that the probability of the RRT finding the (almost) optimal path to a goal is zero. The authors then propose several changes to the classic RRT through which they achieve finding almost optimal paths. This new algorithm is referred to as the RRT\* motion planning algorithm. The authors also present mathematical proof that their algorithm converges to the (almost) optimal path, as the number of sampled milestones increases.

S. Karaman, et al. implemented the algorithm without taking kinodynamic and nonholonomic motion constraints into account. For this project these constraints cannot be relaxed, and they need to be taken into account during planning. Subsection 3.4.1 presents a discussion on kinodynamic and nonholonomic motion planning. The manoeuvres presented in Subsection 3.4.2 are used in this project to address this problem.

The Steer method (Subsection 4.9.1) uses manoeuvres to connect subsequent milestones, which allows the vehicle to travel from one milestone to the next while staying within its envelope of operation<sup>9</sup>.

For this project, the environment may also be dynamic, and therefore the motion planner must associate time with each vehicle position. The motion planner thus plans in three position ( $x, y$ , and  $z$ ) dimensions, the heading ( $psi$  or  $\psi$ ) dimension, as well as the time ( $t$ ) dimension.

#### 4.9.1 The Steer method

To connect milestones, the RRT\* uses a method called Steer. This method takes the same parameters as the LPM, however, the heading of the second milestone is ignored. This is because the RRT\* only “steers” toward the second milestone, i.e. it does not necessarily generate a path that reaches the second milestone. This is because the Steer method uses a constant to limit the length of any manoeuvre so that the sum of the lengths of the manoeuvre primitives are smaller than or equal to  $\eta$ .

The problem with this is that certain milestones (e.g. the goal milestone) cannot be connected via the Steer method because the end point headings cannot be ignored. Subsection 4.9.3, as well as the theoretical analysis of the next chapter presents more on this problem.

**Input:**  $\mathbf{m}_1$ , and  $\mathbf{m}_2$ .

**Output:** Returns a path from  $\mathbf{m}_1$  to  $\mathbf{m}_2$  and stores it in  $\mathbf{m}_2$ .

**right\_distance** = the distance between a point to the right of  $\mathbf{m}_1$  and  $\mathbf{m}_2$ ;

**left\_distance** = the distance between a point to the left of  $\mathbf{m}_1$  and  $\mathbf{m}_2$ ;

**if**  $right\_distance \leq left\_distance$  **then**

    determine the mid point of the right turning circle;

    determine the point at which the vehicle should leave the turning circle,  $(\mathbf{bx}, \mathbf{by})$ ;

    determine the arc length  $\mathbf{l1}$  from  $\mathbf{m}_1$  to  $(\mathbf{bx}, \mathbf{by})$ ;

    determine the length  $\mathbf{l2}$  from  $(\mathbf{bx}, \mathbf{by})$  to  $\mathbf{m}_2$ ;

    limit the combined length of  $\mathbf{l1}$  and  $\mathbf{l2}$  to be smaller than  $\eta$ ;

$\mathbf{m}_2.turn1\_length = \mathbf{l1}$ ;

$\mathbf{m}_2.line\_length = \mathbf{l2}$ ;

    move  $\mathbf{m}_2$  to coincide with the end of the manoeuvre.

**else**

    do the same, except by using a left turning circle;

**end**

**Algorithm 6:** The RRT\* steer algorithm.

Algorithm 6 determines when to leave the turning circle through use of trigonometric functions. That is, the manoeuvre primitive (or curve) connecting  $\mathbf{m}_1$  and  $(\mathbf{bx}, \mathbf{by})$  is characterised by trigonometric functions and form a segment of a geometrically describable circle. The manoeuvre primitive is flat in the  $x, y$  plane, which means that the vehicle does not heave while traversing the manoeuvre primitive.

Determining the length of the second manoeuvre primitive (or straight line segment) from  $(\mathbf{bx}, \mathbf{by})$  to  $\mathbf{m}_2$  is straightforward. While the first manoeuvre primitive is flat in the  $x, y$  plane, the second manoeuvre primitive connects in the  $x, y$  and  $z$  dimension. Figure 4.2 shows a RRT\* manoeuvre.

<sup>9</sup>The envelope of operation refers to the inherent constraints a vehicle has while moving. In particular, the kinodynamic and nonholonomic motion constraints are referred to in this context.

The RRT\* only steers towards a milestone, not necessarily reaching the milestone; that is, the manoeuvre is limited. The manoeuvre is limited by restricting the length of the combined manoeuvre primitives (or circle and line segment) by  $\eta$ . The reasons for this are discussed throughout this section as well as in Chapter 5.

Lastly,  $\mathbf{m}_2$  is moved to where the limited manoeuvre ends. This is necessary since the manoeuvre is shortened and will often stop before  $\mathbf{m}_2$  is reached.

### 4.9.2 The Extend method

The Extend method bridges the gap between being able to connect to milestones (Steer method) and the path planner. The Extend method's responsibility is to find if a suitable parent milestone exists for a newly sampled milestone, and if one exists determine which parent is the best. The method also determines if this newly sampled milestone can become a better parent for some other milestone in the  $\mathbf{tree}_{rrt^*}$ .

#### Nearest neighbour

First, a search is done for the nearest milestone to  $\mathbf{m}_{rand}$ ; let  $\mathbf{m}_{nearest}$  be the nearest milestone to  $\mathbf{m}_{rand}$  in the  $\mathbf{tree}_{rrt^*}$ . The Steer method is then used to steer towards  $\mathbf{m}_{rand}$  and the path it returns is tested for conflict. If conflict is detected,  $\mathbf{m}_{rand}$  is discarded and a new  $\mathbf{m}_{rand}$  is sampled. If no conflict is detected,  $\mathbf{m}_{rand}$  is moved to coincide with where the path returned by the Steer function ended.

As mentioned in Section 4.4, this project samples milestones in five dimensions:  $x$ ,  $y$ ,  $z$ ,  $time$ , and  $heading$ ; however, the nearest neighbour search in this project only looks at the  $x$ ,  $y$ ,  $z$  dimensions.

#### Parent optimisation

Parent optimisation refers to searching for the best parent for a milestone  $\mathbf{m}_{child}$ , such that the cost of reaching  $\mathbf{m}_{child}$  is as low as possible. The result is lower costs to reach milestones, and will result in a lower cost path to the goal.

To determine this, an area defined by a ball of radius  $r$  around  $\mathbf{m}_{rand}$  is searched for milestones (within the  $\mathbf{tree}_{rrt^*}$ ) to which  $\mathbf{m}_{rand}$  can be connected. All of these milestones are connected to  $\mathbf{m}_{rand}$  via the Steer method, and the milestone from which  $\mathbf{m}_{rand}$  can be reached with the smallest cumulative cost is then set as the parent of  $\mathbf{m}_{rand}$ . The radius  $r$  is determined according to Equation 4.9.4.

#### Determining $r$

$$unit\_sphere = \frac{4\pi}{3} \quad (4.9.1)$$

$$\gamma = 2^d \times \left(1 + \frac{1}{d}\right) \times (volume\ of\ search\ space) \quad (4.9.2)$$

$$growth\_decay = \frac{\log(n)}{n} \quad (4.9.3)$$

$$r = \left(\frac{\gamma \times growth\_decay}{unit\_sphere}\right)^{\frac{1}{d}} \quad (4.9.4)$$

The equation above is taken from S. Karaman, et al. [17]. Equation 4.9.1 is the volume of a unit sphere in three dimensions,  $d$  in Equations 4.9.2 and 4.9.4 is the number of dimensions,  $n$  in Equation 4.9.3 is the number of milestones in the  $\mathbf{tree}_{rrt*}$ , and Equation 4.9.4 is used to determine  $r$ .

It is seen from Equation 4.9.3 that the value of  $r$  will decrease as  $n$  increases; that is, as more milestones are included in the  $\mathbf{tree}_{rrt*}$  the value of  $r$  decreases which enables the the **Parent optimisation** and **New milestone optimisation** to occur without becoming very (CPU-time) expensive. For an in depth discussion the reader is referred to the paper of origin.

### **New milestone optimisation**

After  $\mathbf{m}_{rand}$  is successfully added to the  $\mathbf{tree}_{rrt*}$ , the milestones that fall within the  $r$  radius ball from  $\mathbf{m}_{rand}$  are checked to see if  $\mathbf{m}_{rand}$  can lower any cumulative path-cost to connect to them. It should be noted that the LPM (not Steer) method is used for this case as the



heading of both milestones should be taken into account for any path shortening.

**Input:** the  $\text{list}_{\text{rrt}^*}$  list,  $\text{latest\_milestone}$ ,  $\mathbf{m}_{\text{rand}}$ ,  $\text{goal}$ ,  $\text{environment}$

**Output:** Attempts to add the  $\mathbf{m}_{\text{rand}}$  to the  $\text{list}_{\text{rrt}^*}$  list

$\text{nearest\_milestone}$  = nearest milestone to  $\mathbf{m}_{\text{rand}}$ ;

use the steer method to connect  $\text{nearest\_milestone}$  to  $\mathbf{m}_{\text{rand}}$ ;

test for conflict on the path between  $\text{nearest\_milestone}$  to  $\mathbf{m}_{\text{rand}}$ ;

$\text{best\_cost}$  =  $\text{random\_milestone.cost}$ ;

**if** the path is conflict free **then**

**for** each  $\mathbf{m}$  in  $\text{list}_{\text{rrt}^*}$  **do**

$\text{distance}$  = distance between  $\mathbf{m}$  and  $\mathbf{m}_{\text{rand}}$ ;

**if**  $\text{distance} < \eta$  **then**

$\text{temp\_milestone}$  = copy of  $\mathbf{m}_{\text{rand}}$ ;

      use steer to connect  $\mathbf{m}$  to  $\text{temp\_milestone}$ ;

      test for conflict on path between  $\mathbf{m}$  and  $\text{temp\_milestone}$ ;

**if**  $\text{random\_milestone.cost} > \text{temp\_milestone}$  **then**

$\text{random\_milestone.cost}$  =  $\text{temp\_milestone.cost}$ ;

$\text{random\_milestone.path}$  =  $\text{temp\_milestone.path}$ ;

**end**

**end**

**end**

insert  $\mathbf{m}_{\text{rand}}$  into  $\text{list}_{\text{rrt}^*}$ ;

$\text{latest\_milestone}$  =  $\mathbf{m}_{\text{rand}}$ ;

**for** each  $\mathbf{m}$  in  $\text{list}_{\text{rrt}^*}$  **do**

$\text{distance}$  = distance between  $\mathbf{m}_{\text{rand}}$  and  $\mathbf{m}$ ;

**if**  $\text{distance} < \eta$  **then**

$\text{temp\_milestone}$  = copy of  $\mathbf{m}$ ;

    use the LPM to connect  $\mathbf{m}_{\text{rand}}$  to  $\text{temp\_milestone}$ ;

    test for conflict on path between  $\mathbf{m}_{\text{rand}}$  and  $\text{temp\_milestone}$ ;

**if**  $\mathbf{m.cost} > \text{temp\_milestone.cost}$  **then**

$\mathbf{m.cost}$  =  $\text{temp\_milestone.cost}$ ;

$\mathbf{m.path}$  =  $\text{temp\_milestone.path}$ ;

      update the children of  $\mathbf{m}$  with the lower cost;

**end**

**end**

**end**

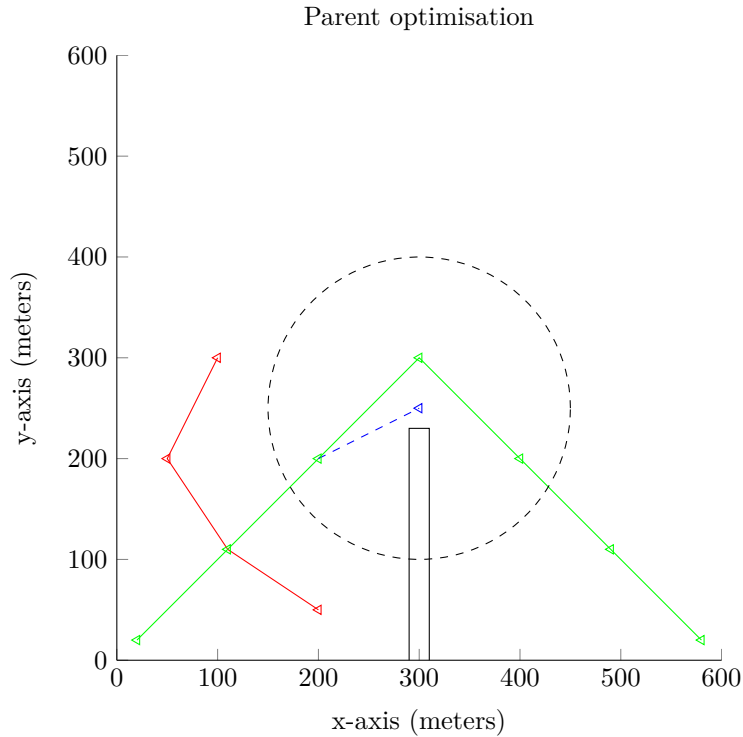
**end**

**Algorithm 7:** The RRT\* extend algorithm.

The RRT\* Extend algorithm presented in Algorithm 7 works very differently to its PRM counterpart. The Extend algorithm starts by testing if  $\mathbf{m}_{\text{rand}}$  can be connected to from its nearest neighbour; if this test fails, then  $\mathbf{m}_{\text{rand}}$  is discarded. This test vets random samples that are difficult to add to the tree, and potentially saves a lot of CPU-time as only one steer and conflict test is needed.

If  $\mathbf{m}_{\text{rand}}$  can be reached from its nearest neighbour, then more milestones in the  $\text{list}_{\text{rrt}^*}$  are checked. Since the Steer algorithm limits the distance that a single path between two milestones can reach, it does not make sense to iterate through the entire  $\text{list}_{\text{rrt}^*}$ . In algorithm 7 it is seen that (before the CPU-time expensive operations are executed) the distance between  $\mathbf{m}$  and  $\mathbf{m}_{\text{rand}}$  is checked, and milestones further than  $\eta$  away are discarded immediately. That is, a ball of radius  $\eta$  is formed around  $\mathbf{m}_{\text{rand}}$ , and milestones outside of this ball are not considered in this process. After the best path to  $\mathbf{m}_{\text{rand}}$  is found it is inserted into  $\text{list}_{\text{rrt}^*}$ .

Next, it is necessary to check whether any known path can be improved by going through  $\mathbf{m}_{rand}$ . In Figure 4.4 a recently added  $\mathbf{m}_{rand}$  is shown in blue. It is seen that this milestone can decrease the cost of the path to the goal, shown in green. It is also seen that only milestones within the black dotted circle need to be considered, as only those milestones are close enough, i.e. they are within a distance  $\eta$  from  $\mathbf{m}_{rand}$ .



**Figure 4.4** – This figure shows a recently added milestone in blue, a ball (shown as a circle in 2 dimensions) of radius  $\eta$  in black, and a path between the start point and goal in green. The milestones shown in red are in the  $\text{list}_{rrt^*}$  list but do not form part of the path to the goal.

From Figure 4.4 it is seen that the size of the ball directly influences the number of milestones that has to be considered when searching to improve paths, as well as when  $\mathbf{m}_{rand}$  is added to the  $\text{list}_{rrt^*}$  list. Hence the choice of  $\eta$  is very important and is discussed further in Subsection 5.3.1.4. Figure 4.5 shows the improved path.

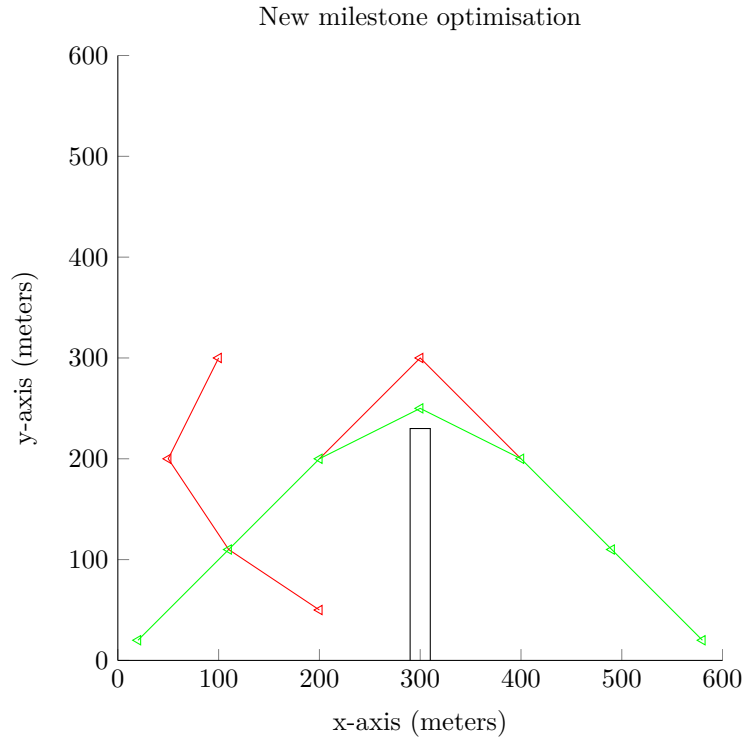


Figure 4.5 – This figure shows the improved path of Figure 4.4.

### 4.9.3 Path planner

The RRT\* path planner presented in algorithm 8 works very similar to the PRM path planner method. It also has a pointer to  $\mathbf{m}_{newest}$  which it tries to connect to the goal milestone, however the goal has to be reached at a specific heading, which means that the Steer method cannot be used for this. Fortunately, another method already exists that takes the initial and end heading into account, the LPM. Thus, instead of using the Steer method, the LPM is used to try and connect  $\mathbf{m}_{newest}$  to the goal milestone.

This has another benefit, the RRT\* algorithm now no longer needs to explore until it happens to reach a goal region: A manoeuvre from the LPM is not limited by  $\eta$ . This means that as soon as one milestone can “see” another milestone it can be connected in a single manoeuvre,

resulting in a single RRT\* path planner iteration once the goal comes into view.

**Input:** the **initial** milestone, the **goal** milestone, and the **environment**

**Output:** A path between the **initial** and **goal** milestone is returned

initialise  $\text{list}_{\text{rrt}^*}$  tree;

$\text{latest\_milestone} = \text{initial}$ ;

**while** *algorithm should keep searching* **do**

    use the LPM to try and connect the  $\text{latest\_milestone}$  to the **goal**;

    test for conflict between the  $\text{latest\_milestone}$  and the **goal**;

**if** *the path between the latest\_milestone and the goal is conflict free* **then**

**if** *the cost to the goal is improved* **then**

            | save the new path to the goal;

**end**

**if** *algorithm should stop as soon as a path is found* **then**

            | **return** path to the goal;

**end**

**else**

**if** *number of iterations exceeds limit* **then**

            | **return** path to the goal;

**end**

**if** *number of milestones exceeds limit* **then**

            | **return** path to the goal;

**end**

**if** *execution time exceeds limit* **then**

            | **return** path to the goal;

**end**

        generate new  $\mathbf{m}_{\text{rand}}$ ;

        use *extend\_rrt* to insert  $\mathbf{m}_{\text{rand}}$  into  $\text{list}_{\text{rrt}^*}$ ;

$\text{latest\_milestone} =$  returned by *extend\_rrt*;

**end**

**end**

**Algorithm 8:** The RRT\* path planner algorithm.

## 4.10 Conclusion

In this chapter two sampling based algorithms are implemented, the PRM and RRT\*. Both algorithms use a few common building blocks: the data structure, conflict detection algorithm and the sampling algorithm. However, the similarities stop there, and at a very low level they differ significantly: The PRM uses the LPM to connect milestones and the RRT\* uses the Steer algorithm. These two algorithms differ significantly, and both offer advantages as well as disadvantages to their respective motion planning algorithm.

Next, the respective Extend algorithms are found, and they also differ significantly, even though they provide the same function. The function of both Extend algorithms is to add a random milestone to its tree of known milestones. The respective path planner algorithms of the PRM and RRT\* are essentially the same, again providing the same function. Both path planner algorithms take exactly the same parameters, and provide the same output: a path between the initial and goal milestones.

Since both algorithms can provide solutions to the problem statement, it is necessary to explore two more aspects: how good are the solutions, and how long does it take to find a solution? As it is known that there are implementation differences, it is necessary to know what effect these differences have. For this reason, Chapter 5 is presented next, wherein both the PRM and RRT\* are analysed.

## Chapter 5

# Algorithm Analysis

### 5.1 Introduction

This project concerns autonomous navigation for UAVs, of which motion planning is an essential part. Different options for motion planning are discussed in Section 3.2, and it is concluded therein that sampling based motion planning algorithms are the best option for this project. The implementation of two sampling based algorithms, the PRM and RRT\* are presented in Chapter 4. At the top level, the two algorithms provide the same functionality, that is, they both provide a path between the initial and goal milestones<sup>1</sup>. In contrast, the working of the low level building blocks of the PRM and RRT\* differ significantly.

As part of this project's problem statement, a path must be found in real time, and therefore it is important to analyse the performance<sup>2</sup> of the PRM and RRT\*. By analysing the effect different environments have on the algorithms' performance, it is possible to identify the effect the aforementioned differences have: if the performance of the PRM and RRT\* differs for similar environments, and the only differences are those from the respective LPM, Steer, and Extend algorithms (Subsections 4.8.1, 4.9.1, 4.8.2, and 4.9.2), then the difference in performance is a direct result from the differences between those algorithms.

In this chapter the performance<sup>3</sup> of the PRM (Section 5.2) and RRT\* (Section 5.3) is analysed and compared in a specific environment. By analysing the effect an environment has on the performance of the PRM and RRT\* algorithms, it is possible to gain insight into the effect their differences have. Furthermore, if it is possible to determine how many iterations<sup>4</sup> or milestones each algorithm requires in a specific environment, the CPU time required to guarantee finding a solution can be determined. From this analysis, it is then possible to determine which algorithm to use, given a specific type of environment.

For the analysis, parts of the respective proofs for probabilistic completeness of the PRM and RRT\* are used to determine theoretical performance bounds. Lastly, a histogram analysis (Section 5.4) is performed to confirm whether the theoretical performance bounds of the PRM and RRT\* holds when compared to practical measurements.

---

<sup>1</sup>For a discussion on what a milestone is and how it is used the reader is referred to Subsection 4.6.1.

<sup>2</sup>The cost of a path and the time it takes to find a path.

<sup>3</sup>With respect to how many milestones or iterations are necessary to guarantee that a path between the initial and goal milestone in a given environment will be found with a required probability.

<sup>4</sup>For each algorithm iteration exactly one milestone is sampled, however, the milestone may be discarded and not inserted into the PRM or RRT tree.

## 5.2 PRM analysis

At the top level, the PRM and RRT\* algorithms provide the same functionality, however, the inner working of the algorithms differ significantly. The area where this difference matters most is performance, that is, how fast an algorithm can provide a path and what the quality of such a path is.

To analyse the PRM, a theorem presented by Hsu, Kindel, Latombe and Rock [16] is used to provide an upper bound on the PRM's performance in a specific environment. This theorem forms part of the proof for probabilistic completeness for the PRM. It is seen that this theorem has three variables ( $\alpha$ ,  $\beta$  and  $g$ ) which are environment dependent. If these three variables can be determined for a given environment, then it is possible to determine performance bounds for the PRM for a specific environment.

Determining  $\alpha$  and  $\beta$  analytically may be possible for very simple cases (as noted by Hsu, et al. [16]), however, calculating  $\alpha$  and  $\beta$  for more complex environments is still an open problem. This section presents non-analytical methods of estimating values for both  $\alpha$  and  $\beta$ .

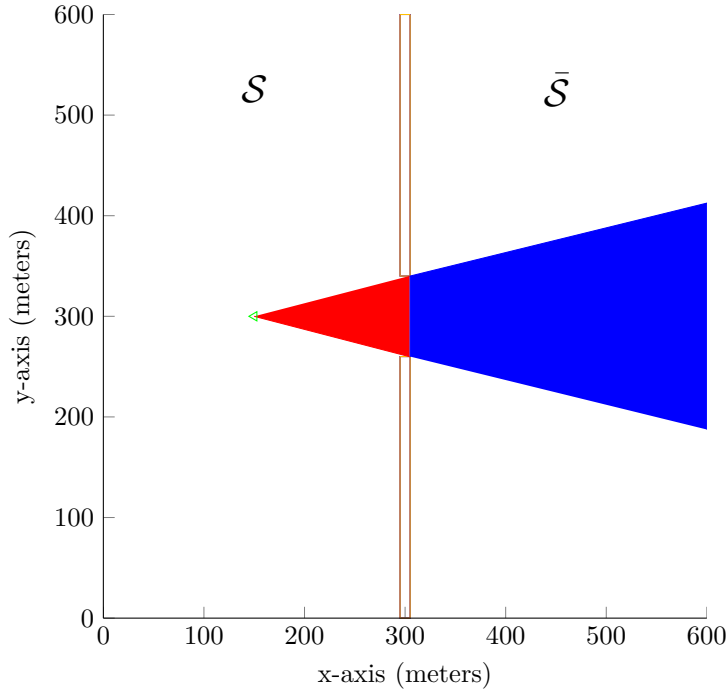
### 5.2.1 Key concepts necessary to determine the theoretical upper bound

In order to analyse the PRM, a theorem presented by Hsu, et al. is used to determine a theoretical performance bound for finding a path<sup>5</sup> in a specific environment. In this subsection, concepts critical to the theorem of Hsu, et al. are introduced before the theorem is presented in Subsection 5.2.2.

---

<sup>5</sup>A path is found with a guaranteed success percentage.

**Introducing the concept of the LPM reachability of a point**



**Figure 5.1** – LPM reachability for a point  $p$ .

In Figure 5.1 two walls are shown in the middle of the x-axis, one stretches from  $y = 0m$  to  $y = 260m$  and the other stretches from  $y = 340m$  to  $y = 600m$ . In the figure  $\mathcal{S}$  denotes a set of points to the left of the wall, and  $\bar{\mathcal{S}}$  (read not  $\mathcal{S}$ ) denotes the set of points to the right of the wall. The green triangle denotes a point  $p$  located in  $\mathcal{S}$ , and the set of points shown in blue is outside  $\mathcal{S}$  (denoted by  $\bar{\mathcal{S}}$ ) wherein  $p$  can reach any point using a single LPM<sup>6</sup> run. Formally, let:

$$\mathcal{R}_{lpm}(p) \tag{5.2.1}$$

denote the LPM reachable area from  $p$ , and let:

$$\mathcal{R}_{lpm}(p) \setminus \mathcal{S} \tag{5.2.2}$$

denote the LPM reachable area from  $p$  outside of  $\mathcal{S}$ .

**Introducing the concept of the reachability of a set**

The reachability of a set of points is the set of points reachable through multiple LPM runs. In Figure 5.1 the reachability of  $\mathcal{S}$  will include almost all the points included in the combined set  $\mathcal{S} \cup \bar{\mathcal{S}}$ .

Formally, let:

$$\mathcal{R}(\mathcal{S}) \setminus \mathcal{S} \tag{5.2.3}$$

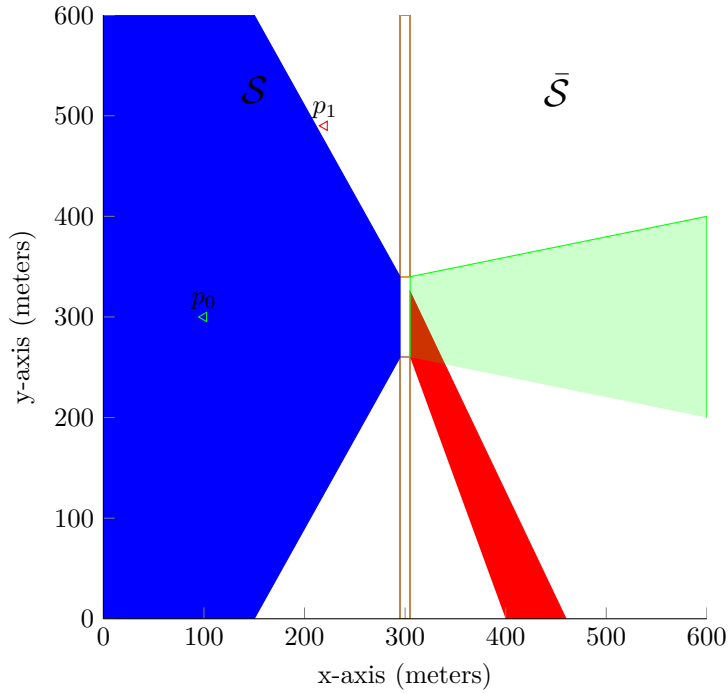
<sup>6</sup>Straight lines are used for illustrative purposes only, in the rest of the chapter the LPM will use the manoeuvre library as defined in Section 4.5.

denote the reachability of the set  $\mathcal{S}$  outside of  $\mathcal{S}$ .

The major difference between a set of points denoted by  $\mathcal{R}(\mathcal{S})$  and  $\mathcal{R}_{lpm}(p)$  is that the former includes points that are only reachable by multiple LPM runs, and the latter is restricted to points that are reachable by no more than a single LPM run. Another difference is that a point is considered in  $\mathcal{R}(\mathcal{S})$  if it is reachable from any point in  $\mathcal{S}$ , whereas  $\mathcal{R}_{lpm}(p)$  denotes a set of points that must be reachable from  $p$ .

### Introducing the concept of the $\beta$ -lookout of $\mathcal{S}$

Following the introduction of  $\mathcal{R}(\mathcal{S})$  and  $\mathcal{R}_{lpm}(p)$  it is now possible to introduce the  $\beta$ -lookout of  $\mathcal{S}$  in Figure 5.2.



**Figure 5.2** – In blue the  $\beta$ -lookout of  $\mathcal{S}$  is shown for a **small** (close to zero) value of  $\beta$ . The red area shows the LPM reachability set  $\mathcal{R}_{lpm}(p_1) \setminus \mathcal{S}$  and the green area shows the LPM reachability set  $\mathcal{R}_{lpm}(p_0) \setminus \mathcal{S}$ .

In Figure 5.2, the blue area is referred to as the  $\beta$ -lookout of a set of points  $\mathcal{S}$ ; generally speaking it is the set of points in  $\mathcal{S}$  that can reach a required percentage of points in  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$  through a single LPM run.

Formally, let  $p$  be a point in the set  $\mathcal{S}$  denoted by:

$$\{p \in \mathcal{S}\} \tag{5.2.4}$$

and let:

$$volume\ of\ (\mathcal{R}_{lpm}(p) \setminus \mathcal{S}) \geq volume\ of\ (\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}) \times required\ percentage \tag{5.2.5}$$

denote the required percentage of points.

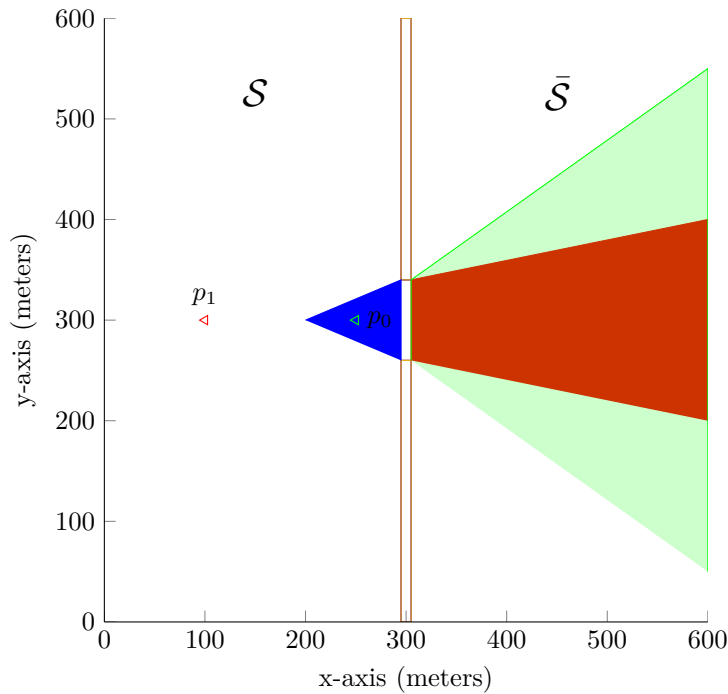


The red area of Figure 5.2 is the LPM reachability set  $\mathcal{R}_{lpm}(p_1) \setminus \mathcal{S}$  and it is seen that  $p_1$  lies just outside the blue area. This is because the volume of the points in  $\mathcal{R}(p_1) \setminus \mathcal{S}$  relative to the volume of points in  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$  is smaller than the *required percentage* of Equation 5.2.5. The green area shows the LPM reachability set  $\mathcal{R}(p_2) \setminus \mathcal{S}$  and it is seen that point  $p_0$  is inside the blue area. This is because the *required percentage* of Equation 5.2.5 is large enough for  $p_0$  to be inside the blue area.

By letting  $\mu(\mathcal{S})$  denote the volume of the set of points  $\mathcal{S}$ ,  $\beta$  denote the *required percentage*, and combining Equations 5.2.4 and 5.2.5, a mathematical definition of the blue area is formed:

$$\{p \in \mathcal{S} \mid \mu(\mathcal{R}_{lpm}(p) \setminus \mathcal{S}) \geq \beta \times \mu(\mathcal{R}(\mathcal{S}) \setminus \mathcal{S})\}. \quad (5.2.6)$$

Figure 5.2 shows the  $\beta$ -lookout of  $\mathcal{S}$  for a **small** (close to zero) value of  $\beta$ , and Figure 5.3 shows the  $\beta$ -lookout of  $\mathcal{S}$  for a **large** (close to one) value of  $\beta$ . In the latter figure the blue area is very small because for a point  $p$  to lie in the  $\beta$ -lookout of  $\mathcal{S}$  the LPM reachability  $\mathcal{R}_{lpm}(p)$  has to include most of  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$  since its volume has to be almost equal to the volume of  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$ .



**Figure 5.3** –  $\beta$ -lookout of  $\mathcal{S}$  for a **large** (close to one) value of  $\beta$ .

The red area of Figure 5.3 is the LPM reachability set  $\mathcal{R}_{lpm}(p_1) \setminus \mathcal{S}$  and it is seen that  $p_1$  lies just outside the blue area. This is because the volume of the points in  $\mathcal{R}(p_1) \setminus \mathcal{S}$  relative to the volume of points in  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$  is smaller than the *required percentage* of Equation 5.2.5. The green area shows the LPM reachability set  $\mathcal{R}(p_2) \setminus \mathcal{S}$  and it is seen that point  $p_0$  is inside the blue area. This is because the *required percentage* of Equation 5.2.5 is large enough for  $p_0$  to be inside the blue area.

**5.2.1.1 Definition of the  $\beta$ -lookout( $\mathcal{S}$ )**

Next, a formal definition of the  $\beta$ -lookout of a set  $\mathcal{S}$  is presented. Let  $\mathcal{S}$  denote the *state space* of the vehicle,  $\mathcal{T}$  denote the *time space* of the vehicle,  $\mathcal{F}$  denote the set of conflict free points in  $\mathcal{S} \times \mathcal{T}$ , and  $\mathcal{X}$  denote the set of points reachable from the initial vehicle state. Further, let  $\beta$  be a constant in  $(0, 1]$ , and  $\mu(A)$  denote the volume of a set  $A \subset \mathcal{F}$ .

The  $\beta$ -lookout of a set  $\mathcal{S} \subset \mathcal{F}$  is:

$$\beta\text{-lookout}(\mathcal{S}) = \{p \in \mathcal{S} \mid \mu(\mathcal{R}_{lpm}(p) \setminus \mathcal{S}) \geq \beta \times \mu(\mathcal{R}(\mathcal{S}) \setminus \mathcal{S})\} \tag{5.2.7}$$

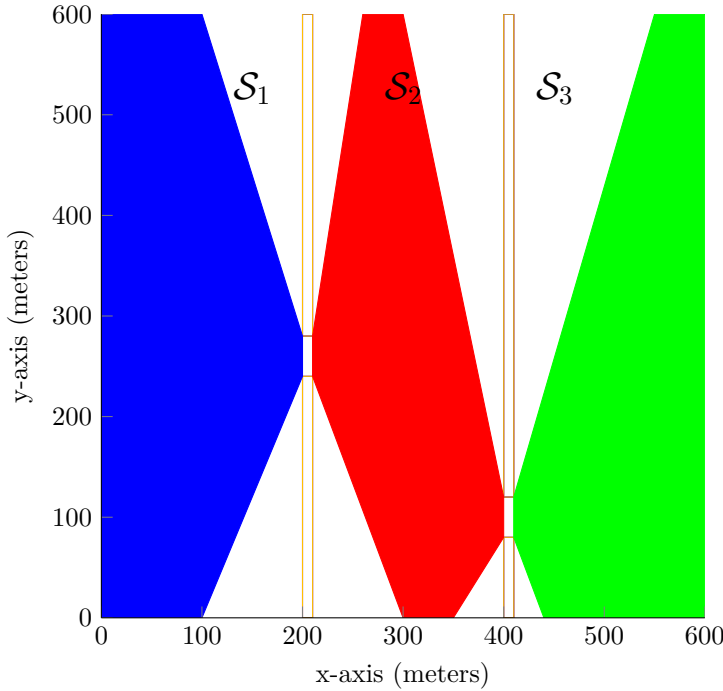
It should be noted that Hsu, et al. scaled all volumes so that  $\mu(\mathcal{X}) = 1$ .

**Introducing the concept of expansiveness**

Generally speaking, expansiveness refers to the percentage of points in  $\mathcal{S}$  that can reach points outside of  $\mathcal{S}$  through a single LPM run. Intuitively, if this percentage is high then points in  $\mathcal{S}$  can easily be connected to points outside of  $\mathcal{S}$ , and inversely so for a low percentage.

Formally, let this percentage be defined as an expansiveness ratio:

$$\text{expansiveness ratio} = \frac{\mu(\beta\text{-lookout}(\mathcal{S}))}{\mu(\mathcal{S})}. \tag{5.2.8}$$



**Figure 5.4** – In blue, the  $\beta$ -lookout( $\mathcal{S}_1$ ), in red the  $\beta$ -lookout( $\mathcal{S}_2$ ), and in green the  $\beta$ -lookout( $\mathcal{S}_3$ ).

The  $\beta$ -lookout( $\mathcal{S}_1$ ),  $\beta$ -lookout( $\mathcal{S}_2$ ), and  $\beta$ -lookout( $\mathcal{S}_3$ ) are shown in blue, red and green in Figure 5.4 for a small value (close to zero) of  $\beta$ .

Next, consider a point  $p$  anywhere in the blue area of Figure 5.4. The set of points  $\mathcal{R}(p)$  can be divided into three subsets:  $\mathcal{S}_1$ ,  $\mathcal{S}_2$ , and  $\mathcal{S}_3$ , where the subsets must be strongly connected. This means that, for any two points  $m, n$  in  $\mathcal{S}_1$ ,  $n$  must be in  $\mathcal{R}_{lpm}(m)$  and  $m$  must be in  $\mathcal{R}_{lpm}(n)$ . Other choices of subsets are elaborated on later in this chapter.

If there exists an expansiveness ratio  $\alpha$  and a value for  $\beta$  such that the following:

$$\mu(\beta\text{-lookout}(\mathcal{S}_1)) \geq \alpha \times \mu(\mathcal{S}_1) \quad (5.2.9)$$

$$\mu(\beta\text{-lookout}(\mathcal{S}_2)) \geq \alpha \times \mu(\mathcal{S}_2) \quad (5.2.10)$$

$$\mu(\beta\text{-lookout}(\mathcal{S}_3)) \geq \alpha \times \mu(\mathcal{S}_3) \quad (5.2.11)$$

is mathematically sound, then the set  $\mathcal{R}(p)$  is said to be  $(\alpha, \beta)$ -expansive. That is, the expansiveness of the set  $\mathcal{R}(p)$  is now characterised by  $\alpha$  and  $\beta$ . Lastly, this is extended to any set of points: any set of points  $\mathcal{A}$  can be characterised as  $(\alpha, \beta)$ -expansive, if for every  $p \in \mathcal{A}$ ,  $\mathcal{R}(p)$  is  $(\alpha, \beta)$ -expansive.

### 5.2.1.2 Definition of $\alpha$

Following the formal definition of the  $\beta$ -lookout( $\mathcal{S}$ ) and the introduction of  $(\alpha, \beta)$ -expansiveness, a formal definition of  $\alpha$  is presented. Let  $\alpha$  and  $\beta$  be two constants in the range  $(0, 1]$ . For any  $p \in \mathcal{F}$ , the set  $\mathcal{R}(p)$  is  $(\alpha, \beta)$ -expansive, if for every connected subset  $\mathcal{S} \subset \mathcal{R}(p)$ :

$$\mu(\beta\text{-lookout}(\mathcal{S})) \geq \alpha \times \mu(\mathcal{S}). \quad (5.2.12)$$

## Summary

In order to use the theorems by Hsu et al. [16], several key concepts are first introduced in this subsection (Subsection 5.2.1). The theorems are used to determine a performance bound on the number of milestones the PRM requires to guarantee finding a path, with a specified success probability.

The key concepts are:

- the LPM reachability of a point  $p$ , denoted by  $\mathcal{R}_{lpm}(p)$ ,
- the reachability of a set, denoted by  $\mathcal{R}(\mathcal{S})$ ,
- the  $\beta$ -lookout of a set  $\mathcal{S}$ , denoted by  $\beta\text{-lookout}(\mathcal{S})$ , and
- the  $(\alpha, \beta)$ -expansiveness of an environment.

## 5.2.2 Theoretical performance bound for finding a path with a PRM

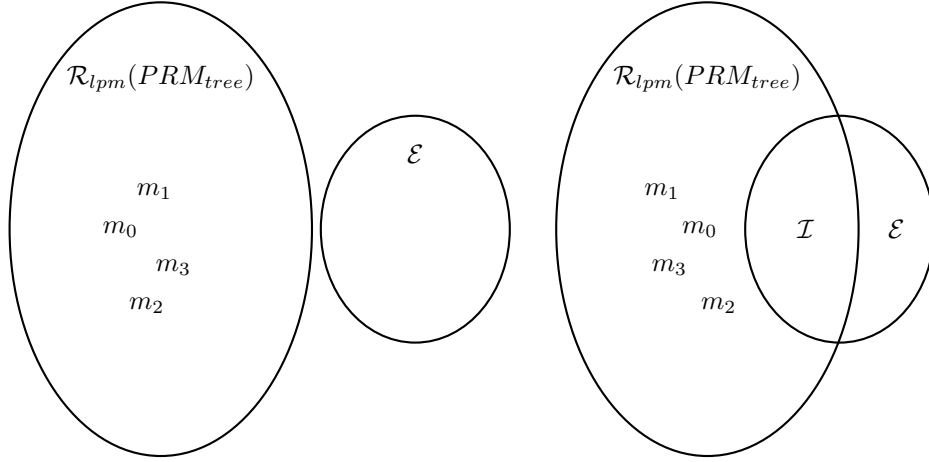
In order to determine a theoretical performance bound on the number of milestones that the PRM requires to find a path, the previous subsection (Subsection 5.2.1) first introduces several key concepts that are used in this subsection. The theorem by Hsu, et al. which is used to determine this performance bound is introduced in this subsection (Subsection 5.2.2). This subsection also presents a less conservative performance bound by rewriting some of the equations of the theorem by Hsu, et al.

### Introducing Cases A and B

For a path to exist between an initial and goal milestones it is necessary to sample new milestones until:

1. a sampled milestone  $m_{new}$  falls in a region where it can connect to the goal milestone using the LPM, and
2. at least one milestone in the  $PRM_{tree}$  can connect to  $m_{new}$ .

In other words, it is necessary to expand the LPM reachability  $\mathcal{R}_{lpm}(PRM_{tree})$ <sup>7</sup> until it intersects with the goal milestone, and then sample a milestone in this intersection to find a path. This forms the basis of the theorem by Hsu, et al.



(a) This figure illustrates the case where there is no intersection between the  $\mathcal{R}_{lpm}(PRM_{tree})$  and the endgame region.

(b) This figure illustrates the case where no milestones of a subset of all reachable milestones fall in the intersection between the endgame region and  $\mathcal{R}_{lpm}(PRM_{tree})$ .

**Figure 5.5** – Introducing Cases A and B.

For this discussion let  $\mathcal{E}$  denote a set of points wherein any point  $p$  is within the LPM reachable set of the goal milestone, formally let:

$$\mathcal{E} = \{p \in \mathcal{X} \mid \text{goal milestone} \in \mathcal{R}_{lpm}(p)\} \quad (5.2.13)$$

Furthermore, let  $\mathcal{I}$  denote the intersection between  $\mathcal{E}$  and  $\mathcal{R}_{lpm}(PRM_{tree})$ . It is necessary to sample a milestone  $m_{new}$  in  $\mathcal{I}$  to enable  $\mathcal{R}_{lpm}(PRM_{tree})$  to include the goal milestone. This concept is illustrated in Figures 5.5(a), 5.5(b), and 5.6. A path can only be formed between the initial and goal milestones when  $\mathcal{R}_{lpm}(PRM_{tree})$  includes the goal milestone.

Lastly, let the  $PRM_{tree}$  consist of two subsets of milestones,  $PRM_{tree} = \{M', M''\}$ , where  $M'$  consists of the first  $r'$  milestones of  $PRM_{tree}$ , and  $M''$  consists of the next  $r'' = r - r'$  milestones. In Figures 5.5(a), 5.5(b), and 5.6, let  $M' = \{m_0, m_1\}$  and  $M'' = \{m_2, m_3\}$

The theorem presented by Hsu, et al. defines two cases which leads to  $PRM_{tree}$  not containing a milestone in  $\mathcal{E}$ . Let the first case (Case A) be defined as the case where there

<sup>7</sup>Defined in Subsection 5.2.1.

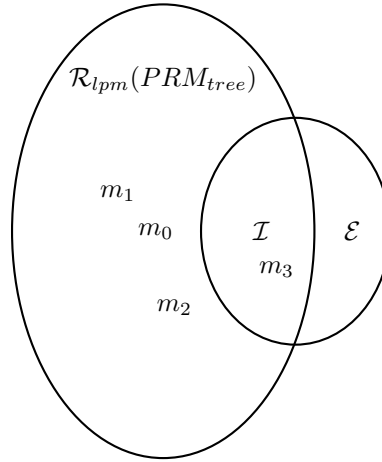
is no intersection between the  $\mathcal{R}_{lpm}(PRM_{tree})$  and the endgame region, as illustrated in Figure 5.5(a). Let the second case (Case B) be defined as the case that no milestone of  $M''$  is contained in  $\mathcal{I}$ , as illustrated in Figure 5.5(b).

### Introducing the union of Cases A and B

The union of cases A and B denotes the probability of not finding a path between the initial and goal milestones. Let the probability of not finding a path be denoted by  $\gamma$ . The union of Cases A and B is formally written as  $\mathcal{P}(A \cup B)$ , and using:

$$\mathcal{P}(A \cup B) \leq \mathcal{P}(A) + \mathcal{P}(B) \leq \gamma, \quad (5.2.14)$$

a lower bound on the probability of not finding a path can be determined if  $\mathcal{P}(A)$  and  $\mathcal{P}(B)$  is determined.



**Figure 5.6** – This figure illustrates a large intersection between the endgame region and  $\mathcal{R}_{lpm}(PRM_{tree})$ , as well as a milestone in the intersection. For this case a path is formed between the initial milestone and goal milestones.

### Divergence from the work by Hsu et al.

It is important to note that the work done for this project, and that done by Hsu et al. diverges from this point onwards. The work by Hsu et al. strives to determine an equation with which the relationship between the following is characterised:

1. the number of milestones the PRM requires to guarantee finding a path, with a specified probability, and
2. the probability of finding a path.

It is important that this characterisation shows that a number of milestones  $r$  can be determined, from a required success probability, that guarantees a path will be found. This is because it will show that: for any required success probability, there exists a number of milestones with which the PRM will find a path, even as that success probability strives towards one (or certainty). It is important since the proof of *probabilistic completeness* works on this premise.

In order to obtain such a characterisation, Hsu et al. made a few substitutions which causes the value of  $r$  to be more conservative than need be. However, for this analysis it is important to work with the least conservative value of  $r$ , while still providing a mathematical guarantee that a path will be found. The discussion on determining a performance bound for the PRM is continued below.

### Determining the probabilities of $\mathcal{P}(A)$ and $\mathcal{P}(B)$

The conditions necessary to determine the probability to *not* find a path is split up into two cases, A and B.

To determine  $\mathcal{P}(A)$ , Lemma 2 by Hsu et al. is used. To determine the lemma, it is necessary to determine the values of  $\alpha$  and  $\beta$ <sup>8</sup> that characterise the specific environment for which the performance bound is being calculated. Furthermore, let  $g = \mu(\mathcal{E})$  and  $k = \frac{1}{\beta} \ln(\frac{2}{g})$ . Note that  $r = r' + r''$  denotes the number of milestones in the  $PRM_{tree}$ . The lemma states that the probability of Case A is bounded by:

$$k(1 - \alpha)^{r'/k} \leq \mathcal{P}(A) \quad (5.2.15)$$

where the smallest value for  $r'$ <sup>9</sup> that satisfies the inequality is used.

To determine  $\mathcal{P}(B)$ , Lemma 1 by Hsu et al. is used. The Lemma is used to determine the volume of  $\mathcal{R}_{lpm}(PRM_{tree})$  relative to  $g$ :

$$\mu(\mathcal{R}_{lpm}(PRM_{tree})) \geq 1 - \frac{g}{2}. \quad (5.2.16)$$

Note that  $\mathcal{X}$  denotes the set of all points reachable from the initial milestone, and that all volumes used are scaled  $\mathcal{X}$  so that  $\mu(\mathcal{X}) = 1$ . Also note that the point sets  $\mathcal{R}_{lpm}(PRM_{tree})$  and  $\mathcal{E}$  must be a subset of  $\mathcal{X}$ <sup>10</sup>.

Since the volume of  $\mathcal{X}$  is 1, Equation 5.2.16 can be rewritten as:

$$\mu(\mathcal{R}_{lpm}(PRM_{tree})) \geq \mu(\mathcal{X}) - \frac{g}{2}, \quad (5.2.17)$$

$$\Rightarrow \mu(\mathcal{R}_{lpm}(PRM_{tree})) + \frac{g}{2} \geq \mu(\mathcal{X}), \quad (5.2.18)$$

which is only possible if  $\mathcal{R}_{lpm}(PRM_{tree})$  and  $\mathcal{E}$  overlap. Therefore, from Equation 5.2.18 an intersection  $\mathcal{I}$  exists with a volume  $\mu(\mathcal{I}) \geq \frac{g}{2}$ .

Since milestones are sampled uniformly over  $\mathcal{X}$ ,  $M''$  does not contain a milestone in  $\mathcal{I}$  with probability at least:

$$(1 - \mu(\mathcal{I}))^{r''} \leq \mathcal{P}(B), \quad (5.2.19)$$

$$\Rightarrow (1 - \frac{g}{2})^{r''} \leq \mathcal{P}(B). \quad (5.2.20)$$

Equations 5.2.15 and 5.2.20 can now be combined:

$$k(1 - \alpha)^{r'/k} + (1 - g/2)^{r''} \leq \gamma, \quad (5.2.21)$$

where  $r$  should be the smallest value that satisfies the inequality.

To determine a value of  $r$ , values for  $\alpha$ ,  $\beta$ , and  $g$  is required, and a choice for  $\gamma$  is required. The value of  $\gamma$  is the probability of not finding a path, and noting that the cases:

<sup>8</sup>The parameters  $\alpha$  and  $\beta$  are defined in Subsections 5.2.1.2 and 5.2.1.1, however determining its value is left until Subsection 5.2.3.

<sup>9</sup>I.e. the smallest amount of milestones necessary to find a path.

<sup>10</sup>Note that it is assumed a path between the initial and goal milestones is possible, hence  $\mathcal{R}_{lpm}(goal) = \mathcal{E}$  lies in  $\mathcal{R}(initial) = \mathcal{X}$ .

1. finding a path (**success**), and
2. not finding a path (**failure**),

are mutually exclusive, it is possible to state that:

$$\mathcal{P}(\mathbf{failure}) = \gamma, \quad (5.2.22)$$

$$\Rightarrow \mathcal{P}(\mathbf{success}) = 1 - \gamma, \quad (5.2.23)$$

whereby a value of  $\gamma$  can be determined for a required success probability. For example, a success probability of 99.99% or 0.9999 is required, using Equation 5.2.23 it is possible to determine that  $\gamma = 0.0001$ .

### Summary

To determine the probability of **not** finding a path between the initial and goal milestones, this subsection starts by introducing two cases (A and B) that are necessary to not find a path. Cases A and B are illustrated in Figures 5.5(a) and 5.5(b). Next, this subsection continues with introducing the union of Cases A and B in Equation 5.2.14, where it is seen that the union provides a bound on the probability of not finding a path, if the probabilities of Cases A and B are determined. Following this, the subsection continues with determining the probabilities  $\mathcal{P}(A)$  and  $\mathcal{P}(B)$ . The subsection then concludes with combining probabilities  $\mathcal{P}(A)$  and  $\mathcal{P}(B)$  to form Equation 5.2.21, which provides a bound on the number of milestones necessary to guarantee that the PRM will find a path.

### 5.2.3 Determining $\alpha$ and the $\beta$ -lookout( $\mathcal{S}$ ) for the PRM

A theorem by Hsu et al. which provides a performance bound on the number of milestones required to find a path is discussed in the previous subsection. The theorem by Hsu et al. is rewritten in a less conservative form, and the result is shown in Equation 5.2.21. In order to use Equation 5.2.21 to calculate a performance bound, values for  $\alpha$  and  $\beta$  need to be determined.

Two lemmas are presented in this subsection with which Equations 5.2.7 and 5.2.12 are rewritten, thereby enabling the calculation of  $\alpha$  and  $\beta$  for a specific environment and vehicle, using a computer.

#### 5.2.3.1 Probability with relative frequency

**Lemma 5.2.1** *For any set of points  $\mathcal{S}_1 \subset \mathcal{S}$  that has a volume larger than zero:*

$$\frac{\mu(\mathcal{S}_1)}{\mu(\mathcal{S})} = \mathcal{P}(p \in \mathcal{S}_1 \mid p \in \mathcal{S}), \quad (5.2.24)$$

where  $\mathcal{P}(A)$  denotes the probability of event  $A$ , and point  $p$  is uniformly chosen in  $\mathcal{S}$ .

**Lemma 5.2.2** *For  $n$  points chosen uniformly in  $\mathcal{S}$  and  $n_{p \in \mathcal{S}_1}$  points out of  $n$  landing in  $\mathcal{S}_1$ , the probability of a point  $p$  being in  $\mathcal{S}_1$  is:*

$$\lim_{n \rightarrow \infty} \frac{n_{p \in \mathcal{S}_1}}{n} = \mathcal{P}(p \in \mathcal{S}_1) \quad (5.2.25)$$

### 5.2.3.2 Determining $\alpha$ and the $\beta$ -lookout( $\mathcal{S}$ ) using relative frequency

Equation 5.2.7 states that the  $\beta$ -lookout( $\mathcal{S}$ ) is defined as the set of points,  $\{p \in \mathcal{S}\}$ , where:

$$\mu(\mathcal{R}_{lpm}(p) \setminus \mathcal{S}) \geq \beta \times \mu(\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}), \quad (5.2.26)$$

which means that a point  $p$  is in the  $\beta$ -lookout( $\mathcal{S}$ ) if the inequality is satisfied.

By dividing both sides by  $\mu(\bar{\mathcal{S}})$ , the volume not in  $\mathcal{S}$ , and rewriting Equation 5.2.26, the following is obtained:

$$\frac{\mu(\mathcal{R}_{lpm}(p) \setminus \mathcal{S})}{\mu(\bar{\mathcal{S}})} \geq \beta \times \frac{\mu(\mathcal{R}(\mathcal{S}) \setminus \mathcal{S})}{\mu(\bar{\mathcal{S}})}. \quad (5.2.27)$$

Let  $p_{\bar{\mathcal{S}}}$  be a newly sampled point in  $\bar{\mathcal{S}}$ . By using Lemma 5.2.1, with  $p_{\bar{\mathcal{S}}} \in \bar{\mathcal{S}}$ , it is possible to write:

$$\mathcal{P}(p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)) \geq \beta \times \mathcal{P}(p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})). \quad (5.2.28)$$

Next, Lemma 5.2.2 is used to write the following: for  $n$  uniformly distributed new points in  $\bar{\mathcal{S}}$ ,  $n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}$  denotes the number of new points sampled in  $\mathcal{R}_{lpm}(p) \setminus \mathcal{S}$ . Similarly, for  $m$  uniformly distributed new points in  $\bar{\mathcal{S}}$ ,  $m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}$  denotes the number of new points sampled in  $\mathcal{R}(\mathcal{S}) \setminus \mathcal{S}$ :

$$\lim_{n \rightarrow \infty} \frac{n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}}{n} \geq \beta \times \lim_{m \rightarrow \infty} \frac{m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}}{m}. \quad (5.2.29)$$

By using Equation 5.2.29, it can be determined whether any sampled point  $p$  in  $\mathcal{S}$  is part of the set of points  $\{p \in \beta\text{-lookout}(\mathcal{S})\}$ , by:

1. sampling  $n$  points in  $\bar{\mathcal{S}}$  and testing how many points lie within  $\mathcal{R}_{lpm}(p)$  (denoted by  $n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}$ ),
2. sampling  $m$  points in  $\bar{\mathcal{S}}$  and testing how many points lie within  $\mathcal{R}(\mathcal{S})$  (denoted by  $m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}$ ), and
3. evaluating whether  $\beta \times \frac{m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}}{m} \leq \frac{n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}}{n}$ ,

where the number of samples for  $n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}$  and  $m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}$  must be enough to provide consistent approximations<sup>11</sup> for the ratios  $\frac{n_{p_{\bar{\mathcal{S}}} \in \mathcal{R}_{lpm}(p)}}{n}$  and  $\frac{m_{p_{\bar{\mathcal{S}}} \in \mathcal{R}(\mathcal{S})}}{m}$ .

Now that it is possible to determine the  $\beta$ -lookout( $\mathcal{S}$ ), it is possible to move on to determining  $\alpha$ . From Subsection 5.2.1.2:

$$\alpha \leq \frac{\mu(\beta\text{-lookout}(\mathcal{S}))}{\mu(\mathcal{S})}, \quad (5.2.30)$$

and by noting that the  $\mu(\beta\text{-lookout}(\mathcal{S})) \subset \mu(\mathcal{S})$ , Lemma 5.2.1 may be used to write:

$$\alpha \leq \mathcal{P}(p \in \beta\text{-lookout}(\mathcal{S})), \quad (5.2.31)$$

and Lemma 5.2.2 may be used to write that:

$$\alpha \leq \lim_{k \rightarrow \infty} \frac{k_{p \in \beta\text{-lookout}(\mathcal{S})}}{k} \quad (5.2.32)$$

which enables the calculation of  $\alpha$  for a specific  $\beta$ -lookout( $\mathcal{S}$ ), on a computer.

<sup>11</sup>By determining each ratio multiple times and comparing multiple results from one ratio, it is possible to determine how many samples are required to accurately approximate the ratios.



### The relationship between $\alpha$ and $\beta$

In the process to determine values for  $\alpha$  and  $\beta$ , it is important to note two choices; it is possible to:

1. choose a range of values for  $\beta \in (0, 1]$ , where each value in the range will correspond to a bound on the value of  $\alpha$ , or
2. choose a range of values for  $\alpha \in (0, 1]$ , where each value in the range will correspond to a bound on the value of  $\beta$ .

**Choice 1** For a specific value of  $\beta$ , a specific set of points  $\beta$ -lookout( $\mathcal{S}$ ) exists (Equation 5.2.7), which has a specific volume,  $\mu(\beta$ -lookout( $\mathcal{S}$ )). From this volume a bound on the size of  $\alpha$  can be determined through Equation 5.2.12. By looking at Equation 5.2.21, specifically the term  $k(1 - \alpha)^{r/k}$ , it is noticed that a value of  $\alpha$  closer to 1 than 0 allows the aforementioned term to decrease in value faster<sup>12</sup> for increasing values of  $r$ . This means that the inequality of Equation 5.2.21 is satisfied for smaller values of  $r$  when larger values of  $\alpha$  is used. Therefore, the largest value of  $\alpha$ , allowable by its bound, is the optimum choice for  $\alpha$ , as a smaller number of milestones are required to guarantee the PRM will find a path.

**Choice 2** For a specific value of  $\alpha$ , a range of volumes  $\mu(\beta$ -lookout( $\mathcal{S}$ )) exist (Equation 5.2.12), where each of these volumes has a specific value of  $\beta$  associated with it (Equation 5.2.7). However, determining the associated value of  $\beta$  is cumbersome since it is not possible to directly determine  $\beta$  from only the volume of the set  $\beta$ -lookout( $\mathcal{S}$ ). For a range of values for  $\beta$  the corresponding volumes will have to be calculated until a calculated volume matches the volume of the set  $\beta$ -lookout( $\mathcal{S}$ ), as specified by  $\alpha$ .

**Choices 1 vs. 2** The amount of operations required to calculate  $\beta$  from  $\alpha$  (Choice 2) is significantly more than that required by calculating  $\alpha$  from  $\beta$  (Choice 1). In the next subsection (Subsection 5.2.4 the implementation of Choice 1 is discussed.

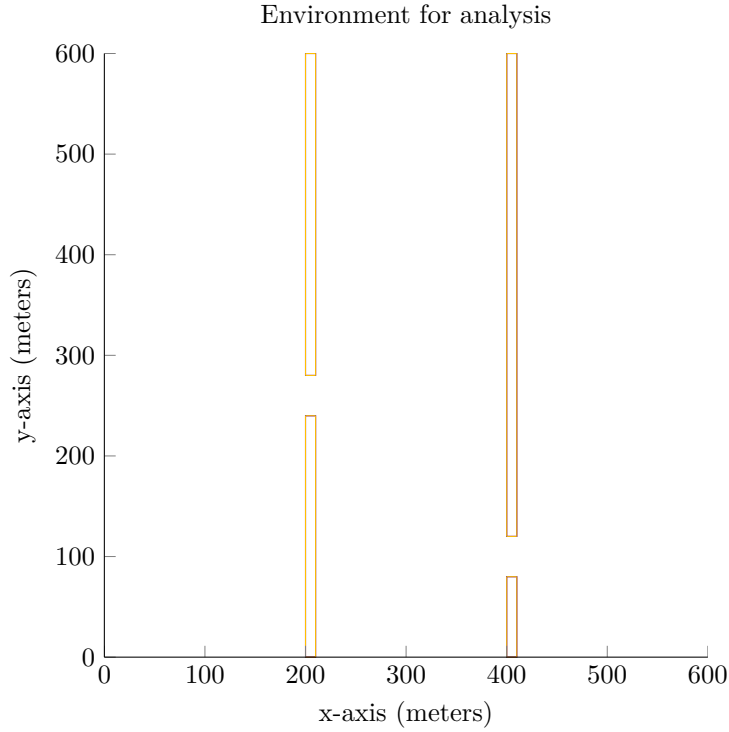
### Summary

This subsection continues the work on determining a bound on the number of milestones required to guarantee the PRM will find a path. The concept of relative frequency is introduced with Lemmas 5.2.1 and 5.2.2, which enabled the rewriting of Equations 5.2.7 and 5.2.12 into Equations 5.2.32 and 5.2.29. It is seen that two choices are available to calculate  $\alpha$  and  $\beta$ , and after a short discussion it is decided that the implementation of Choice 1 will be presented in the next subsection (Subsection 5.2.4).

### 5.2.4 Relating $\alpha$ and $\beta$ on a computer for a specific environment and a specific vehicle

This subsection uses equations introduced in the previous subsection to relate  $\alpha$  and  $\beta$ , determine values for each, and then calculate a performance bound on the number of milestone that will guarantee the PRM will find a path, for a specific environment and vehicle. Figure 5.7 shows the environment.

<sup>12</sup>Note that the value of  $\alpha$  lies in  $(0, 1]$ .



**Figure 5.7** – The environment wherein the PRM and RRT\* algorithms are analysed.

Choice 1 from the previous subsection is chosen to be implemented for this project. In order to establish the relationship between  $\alpha$  and  $\beta$ , a vector  $\mathbf{v}_\beta$  of values for  $\beta$  (increasing with a specified step size from close to zero to one) is created, and then  $\alpha$  is determined for each value of  $\beta$  in this vector. The process starts by determining the points that lie in the  $\beta$ -lookout( $\mathcal{S}$ ), hence the process to determine whether a single point lies in the  $\beta$ -lookout( $\mathcal{S}$ ) is first presented.

#### Determining whether a single point $p$ lies in $\beta$ -lookout( $\mathcal{S}$ )

The process starts by first determining the right-hand side  $\left(\beta \times \frac{m_{p\bar{\mathcal{S}} \in \mathcal{R}(\mathcal{S})}}{m}\right)$  of Equation 5.2.29, which is achieved by sampling  $m$  points in  $\bar{\mathcal{S}}$ , and testing how many of these points lie in  $\mathcal{R}(\mathcal{S})$ . To determine whether a point lies in  $\mathcal{R}(\mathcal{S})$ , a tree is grown backwards for each of the  $m$  points until the tree can be connected to the initial milestone, or the number of milestones in the tree exceed some specified number  $e$ <sup>13</sup>.

Next, the left-hand side  $\left(\frac{n_{p\bar{\mathcal{S}} \in \mathcal{R}_{LPM}(p)}}{n}\right)$  of Equation 5.2.29 is determined by sampling a point  $p$  in  $\mathcal{S}$ , and then sampling  $n$  points in  $\bar{\mathcal{S}}$ . For each of the  $n$  points, the LPM and *conflict detection* algorithms are used to determine the number points to which  $p$  can be connected to in  $\bar{\mathcal{S}}$ . The point  $p$  lies in the  $\beta$ -lookout( $\mathcal{S}$ ) if the left-hand side is larger than the right hand side.

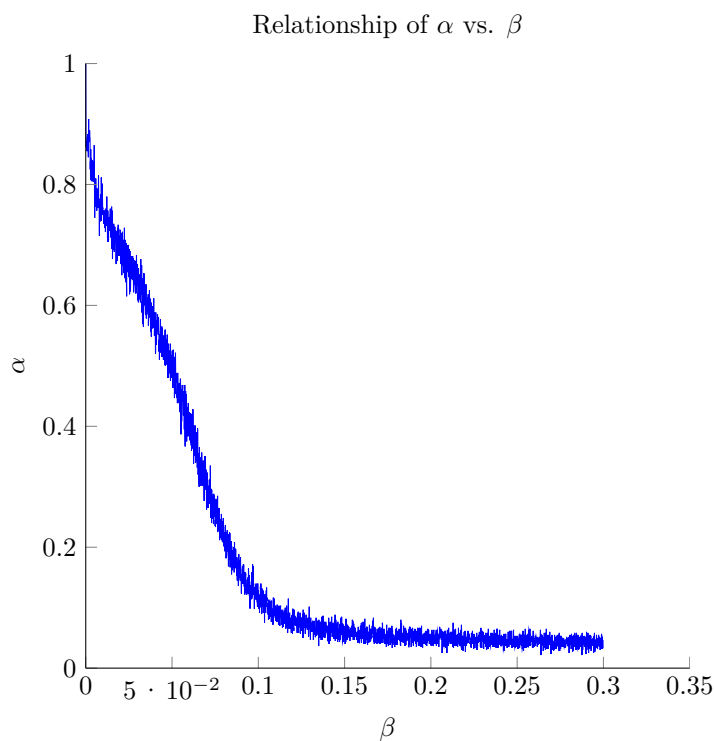
<sup>13</sup>As long as the choice of  $e$  is significantly larger than the predicted amount of milestones necessary to guarantee finding a path, the choice of  $e$  is deemed acceptable.

### Determining the set of points $\beta$ -lookout( $\mathcal{S}$ )

To determine the set of points  $\beta$ -lookout( $\mathcal{S}$ ) the above process is repeated  $a$  times, each time sampling a new point  $p$ . The number of  $p$  points that fall in  $\beta$ -lookout( $\mathcal{S}$ ) divided by  $a$  denotes the volume of  $\beta$ -lookout( $\mathcal{S}$ ) relative to  $\mathcal{S}$ . Note that the right-hand side of Equation 5.2.29 does not change for a newly sampled point, hence it is only determined once.

### Determining values for $\alpha$

After the set of volume  $\mu(\beta$ -lookout( $\mathcal{S}$ )) is determined, Equation 5.2.32 is used to determine a value for  $\alpha$ . The process of determining a value for  $\alpha$  from a value of  $\beta$  is now completed; however, it is necessary to repeat the process for the rest of the values of  $\beta$  in  $\mathbf{v}_\beta$ . When this is completed, the values in  $\mathbf{v}_\beta$  can be plotted against the determined values of  $\alpha$ , as in Figure 5.8.

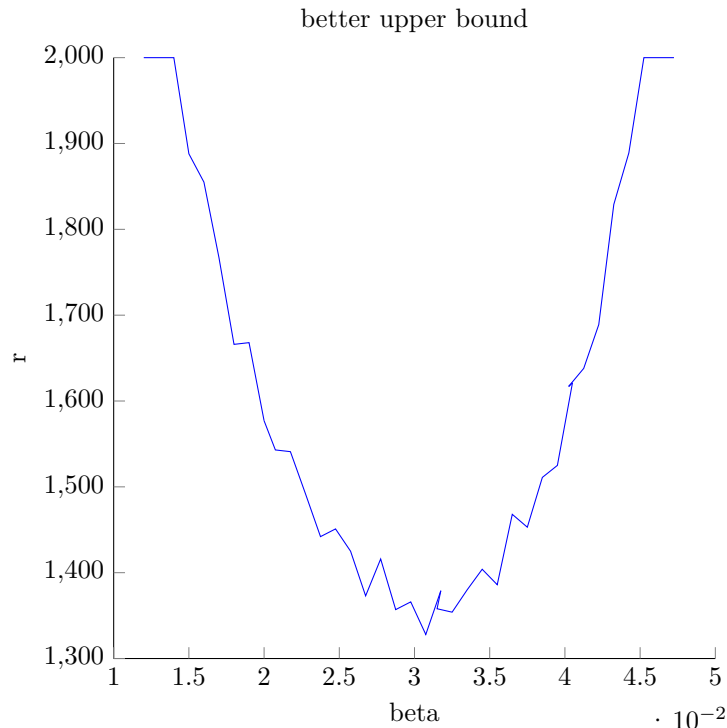


**Figure 5.8** – Plot of  $\alpha$  vs  $\beta$  in the environment shown in Figure 5.7.

From the values of  $\alpha$  and  $\beta$  in Figure 5.8, it is possible to plot  $r$  vs.  $\beta$  by using Equation 5.2.21. To calculate values for  $r$  it is necessary to specify a success percentage  $(1 - \gamma)$ . The success percentage denotes the probability that the PRM will not require more than  $r$  milestones to find a path. For this analysis, and that done for the RRT\*, a success percentage of 99.99%, or 0.9999 is chosen, and by using Equation 5.2.23 it is determined that  $\gamma = 0.0001$ .

It is important to note that all the calculated values for  $r$  denotes a number of milestones that will guarantee the PRM will find a path. Therefore, the smallest value of  $r$  in Figure 5.9

is the least conservative guarantee of the number of milestones that will guarantee the PRM will find a path.



**Figure 5.9** – Upper-bound plot of the number of milestones necessary for a 99.99% probability that the PRM will not require more than  $r$  milestones to find a path vs  $\beta$  for the environment shown in Figure 5.7. The smallest value of  $r = 1328$ , with  $\beta = 0.03075$  and  $\alpha = 0.4408$

### 5.2.5 Conclusion

In this section parts of the proof for probabilistic completeness of the PRM is used to calculate a theoretical bound on the number of milestones that will guarantee the PRM will find a path. This is done for a specific environment as well as a specific vehicle. The theorem used for the proof of probabilistic completeness is developed by Hsu et al. [16], however, parts of the theorem is rewritten in order to obtain a less conservative performance bound.

Following this, two lemmas are presented which enabled the rewriting of ratios of volumes to probabilities, which enabled the use of relative frequency, which allowed these probabilities to be approximated on a computer. After all the equations required to determine the PRM's performance bound is rewritten into relative frequency form, values are calculated for the upper bound. In the next section the RRT\* is analysed in a similar manner.

## 5.3 RRT\* Analysis

Some of the problems presented by the problem statement (Section 3.1) are solved using the PRM and RRT\* motion planning algorithms. The motion planning algorithms are implemented in Sections 4.8 and 4.9, respectively, and the PRM is analysed in the previous section (Section 5.2). The analysis is conducted to provide insight into the behaviour of

the algorithms in specific environments, with the end goal of being able to determine which algorithm performs better in a given environment. This section provides the analysis of the RRT\* algorithm.

The RRT\* is analysed using a theorem presented by S.M. LaValle and J.J. Kuffner [20], which provides an upper bound on the performance of the RRT\*, in a specific environment. It is seen that the theorem works by determining key areas wherein a random milestone must be sampled such that a path can be formed between the initial and the goal milestones. If the probability of generating a milestone in each of these key areas can be determined, then performance bounds can be calculated for the RRT\*, for a specific environment. As with the PRM, the theorem used for the RRT\* depends on parameters that are not easily determined. Once again non-analytical methods are employed to estimate values for these parameters.

Regarding the choice of  $\eta$ , another insight is gained from this analysis. The area within a ball of radius  $\eta$  is searched for better paths towards and through  $\mathbf{m}_{rand}$ , as shown in Subsection 4.9.2. The larger the value of  $\eta$ , the more milestones this ball will contain, and for each milestone therein the path optimisation code has to be executed. Thus the choice of  $\eta$  will indirectly influence the CPU-time required for the path optimisation of the RRT\*.

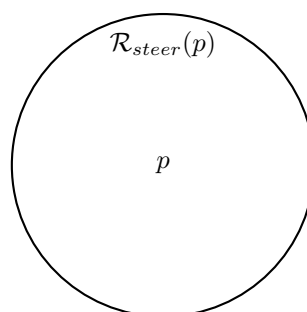
### 5.3.1 Theoretical Upper bound for finding a path with a RRT\*

For the analysis of the RRT\*, a theorem and notation by LaValle is introduced. The theorem forms part of the proof for probabilistic completeness of the RRT\*, and is used specifically to determine an upper bound on the iterations necessary to find a path<sup>14</sup>, in a given environment.

Since the RRT\* uses manoeuvres to connect subsequent milestones, the results of this analysis is vehicle specific; that is, the results of this analysis is only valid for vehicles that can execute the manoeuvre libraries of Subsection 4.5, and for vehicles incapable of doing so, this analysis will yield different results.

In order to analyse the RRT\*, a few concepts are first introduced that are similar to those introduced for the PRM (Section 5.2.1), however, there are subtle differences, and therefore the concepts of this section should be viewed as independent of those in Section 5.2.

#### Introducing the concepts of the Steer reachability set

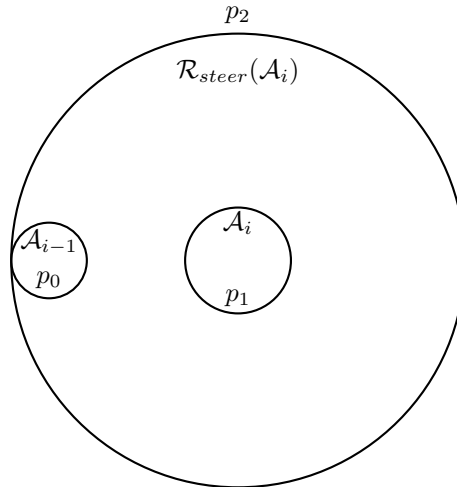


**Figure 5.10** – Steer reachability of  $p$ .

<sup>14</sup>A path is found with a guaranteed success percentage.

Figure 5.10 shows a point  $p$ , and a set of points  $\mathcal{R}_{steer}(p)$  enclosed by a circle. Let the set  $\mathcal{R}_{steer}(p)$  denote the set of points **from which**  $p$  can be reached through a single Steer<sup>15</sup> run.

### Introducing the concept of an attraction set



**Figure 5.11** – Attraction set of  $\mathcal{A}_i$ .

Figure 5.11 shows three points,  $p_0, p_1, p_2$ , and three sets of points,  $\mathcal{A}_{i-1}, \mathcal{A}_i$ , and  $\mathcal{R}_{steer}(\mathcal{A}_i)$ . The sets  $\mathcal{A}_{i-1}$  and  $\mathcal{A}_i$  form an **attraction sequence**, which is indicated by the fact that any point outside of  $\mathcal{R}_{steer}(\mathcal{A}_i)$  is further than any point in the set  $\mathcal{A}_{i-1}$  to any point in the set  $\mathcal{A}_i$ , and any point in  $\mathcal{R}_{steer}(\mathcal{A}_i)$  can connect to any point in  $\mathcal{A}_i$ .

Informally, points contained within an attraction sequence must satisfy two requirements: let  $\rho(p_0, p_1)$  denote some distance metric, i.e.  $\rho(p_0, p_1) = \sqrt{(x_{p_0} - x_{p_1})^2 + (y_{p_0} - y_{p_1})^2}$ ,

1. for a point  $p_0$  in  $\mathcal{A}_{i-1}$ , a point  $p_1$  in  $\mathcal{A}_i$ , and a point  $p_2$  anywhere outside of  $\mathcal{R}_{steer}(\mathcal{A}_i)$ ,  $\rho(p_1, p_0) < \rho(p_1, p_2)$  must hold, and
2. any point within  $\mathcal{R}_{steer}(\mathcal{A}_i)$  can be connected to  $\mathcal{A}_i$  through a single manoeuvre, using the Steer algorithm.

---

<sup>15</sup>defined in Subsection 4.9.1.

## Introducing the concepts of the Attraction sequence

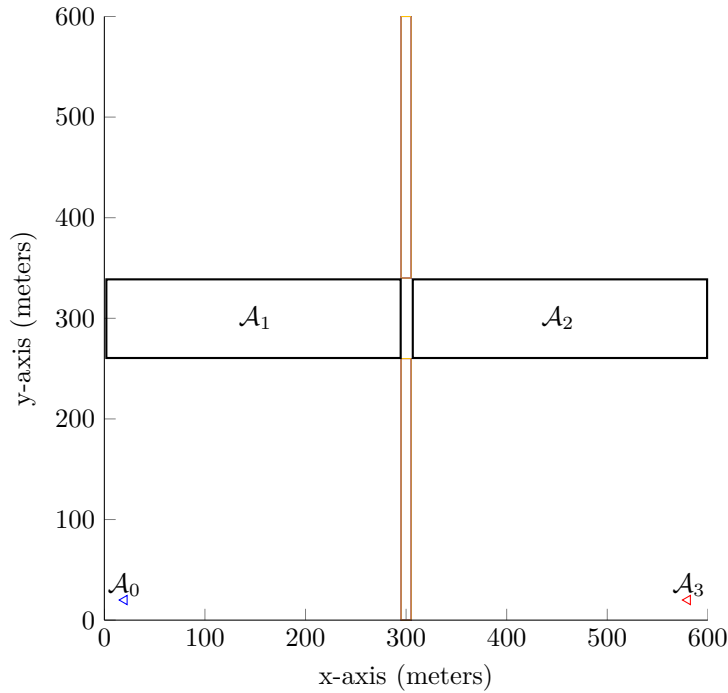
Figure 5.12 – Attraction set of  $\mathcal{A}_1$ .

Figure 5.12 shows four sets of points:

1. the set  $\mathcal{A}_0$ , which contains only a single point, the initial point,
2. the set  $\mathcal{A}_1$ ,
3. the set  $\mathcal{A}_2$ , and
4. the set  $\mathcal{A}_3$ , which contains only a single point, the goal point.

The sets are chosen so that  $\mathcal{A}_0$  falls in  $\mathcal{R}_{steer}(\mathcal{A}_1)$ ,  $\mathcal{A}_1$  falls in  $\mathcal{R}_{steer}(\mathcal{A}_2)$ , and  $\mathcal{A}_2$  falls in  $\mathcal{R}_{steer}(\mathcal{A}_3)$ . This means that each set falls within the Steer reachability of a subsequent set, and it is therefore possible to form a path between the initial and goal points using the Steer algorithm, provided that the sets  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are non-empty.

The sequence of attraction sets,  $\mathcal{A} = \{\mathcal{A}_0, \dots, \mathcal{A}_3\}$ , form an **attraction sequence** of length 4. Next, this concept is formally defined.

### 5.3.1.1 The Attraction sequence

Let  $\mathcal{F}$  denote the set of all conflict-free points,  $\mathcal{S}$  denote the state space of the vehicle, and  $\mathcal{T}$  denote the time space. Furthermore, let  $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_k\}$  be a sequence of subsets of  $\mathcal{F} \subset \mathcal{ST}$ , referred to as an **attraction sequence**, where each  $\mathcal{A}_i$  contains points in  $\mathcal{F} \subset \mathcal{ST}$ .

Formally, for all  $\mathcal{A}_i \in \mathcal{A}$  there must exist a Steer reachability set  $\mathcal{R}_{steer}(\mathcal{A}_i) \subset \mathcal{ST}$ , such that:

1. the points  $p_0 \in \mathcal{A}_{i-1}$ ,  $p_1 \in \mathcal{A}_i$ , and  $p_2 : \{p_2 \notin \mathcal{R}_{steer}(\mathcal{A}_i) \mid p_2 \in (\mathcal{F} \subset \mathcal{ST})\}$ , the inequality  $\rho(p_1, p_0) < \rho(p_1, p_2)$  holds, and
2. any point  $p_3 \in \mathcal{R}_{steer}(\mathcal{A}_i)$  it is possible to use the Steer algorithm to generate a single manoeuvre that will connect  $p_3$  to any point  $p_1 \in \mathcal{A}_i$ , where  $\mathcal{A}_i \subset \mathcal{R}_{steer}(\mathcal{A}_i)$ ,

with the exception that  $\mathcal{A}_0 = \{m_b\}$ , the initial milestone, and  $\mathcal{A}_k = \{m_g\}$ , the goal milestone.

Note that the choice of the parameter  $\eta$  will directly influence the size of  $\mathcal{R}_{steer}(\mathcal{A}_i)$ ; however, a discussion hereof is only presented later.

### 5.3.1.2 Probability of sampling a milestone in each attraction set

Let  $\mu(\mathcal{A}_i)$  denote the volume of a set of points  $\mathcal{A}_i$ , and let  $p$  equal the smallest  $\mu(\mathcal{A}_i)$ , scaled by the volume of the total free space:

$$p = \left( \frac{\mu(\mathcal{A}_i)_{smallest}}{\mu(\mathcal{F} \subset \mathcal{ST})} \right). \quad (5.3.1)$$

Note that the value of  $p$  is the probability of sampling a random milestone in  $\mathcal{A}_i$ . Next, Theorem 2 of LaValle is presented:

If an attraction sequence of length  $k$  exists, the probability  $\gamma \in (0, 1]$  that the RRT\* fails to find a path after  $n$  iterations is governed by the inequality:

$$\gamma < e^{-\frac{1}{2}(np-2k)} \quad (5.3.2)$$

or rewritten, the number of iterations required to guarantee a failure rate of at most  $\gamma$ :

$$n < \frac{2 \ln(\gamma) + 2k}{p} \quad (5.3.3)$$

Since the events of finding a path, and not finding a path, are mutually exclusive, it is possible to state that:

$$\mathcal{P}(\mathbf{fail}) = \gamma, \quad (5.3.4)$$

$$\Rightarrow \mathcal{P}(\mathbf{success}) = 1 - \gamma. \quad (5.3.5)$$

The above result, together with Equation 5.3.3, enables a performance bound to be defined for the RRT\*: by ensuring that,

$$n \geq \frac{2 \ln(\gamma) + 2k}{p}, \quad (5.3.6)$$

a  $1 - \gamma$  success probability is guaranteed.

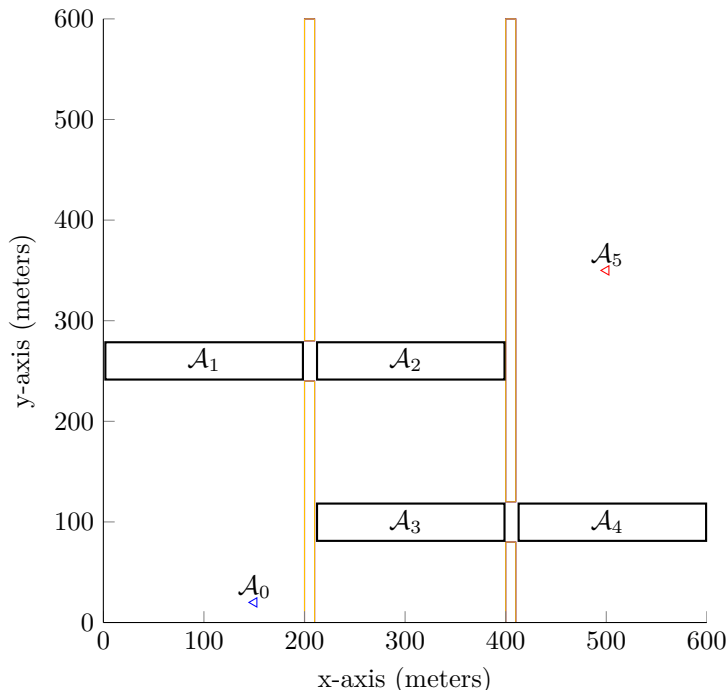
### 5.3.1.3 Determining the location of the attraction sequence

Previously, concepts and equations necessary to understand a theorem by LaValle and Kuffner are introduced, as well as the theorem itself. The theorem provides a performance bound on the RRT\*, and is presented in Equation 5.3.6.

For the performance bound on the number of iterations  $n$  the RRT\* requires to guarantee it finding a path, with a specified success probability, the value of  $n$  needs to be as conservative (or small) as possible, while still adhering to the mathematical bound of Equation 5.3.6,



which means that, the values of  $p$  and  $k$  should minimise  $n$ . By analysing Equation 5.3.6, it is seen that this is achieved by using a value of  $p$  that is as large as possible, and as small as possible a value for  $k$ . The values of  $p$  and  $k$  are directly influenced by the location of the subsets of the attraction sequence  $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_k\}$ , as choosing locations for the subsets might require a larger than necessary  $k$ , or a very small value for  $p$ .



**Figure 5.13** – Six subsets ( $k = 5$ ) are shown. A path between the initial and goal milestones is formed by sampling a milestones in the subsets 1 to 4.

Figure 5.13 shows six attractions sets, which form an attraction sequence of length  $k = 5$ , and by sampling a random milestone in each set,  $\mathcal{A}_1$  to  $\mathcal{A}_4$ , a path between the initial ( $\mathcal{A}_0$ ) and goal ( $\mathcal{A}_5$ ) milestones is formed.

Other choices for the locations of the attractions sets are possible. Intuitively, the locations shown in Figure 5.13 allows a small value of  $k$ , and a large value of  $p$ , given the positions of  $\mathcal{A}_0$  and  $\mathcal{A}_5$ ; however, ensuring that the locations are optimal, remains an open problem. It is important to note that, independent of the locations of the attraction sequence, Equation 5.3.6 still provides a mathematical bound on  $n$ , even when the locations yield a conservative value for  $n$ .

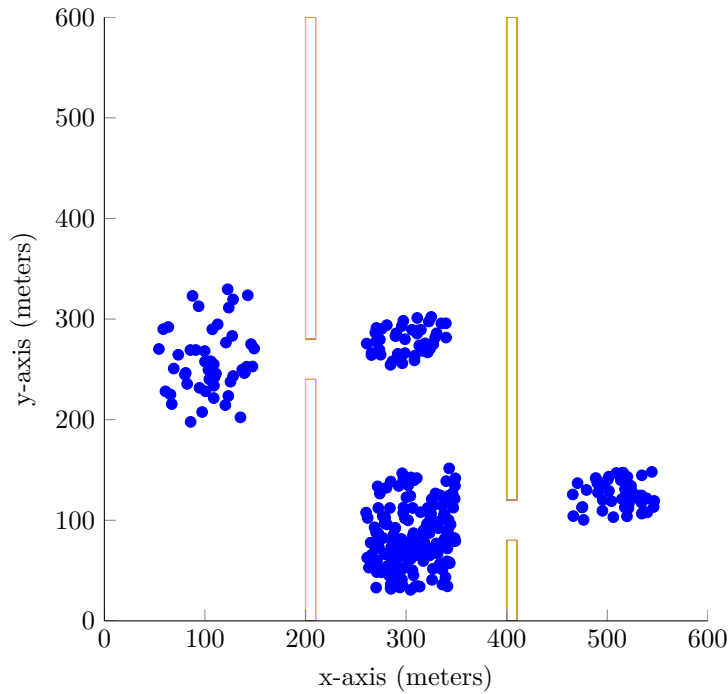
For the attraction sequence, the value of  $\eta$  must be chosen so that each  $\mathcal{A}_{i-1}$  have to be in the set  $\mathcal{R}_{steer}(\mathcal{A}_i)$ , while keeping the value of  $\eta$  to a minimum so that the path optimisation outlined in Subsection 4.9.2 does not become too costly. In regard to this,  $\eta$  is chosen to be exactly equal to the maximum distance between subsequent attraction sets, thereby satisfying both requirements.

### 5.3.1.4 Determining the volume of the subsets of the attraction sequence

To determine the value of  $p$  it is necessary to determine the volume of the subsets of the attraction sequence. The volume of the subsets is determined by using relative frequency

(defined in Lemma 5.2.2), as was done in Section 5.2.2 for the PRM. The concept of relative frequency works by generating numerous random samples, and by comparing the amount of samples that fall in a specific *region* to the total amount of generated samples, the volume of *region* is approximated<sup>16</sup>.

The attraction sequence shown in Figure 5.13 adheres to the rules presented in Subsection 5.3.1.1, and using these rules it is possible to determine whether a sampled milestone lies in an attraction set or not. For example, let a milestone  $m_{rand}$  be sampled at coordinates  $x = 100m$ ,  $y = 300m$  in Figure 5.13. According to the rules set out in Subsection 5.3.1.1, for  $m_{rand}$  to form part of the attraction set  $\mathcal{A}_1$ , it must lie in  $\mathcal{R}_{steer}(\mathcal{A}_2)$ . However,  $m_{rand}$  does not lie in  $\mathcal{R}_{steer}(\mathcal{A}_2)$  as the manoeuvre connecting  $m_{rand}$  to some of the points in  $\mathcal{A}_2$  is not entirely conflict free. Hence, the milestone  $m_{rand}$  is not considered part of  $\mathcal{A}_1$ . In a similar manner, any sampled milestone can be tested whether it lies in a specific subset of the attraction sequence, and therewith the volume of the respective subsets can be determined by using relative frequency. The results of the above described procedure is shown below in Figure 5.14.



**Figure 5.14** – This figure shows an example attraction sequence as determined by the approach in Subsection 5.3.1.4.

By using the approach outlined in this subsection,  $p$  is calculated to be 0.003926, and therewith a value for  $n$  can be calculated by using Equation 5.3.6. For a required success probability of  $\mathcal{P}(\text{success}) = 0.9999$  (or 99.99%),  $\gamma = 0.0001$ , and  $n$  is calculated to be 7174 iterations.

<sup>16</sup>Note that for this discussion the time dimension is left out for clarity, but the arguments presented here are valid for higher dimensions as well.

### 5.3.2 Conclusion

This section introduces parts of the probabilistic completeness proof for the RRT\*. These parts are then used to calculate a theoretical bound on the performance of the RRT\*. Firstly, this section introduces the concept of the Steer reachability set of a point, and then continues to introduce the concept of an attraction sequence. By sampling a milestone in each subset of the attraction sequence, a path between the initial and goal milestones is found. To calculate the probability of sampling a milestone in each subset of the attraction sequence, a theorem by LaValle [20] is introduced. The theorem by LaValle relates the number of samples which are necessary to guarantee sampling a milestone in each subset to a specified probability. To calculate the performance bound of the theorem by LaValle, it is necessary to calculate the volumes of the subsets of the attraction sequence, and relative frequency<sup>17</sup> is used to achieve this. Lastly, implementation details are discussed and the performance bound for the RRT\* is calculated as 7174 iterations (or samples of milestones) to guarantee a 99.00% probability of finding a path.

The calculated performance bound for the RRT\* (together with the performance bound on the PRM) is analysed and compared to practical results, in the next section (Section 5.4).

## 5.4 Histogram analysis

In the previous two sections the PRM and RRT\* motion planning algorithms are analysed. Their analyses allows theoretical bounds to be calculated on the amount of milestones (or iterations) required to find a path in a specific environment. In this section, the PRM and RRT\* algorithms are compared using histograms, for the environment shown in Figure 8.1(a). This subsection provides real measurements of the iterations, milestones, run times and path costs of the PRM and RRT\* algorithms during execution. These measurements are presented in histogram form and they are used to assert the theoretical results from the two previous sections.

### 5.4.1 Iteration count

In this subsection, the iterations required by the PRM and RRT\* algorithms to find a path, is compared. The analysis of the RRT\* provides us with a theoretical bound on the number of iterations the algorithm requires to find a path, whereas the analysis of the PRM provides us with a theoretical bound on the number of milestones in its tree. It is still important to compare iteration measurements for both algorithms as it provides us with insight into the differences between the PRM and RRT\*.

Figure 5.15 shows us a histogram of the amount of iterations the PRM algorithm required to find a path in the above environment, as does Figure 5.16 for the RRT\*. From these figures, it is possible to see that the PRM algorithm required less iterations than the RRT\* algorithm. Next, the two algorithms are compared in order to identify the cause for this result.

For the environment shown in Figure 8.1(a) suppose that the RRT\* tree has grown from the initial milestone to just before the first wall, and there exists a milestone  $\mathbf{m}_{nearest}$  in the tree at position  $x = 190$  and  $y = 400$ . Suppose that the algorithm now generates a new random milestone  $\mathbf{m}_{new}$  just to the right of the wall (i.e. so that  $x > 210$ ) so that the nearest neighbour search returns  $\mathbf{m}_{nearest}$ . The Steer method of the RRT\* will now attempt to connect the two milestones, however it cannot because of the wall, and now the

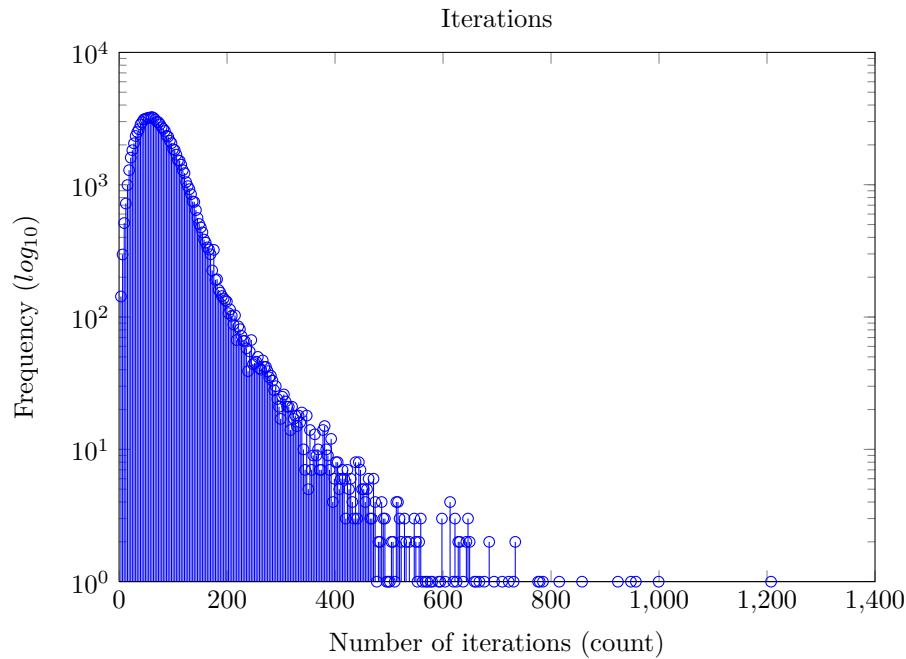
---

<sup>17</sup>Introduced in Subsection 5.2.3.1

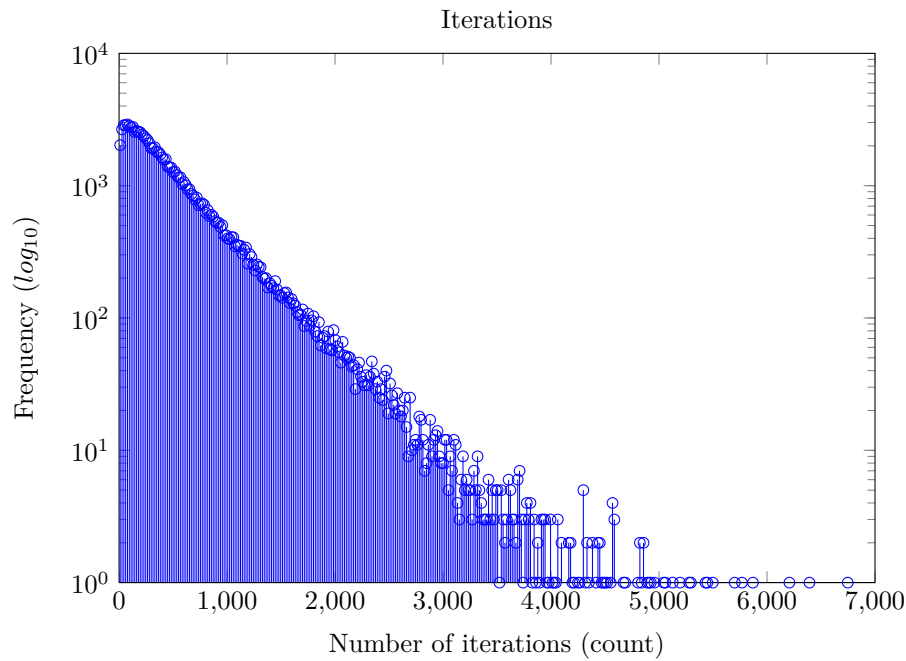
milestone  $\mathbf{m}_{new}$  is discarded. This means that even though  $\mathbf{m}_{new}$  might be contained in  $\mathcal{R}_{steer}(RRT^*-tree)$  it is still discarded.

For the same case as above the classic PRM algorithm will loop though its entire tree to try and connect  $\mathbf{m}_{new}$  to its tree. This means that as long as  $\mathbf{m}_{new}$  lies within  $\mathcal{R}_{lpm}(PRM-tree)$  it will be connected. However, in Subsection 4.8.2 a change is proposed where the entire PRM tree is not searched for a parent milestone, which means that it is not guaranteed that as long as  $\mathbf{m}_{new}$  lies within  $\mathcal{R}_{lpm}(PRM-tree)$  it will be connected. While this guarantee is dropped the PRM still tries longer than the RRT\* to connect  $\mathbf{m}_{new}$  to its tree.

From this discussion it is evident that the PRM algorithm does not “waste” as many iterations as the RRT\* which results in less iterations. The average iterations for the PRM and RRT\* algorithms are 80 and 502 respectively.



**Figure 5.15** – Histogram plot of iterations required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs.



**Figure 5.16** – Histogram plot of iterations required by the RRT\* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. The measurements show that 0 times out of 100000 runs the algorithm required more than 7174 iterations.

#### 5.4.1.1 Theoretical versus Practical iteration requirement

In Section 5.3.1 a theoretical performance upper bound on the number of algorithm iterations the RRT\* requires to find a path is calculated. In this subsection, the theoretical bound is compared with practical measurements of the RRT\* algorithm. The same is done for the PRM algorithm, however, its performance bound is on the number of milestones it requires. The comparison for the PRM is done in Subsection 5.4.2.

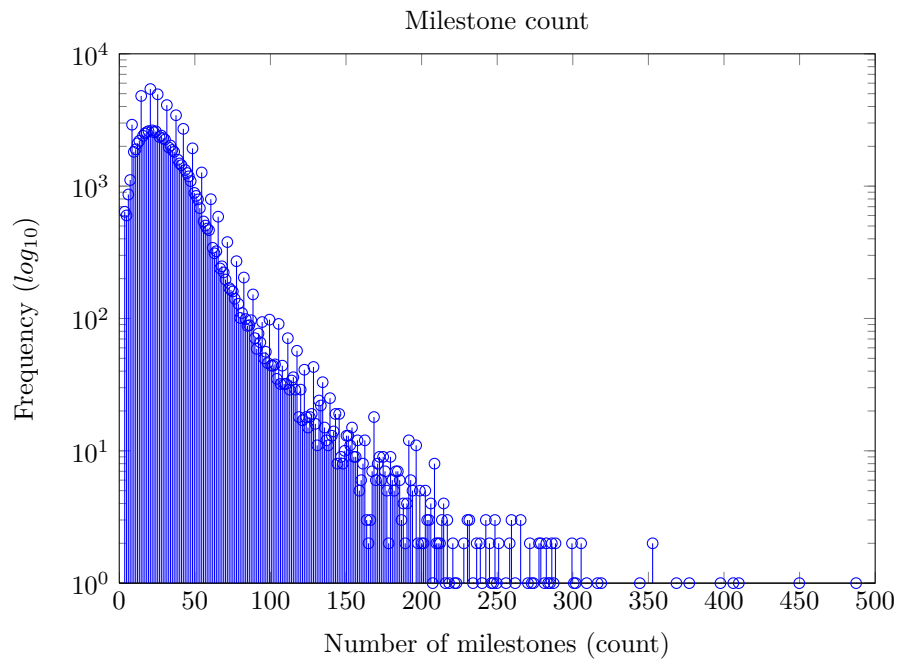
To obtain measurements of the iterations the RRT\* required the algorithm is run a 100000 times. Each run stops as soon as a path was found, and for each run the number of iterations the algorithm required is recorded. In Figure 5.16 a histogram is shown that presents the amount of milestones the RRT\* required on its x-axis. On the y-axis the frequency of the number of iterations is shown.

In Section 5.3.1 the theoretical performance bound for the RRT\* is calculated at 7174 iterations to guarantee a 99.99% success rate for finding a path. For the theoretical calculation to hold, the practical measurements may require 1 in 10000 algorithm runs to require more than 7174 iterations; that is, for 100000 runs this may occur no more than 10 times. The practical measurements show that the RRT\* required more than 7174 algorithm iterations exactly 0 times out of 100000, which means that the theoretical performance bound holds.

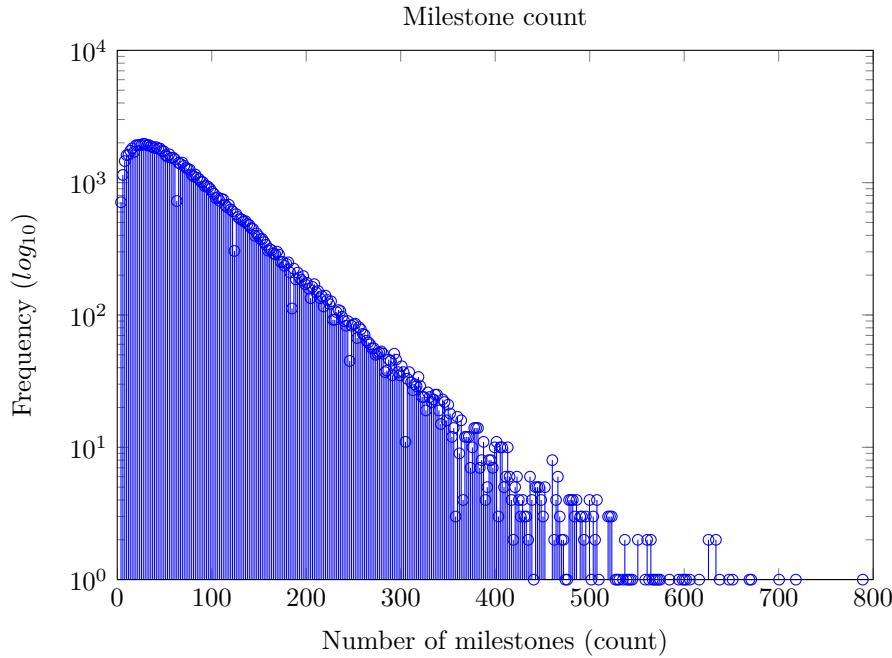
#### 5.4.2 Milestone count

As mentioned in the previous subsection, the PRM provides us with a theoretical bound on the number of milestones it needs to find a path whereas the RRT\*'s theoretical bound is with iterations.

Figure 5.17 shows us a histogram of the amount of milestones the PRM algorithm requires to find a path in the above environment, as does Figure 5.18 for the RRT\*. From these figures, it is possible to see that the PRM algorithm required less milestones than the RRT\* algorithm. After considering both algorithms, the most obvious reason for the much higher number of milestones required by the RRT\* algorithm is because the length of a manoeuvre is limited by  $\eta$ , whereas the PRM algorithm's manoeuvres have no such restriction. The average number of milestones required by the PRM and RRT\* algorithms are 32 and 78, respectively.



**Figure 5.17** – Histogram plot of number of milestones required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs. The measurements showed that 0 time out of 100000 runs the algorithm required more than 1328 milestones.



**Figure 5.18** – Histogram plot of number of milestones required by the RRT\* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs.

#### 5.4.2.1 Theoretical versus Practical milestone requirement

In Section 5.2.2, a theoretical performance bound is determined on the number of milestones the PRM requires to find a path in a specific environment. In this subsection, the theoretical bound is compared to practical measurements of the amount of milestones the PRM required to find a path. The same is done for the RRT\*; however its performance bound is on the number of iterations and its comparison of theoretical to practical measurements is done in Subsection 5.4.1.

For the practical measurements the PRM algorithm is run 100000 times. For each run the amount of milestones it required in its tree is recorded and as soon as a path is found the algorithm is stopped. In Figure 5.17 a histogram is shown that presents the number of milestones the PRM required to find a path on the x-axis. On the y-axis a frequency count of the number of milestones is displayed.

For the theoretical results to hold it is required that the histogram shows that no more than 99.99% of the 100000 runs required more than  $1328^{18}$  milestones. That is, if the PRM uses more than 1328 milestones more than 10 times out of a 100000 runs, the theoretical result is flawed. It is clear from the histogram that the PRM required more than 1328 milestones not even once out of a 100000 runs. This means that the theoretical results holds, albeit somewhat conservatively.

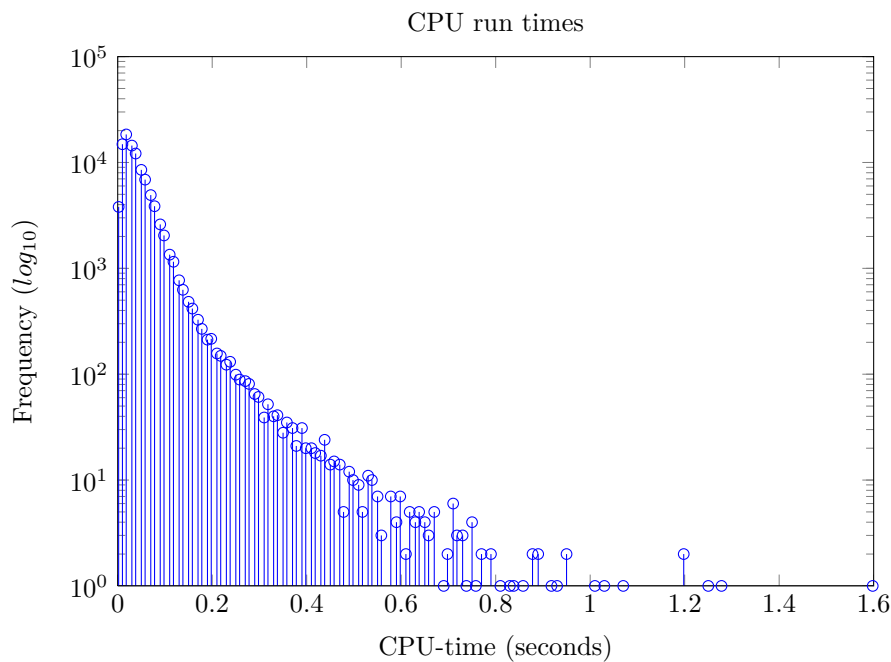
#### 5.4.3 CPU-time

In the previous subsections of this section, theoretical performance bounds of the PRM and RRT\* are compared to practical measurements. In Subsection 5.4.1 the iterations required

<sup>18</sup>This number was determined in Section 5.2.2.

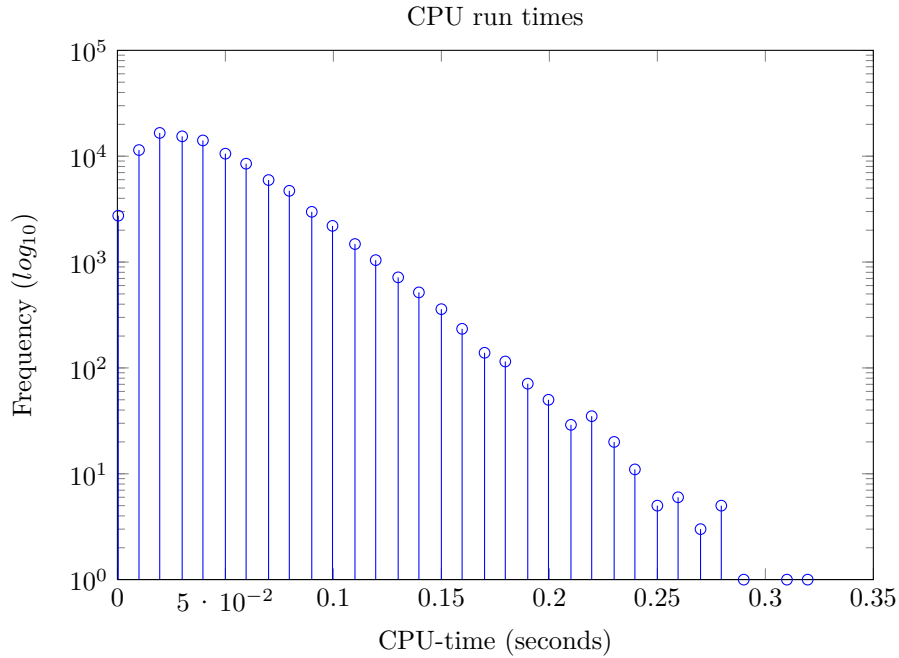
by the RRT\* was analysed and compared, and in Subsection 5.4.2 the milestones required by the PRM was analysed and compared. While these analyses provides us with valuable insight into the differences between the PRM and RRT\* it cannot provide a definitive answer to which algorithm performed the best.

Figure 5.19 shows a histogram of the CPU-time the PRM algorithm required to find a path in environment 8.1, as does Figure 5.20 for the RRT\*. The average CPU-time for the PRM algorithm is 0.0470 seconds, whereas the average CPU-time for the RRT\* algorithm is even less at 0.0285. It can be seen from the histograms that the RRT\* algorithm never took longer than 0.330 seconds to find a path; however, the PRM once took 1.598 seconds to find a path. This is because the RRT\* algorithm only tries to connect a new random milestone  $\mathbf{m}_{new}$  to its nearest neighbour  $\mathbf{m}_{nearest}$  whereas the PRM tries to connect  $\mathbf{m}_{new}$  to multiple milestones in its tree. It is seen from Figures 5.19 and 5.20 that this results in a much more consistent CPU-time required by the RRT\*. That is, the tail of the CPU-histogram of the RRT\* algorithm is almost non-existent relative to the tail of the PRM's CPU-histogram.



**Figure 5.19** – Histogram plot of CPU-time required by the PRM algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs.





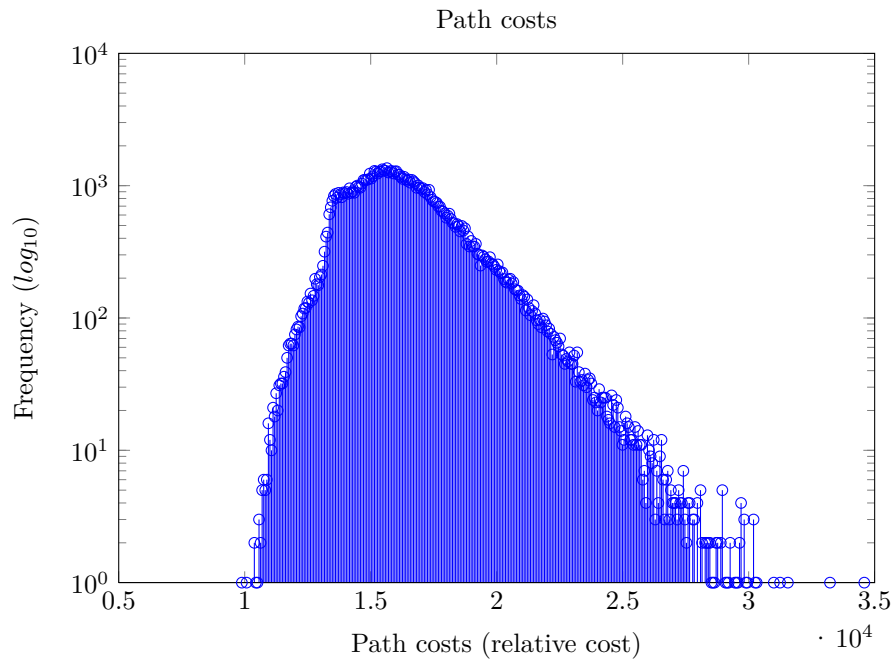
**Figure 5.20** – Histogram plot of cpu time required by the RRT\* algorithm to find a path in the environment shown in Figure 8.1(a) and 8.1(b) for 100000 runs.

#### 5.4.3.1 CPU time of theoretical bound

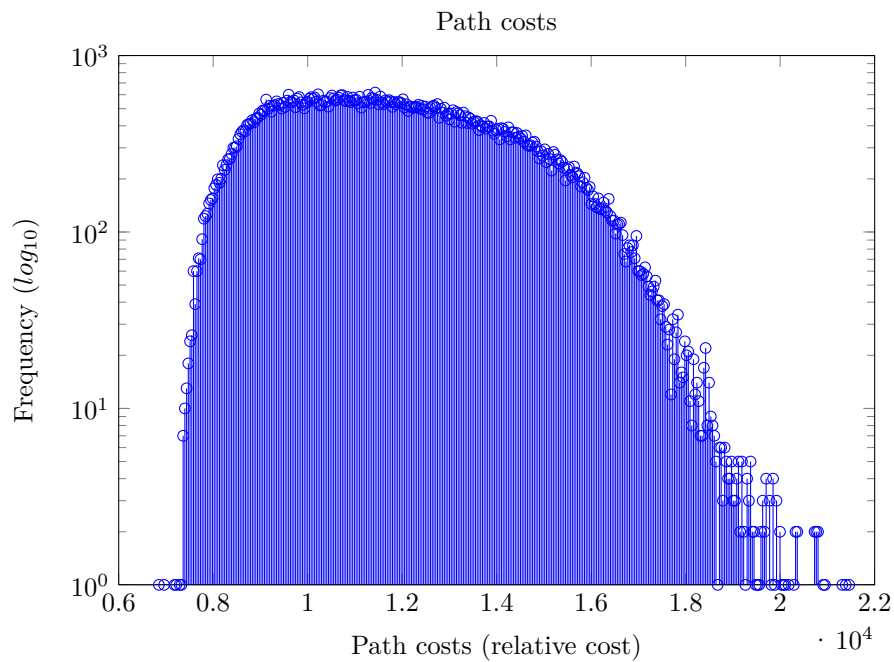
As noted in Subsection 5.4.1 and 5.4.2 the iteration and milestone performance bounds of the RRT\* and PRM algorithms can't be compared directly. It is necessary to first convert these bounds to something common to both algorithms. CPU time is common to both algorithms, and it is possible to convert the performance bounds to CPU time. For the PRM it is proposed to repetitively record the CPU time it takes to add 1328 milestones into its tree; for enough repetitions this will determine a guaranteed maximum time it requires to find a path. Similarly for the RRT\* it is possible to determine this maximum time, except that the time it takes to iterate 7174 times is recorded.

#### 5.4.4 Path cost

Path cost is not really relevant to this discussion, as no theoretical bounds on it was determined. Nonetheless, a path cost comparison is of interest since the RRT\* performs a number of path operations that should result in lower path costs. The average path costs for the PRM and RRT\* algorithms are 16337 and 11877 units, and it is seen from the histogram plots that the path costs of the RRT\* are much more closely distributed as opposed to the PRM path costs that have a somewhat wider distribution.



**Figure 5.21** – Histogram plot of path costs for the PRM in the environment shown in Figures 8.1(a) and 8.1(b) for 100000 runs.



**Figure 5.22** – Histogram plot of path costs for the RRT\* in the environment shown in Figures 8.1(a) and 8.1(b) for 100000 runs.

## 5.5 Conclusion

In Section 5.2.2 the PRM motion planning algorithm is analysed by looking at theorems from its proof of probabilistic completeness. These theorems are rewritten using relative frequency which enables their computation on a computer, which in turn provides a theoretical performance bound for the PRM. Similarly the RRT\* is analysed in Section 5.3.1 wherein a theoretical performance bound is also determined.

In Subsections 5.4.1 and 5.4.2 the theoretical bounds of the PRM and RRT\* are compared to respective practical measurements and it is confirmed that the theoretical performance bounds hold for both the PRM and RRT\*. In Section 5.4.3 the maximum CPU-time the PRM and RRT\* algorithms require to guarantee finding a path is determined, and it is seen that the RRT\* algorithm requires less (maximum) time than the PRM algorithm, even though the average execution time of the PRM is slightly lower than the RRT\*. In Subsection 5.4.4 the differences between the quality of the paths the PRM and RRT\* create is presented and it is seen that the average path cost of the PRM is lower than the RRT\*; however, the PRM generates a few paths that are very expensive, and the RRT\*'s path costs are closely bound.

In this chapter it is seen that both the PRM and RRT\* algorithms have strengths and weaknesses. However, it is seen that the RRT\* has much more consistent behaviour, and therefore the RRT\* is used in the next chapter wherein path replanning is presented.

## Chapter 6

# Path Replanning

### 6.1 Introduction

This project forms part of a larger goal to achieve autonomous navigation of a vehicle. Specifically, this project looks at the motion planning aspect thereof, and in Chapter 3 it was concluded that the RRT\* and PRM motion planning algorithms are to be used in this project. In Section 3.4 different constraints of the PRM and RRT\* motion planning algorithms are presented, amongst others the conflict detection algorithm. Some issues of the conflict detection algorithm are discussed therein and in this chapter path replanning is introduced to address some of these issues:

1. The conflict detection algorithm assumes all information about the environment is known beforehand; that is, positions of obstacles in the future as well as obstacles not in view of the vehicle or far away are all known, without uncertainty. This is not possible for practical implementations as any predictions into future behaviour of the environment will have increasing uncertainty with time.
2. The motion planner (in its current state) does not use any additional information about the environment post initial planning and the planned path stays ‘as is’ during the entire path traversal.

Regarding the first issue, replanning allows the algorithm to reset environment uncertainties at regular intervals. This results in propagating uncertainty only during a replan interval as opposed to propagating uncertainty for the duration of executing the entire path. This benefit will not directly influence this project as a non probabilistic conflict detection algorithm is used; however, as mentioned in Subsection 3.4.3 this project can be adapted to use a probabilistic conflict detection module.

Regarding the second issue, replanning allows the motion planning algorithm to include new information about the environment up until each replan occurrence, thereby enabling the algorithm to adjust or change a path accordingly. This means that if the behaviour of the environment differs from what is initially assumed, a previously safe path might experience conflict. Replanning will enable the motion planning algorithm to plan a different conflict free path. In Section 6.3 other benefits of replanning are also presented.

This chapter starts with discussing implementation details in Section 6.2 and thereafter a simulation example is presented in Section 6.3.

## 6.2 Implementation

In Chapter 5 the PRM and RRT\* motion planning algorithms are analysed and it is concluded that the RRT\* has a much more consistent behaviour compared to the PRM; therefore the path replanning algorithms will be based on the RRT\* algorithm presented in Subsection 4.9.3. The steps involved in replanning are:

1. Plan an initial  $path_1$  - The RRT\* path planner is used to find a path as soon as possible. This path might be cost ineffective but the vehicle can start to execute the path.
2. While the vehicle is executing  $path_1$  the replanner determines where ( $p_{t=1}$ ) on the path the vehicle will be at the next replan interval. That is, if the replan interval is 1 second the algorithm determines where the vehicle will be on the path after 1 second of execution.
3. Position  $p_{t=1}$  is then passed to the RRT\* path planner algorithm as the initial milestone while the goal is kept the same, and the path planner algorithm may then search for 1 second for a better path,  $path_2$ . The milestones that form  $path_1$  are also passed to the algorithm.
4. If  $path_2$  has a better cost than  $path_1$  the vehicle will switch to executing  $path_2$  as soon as  $p_{t=1}$  is reached.
5. The last step is to update the environment to reflect any new information. If  $path_2$  now contains conflict, the milestones of the path are discarded and not passed to the path planner in the next iteration.
6. The process is repeated from where the replanner algorithm determines where the vehicle will be (for the next iteration  $p_{t=2}$ ) after another replan interval.

In the next section the above process is presented together with an example.

## 6.3 Simulation example

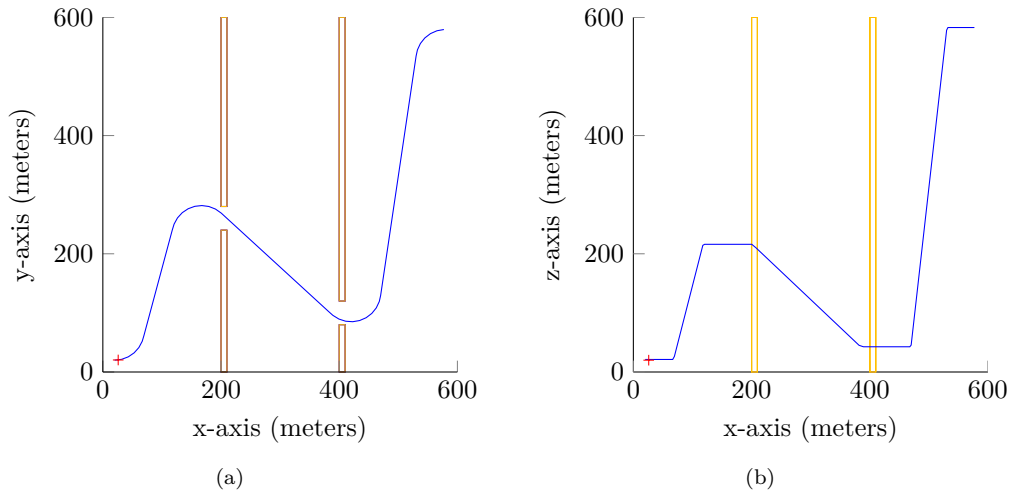
### 6.3.1 Initial path

The RRT\* motion planning algorithm is used to find a path between the goal and initial milestone as fast as possible. This path is shown in Figure 6.1 where the left hand figure shows a path from the top, the right hand shows a path from the side, and the red '+' shows the progress of the vehicle. As soon as this path is found the vehicle can start to execute it, this initial path is found in  $10\text{ms}^1$ . In Chapter 5 it was shown that the RRT\* algorithm has a 99.99% chance to find a path under 0.330 seconds, thereby providing an upper bound on the time the motion planner requires before the vehicle can start to execute a path.

The cost of the path is 9009 which is expensive compared to 6789, which is achieved by the end of this chapter through replanning. While the vehicle is executing part of this path the replanner determines where the vehicle will be after the replan period and uses this position as the initial position during the next replan interval. This position is given to the RRT\* algorithm as the initial milestone and now the algorithm plans for the duration of the replan interval. This path is shown in the next subsection.

---

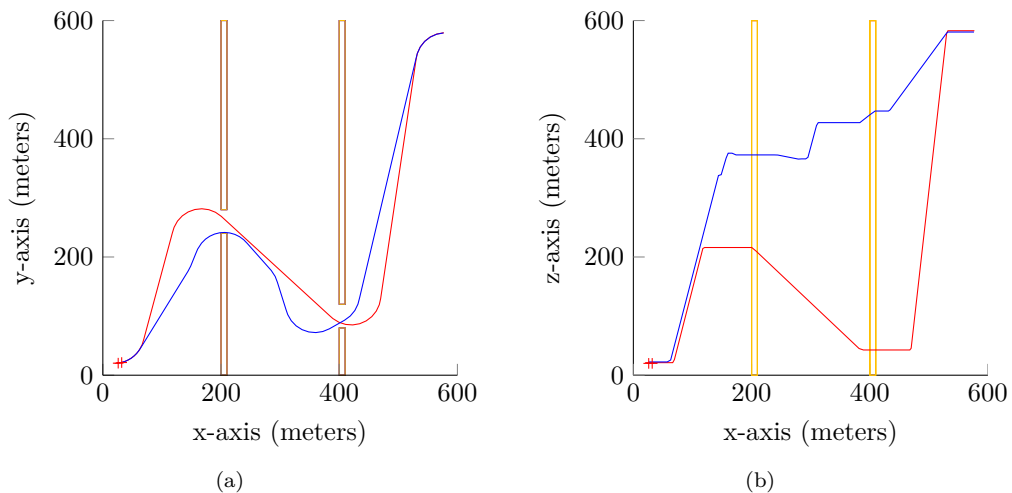
<sup>1</sup>Refer to Section 4.2 for details about the hardware and software used in the implementation.



**Figure 6.1** – A plot showing the top and side view plots of the first path the RRT\* path planner found and the **initial** path the replanner will execute.

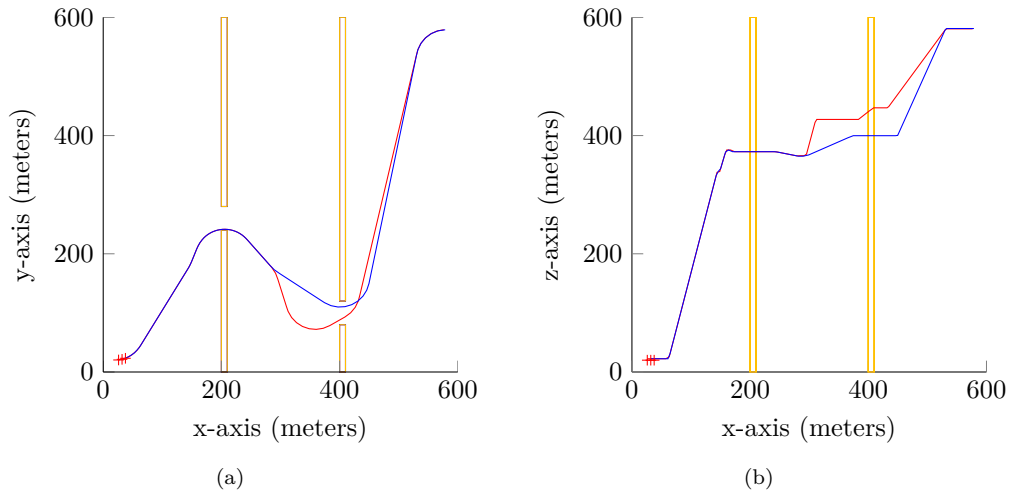
### 6.3.2 Improving the path

For the duration of the replan interval the existing path (shown in red) is improved by the RRT\* algorithm. The improved path (shown in blue) has a cost of 7887 and is shown below. The rest of the figures in this subsection shows how the path improves due to replanning.

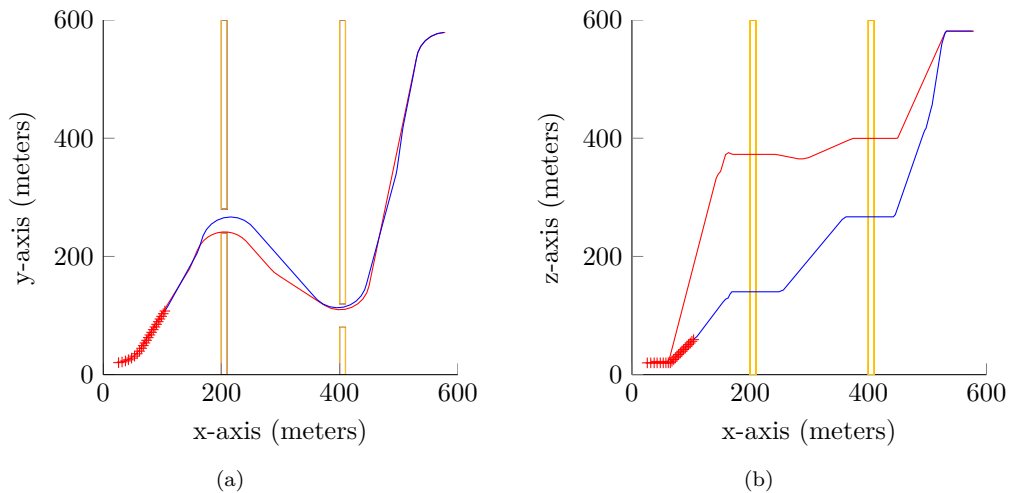


**Figure 6.2** – A plot showing the top and side view plots of an **improved** path (cost: 7887) the RRT\* path planner found in blue and the old path in red.

These figures and their respective path costs show that replanning enables a vehicle to start executing a path very quickly while still allowing the vehicle to traverse a path that is continually improved. In this subsection, the information the conflict detection algorithm had did not change: however, as a vehicle explores an environment, its sensors might become more confident of where objects are. This might lead to observing that obstacle positions



**Figure 6.3** – A plot showing the top and side view plots of an **improved** path (cost: 7556) the RRT\* path planner found in blue and the old path in red.



**Figure 6.4** – A plot showing the top and side view plots of an **improved** path (cost: 7227) the RRT\* path planner found in blue and the old path in red.

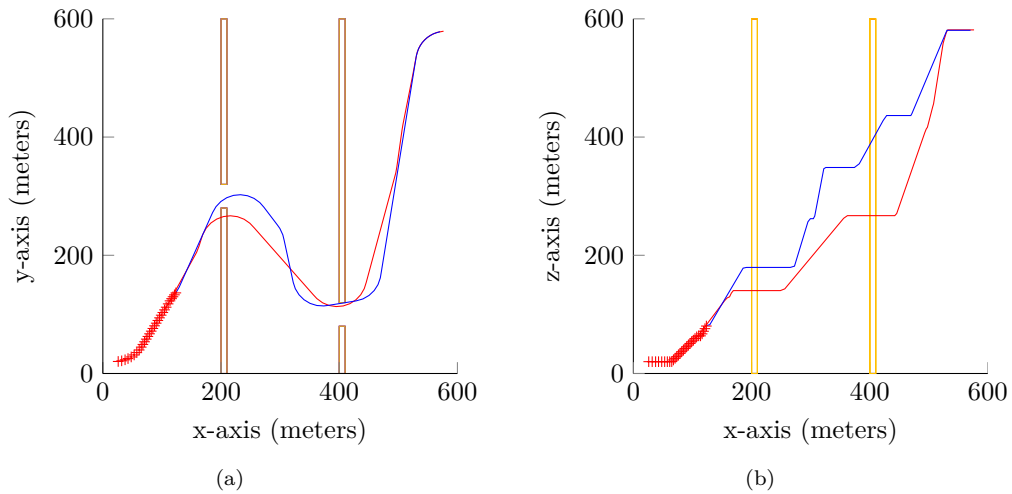
that differ from their initial observed positions<sup>2</sup>, and paths that were previously thought of as conflict free are in fact not. In the next subsection a change in the environment is observed that causes the path to have conflict.

### 6.3.3 Path with new environment information

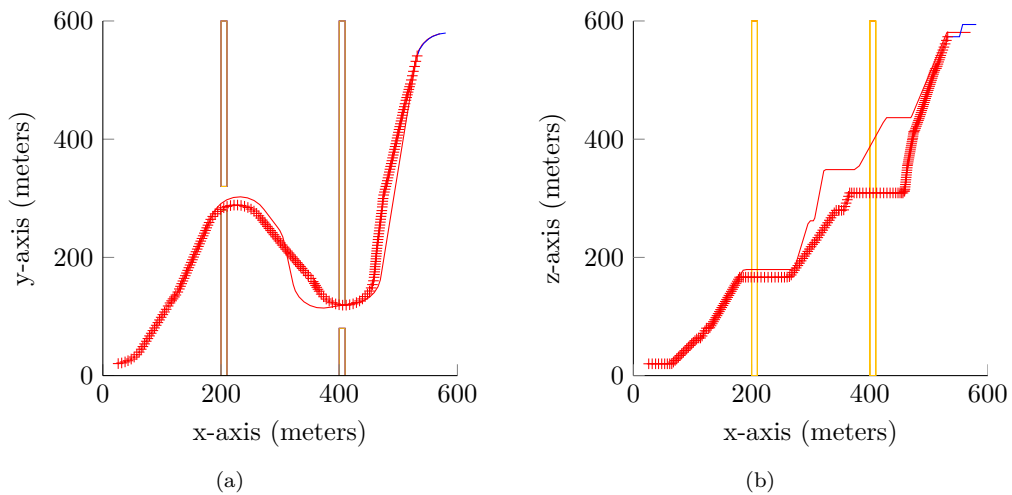
In this subsection, the environment of the previous subsection has slightly changed, which causes the path to have conflict. From Figure 6.5(a) onwards, the gap in the wall at  $x = 200\text{m}$  has moved further away from the vehicle. Fortunately, replanning allows the conflict

<sup>2</sup>This also applies to dynamic obstacles not behaving as predicted.

detection algorithm to update its information about the environment, which enables the motion planning algorithm to search for a conflict free path.



**Figure 6.5** – A plot showing the top and side view plots of the path the RRT\* path planner found with the **updated** environment information.



**Figure 6.6** – A plot showing the top and side view plots of the path (cost: 6789) the vehicle traversed.

Figure 6.6 shows the path the vehicle traversed in red '+'s. The traversed path cost is 6789 units, which is a major improvement over the cost (9009) of the initial path.

## 6.4 Conclusion

In this chapter the path replanning algorithm is introduced as a modification of the RRT\* motion planning algorithm. The implementation difference between the RRT\* and RRT\* with replanning is discussed and it is seen that the algorithms are very similar. The only



differences are to the path planner Algorithm 8, the rest of the RRT\* algorithm as discussed in Section 4.9 stays the same.

The key difference is that the path replanner doesn't stop planning, until the vehicle reaches the goal. At each replan interval, the replanner moves the initial milestone to a predicted future position, and tries to improve upon the current path to the goal.

Next, an example is presented wherein the vehicle could start executing the path after only 10ms. The initial path's cost of 9009 units is much more than 6789, as was achieved through continual use of path replanning. Thus the path replanner enables this project's vehicle to quickly start traversing a path to a goal while still providing it with a cost effective path.

Next, new information about the environment is introduced and the environment changes in such a way that the path the vehicle is traversing has conflict. The path replanner allows this new information to be used by the conflict detection module, thereby allowing the motion planning algorithm to create a new path that is free of conflict. Thus, the path replanning algorithm enables the motion planning algorithm to take new information into account when planning that might not have been available when initial planning begun.

## Chapter 7

# Software simulation

### 7.1 Introduction

This project forms part of a larger goal to achieve autonomous flight of an unmanned rotary wing aircraft, and in Chapter 1 the necessary building blocks are outlined. This project's focus is on the motion planning module, and in Chapter 3 the problem statement for this project is presented. A requirement of the problem statement is that the planned path must satisfy nonholonomic and kinodynamic constraints of a chosen vehicle, i.e. the path must be executable by the vehicle.

The motion planning algorithms used in this project use manoeuvres to adhere to the non-holonomic and kinodynamic constraints of a vehicle during planning. Since manoeuvres are used to construct the path, the planned path is assumed to be executable by the vehicle.

In this chapter, this assumption is tested by executing a planned path using vehicle controllers that have been used in flight tests. Unfortunately, all the necessary building blocks to allow for real world testing of this project are not finished. However, a Simulink simulation that uses software implementations of flight tested controllers is available. The purpose of this chapter is to use the Simulink simulation to determine whether the manoeuvre library used in this project allows the vehicle to remain within its envelope of operation.

In the first section of this chapter (Section 7.2) the Simulink model of the vehicle is discussed, thereafter the necessary changes to the Simulink model to enable executing a planned path are presented.

### 7.2 Simulink model

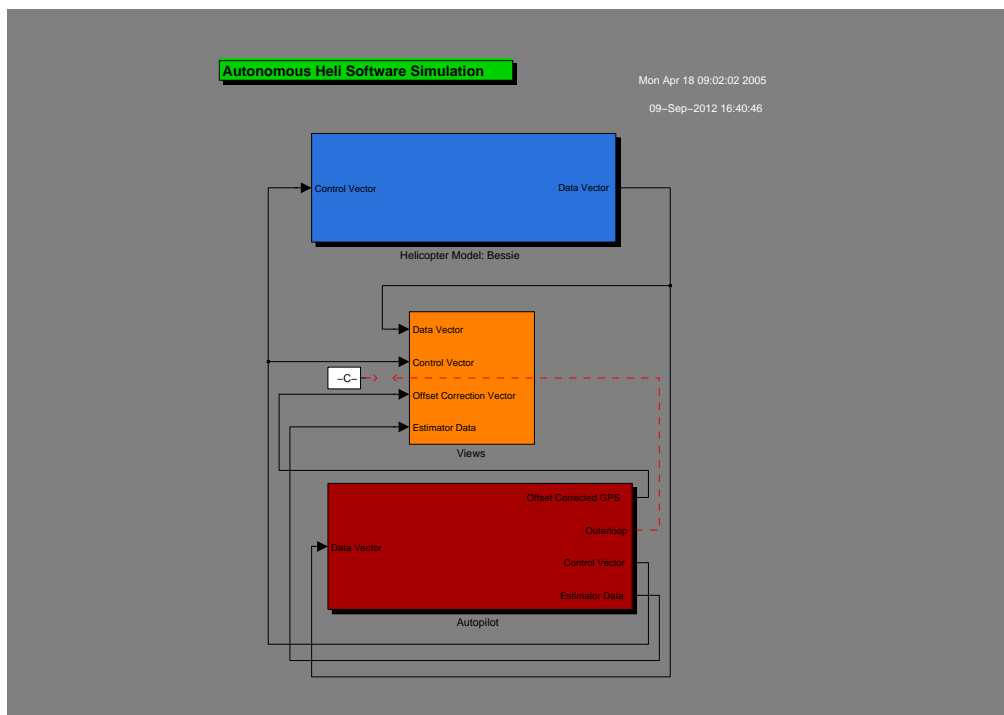
The Simulink model consists purely of software components; that is, no flight related hardware is included in the simulation. Using such a simulation is beneficial for this project for various reasons:

- Data from the simulation can be accessed quickly and analysed through the MATLAB/Simulink environment.
- Since no hardware is included in the simulation it can be performed faster than real time, which allows for a very short turnaround time for modifications.
- Getting started with the Simulink environment is very quick as it is not necessary to build any custom hardware modules.

- Some of the building blocks<sup>1</sup> required for autonomous navigation are not completed or are unavailable in hardware form.

In previous projects by Groenewald [1] and Rossouw [7], components of the Simulink simulation were developed in either block diagram form or MATLAB specific M-code. Figure 7.1 shows the Simulink simulation at a high level. The blocks shown present three main aspects of the simulation:

1. The **helicopter model** wherein a non-linear mathematical model resides that imitates the behaviour of the helicopter. A discussion on this model is presented in Subsection 2.2.3.
2. The **Views** block is responsible for handling the data of the simulation, as well as logging selected data entries.
3. The **autopilot** block was developed partially by Groenewald and Rossouw, and in Figure 7.2 an overview of the autopilot system is shown. The autopilot block contains the controllers of the system and in Figure 7.3 the inner loop controllers are shown. The controllers used here was previously developed and only an overview of them are presented in Subsection 2.2.4.

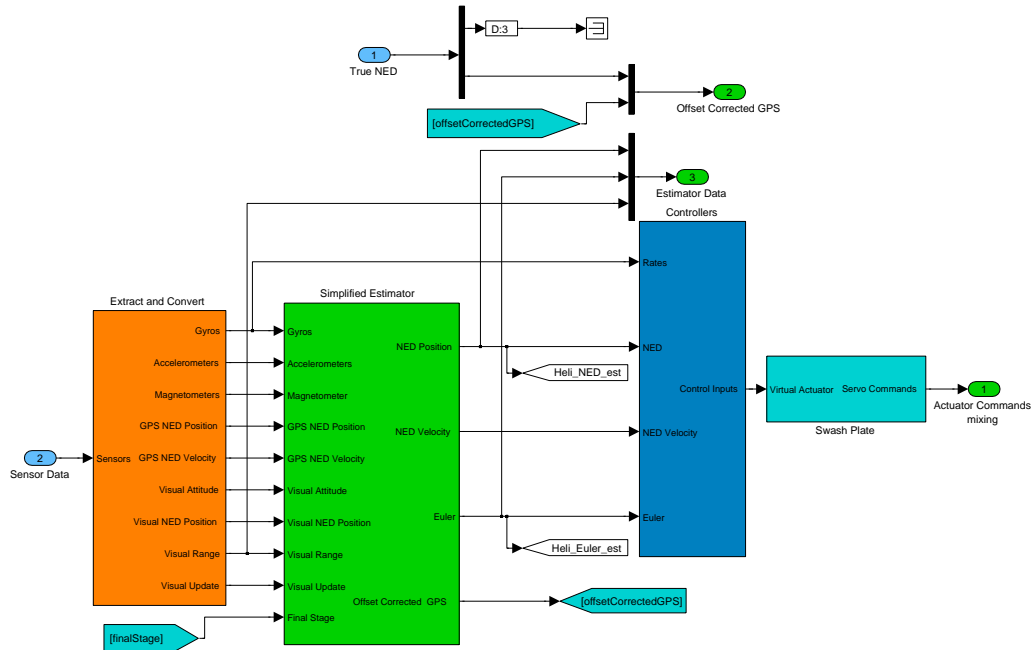


**Figure 7.1** – This figure shows the high level Simulink simulation. The simulation consists of three parts, the helicopter model (the block shown in blue), the autopilot (the block shown in red) and the data viewer or logger (the block shown in orange).

Note that the outer loop is already disconnected in this figure as it is not used in this project. The helicopter model block (shown in blue) contains the model of the vehicle as discussed

<sup>1</sup>The reader is referred to Chapter 1.

in Subsection 2.2.3, and the autopilot block (shown in red) contains the controllers for the vehicle, as discussed in Subsection 2.2.4.



**Figure 7.2** – This figure shows an overview of the Autopilot in the Simulink simulation.

Figure 7.2 shows the inside of the autopilot block of Figure 7.1. The controller block (shown in blue) is of particular interest, as it contains the controllers used later in this chapter. The controller block receives input from the estimator<sup>2</sup>, and provides control inputs to the swashplate<sup>3</sup> (shown in aqua blue). Inside the controller block, the Rates, NED, NED Velocity, and Euler inputs to the controller block is passed to the inner loop controllers of Figure 7.3.

Figure 7.3 shows the inner loop controllers: that is, the heave, longitudinal, lateral, and heading controllers<sup>4</sup>. From the figure it is seen that the inner loop controllers can receive the following references as input: NED Reference, Heading Reference, and Velocity Reference, as shown in Figure 7.4. It is important to note that Figure 7.4 shows how the inputs of the control block of Figure 7.2 are passed to the inputs of Figure 7.3.

<sup>2</sup>Discussed in Subsection 2.2.5.

<sup>3</sup>Control inputs and actuators are discussed in Subsection 2.2.2.

<sup>4</sup>These controllers are discussed in Subsection 2.2.4.

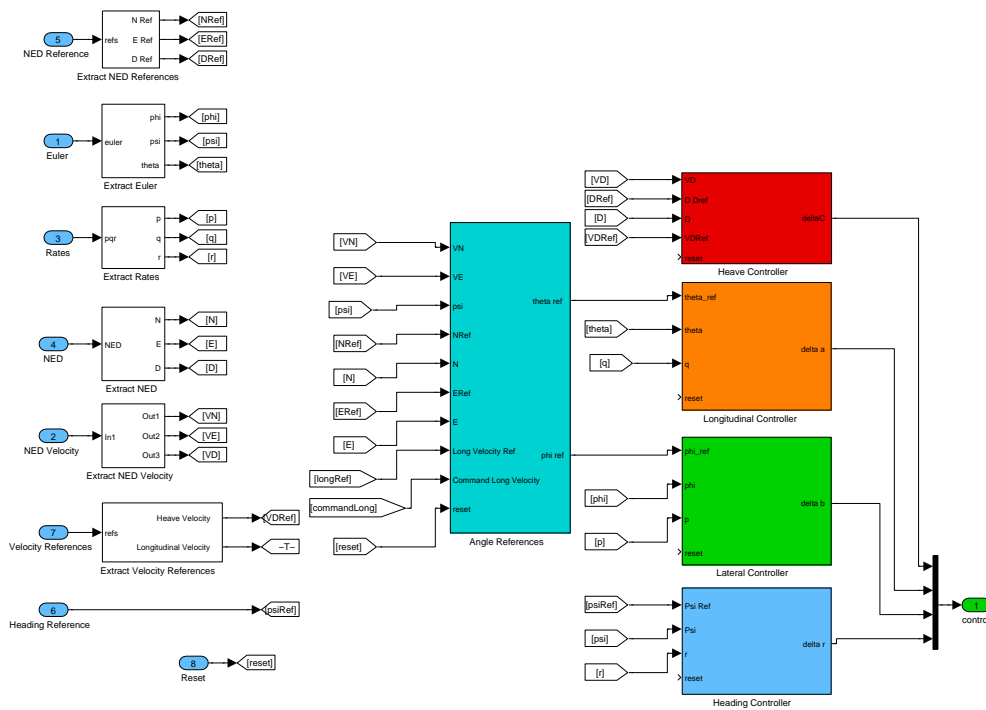


Figure 7.3 – This figure shows the inner loop controllers.

For this project, a design goal is to make as little as possible change to the system, while adding the capability to follow a path generated by the motion planner. For this reason it was decided to add a S-Function block (the path planner block shown in Figure 7.4) on a level between the autopilot system and the inner loop controllers. This allows the path planner block to provide position, heading and velocity references to the inner loop controllers, and in Subsection 7.3.1 it is found that this is enough to enable path execution with acceptable error bounds. In the next subsection, the controllers within the path planner block are presented.

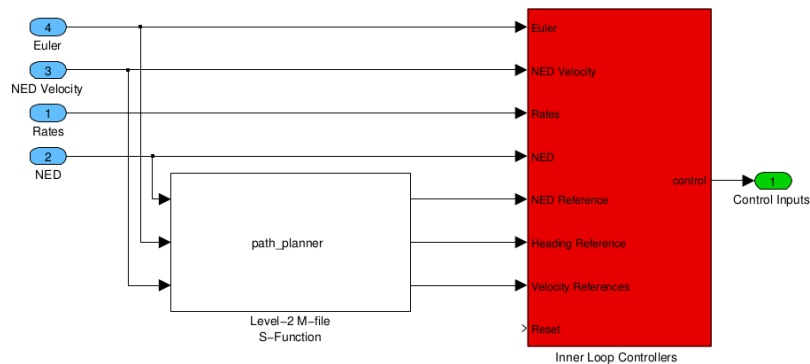


Figure 7.4 – This figure shows the path planner block and its interaction with the inner loop controllers.

### 7.2.1 Path planner block

The main goal of the Simulink simulation is to determine whether the manoeuvre library used in this project allows the vehicle to remain within its envelope of operation. Therefore it is sufficient to plan a path offline, give the path to the path planner block, and let the path planner block provide references to the inner loop controllers based on the given path. That is, the path planner block in Figure 7.4 is responsible for providing position, heading and velocity references to the inner loop controllers so that a path planned by a motion planning algorithm may be executed.

#### 7.2.1.1 Necessary path information

The path generated by the motion planning algorithm is given to the path planner block as a list of manoeuvres and milestones. For an example, the manoeuvres and milestones of the path shown in Figure 7.5 are given to the path planner block in MATLAB code. All the MATLAB code necessary to follow the path is shown in Appendix A. Note that the presented list of milestones and manoeuvres (in the appendix) is defined starting at the goal. Nonetheless, the first entry in the actual list is the first milestone from the initial milestone, that is, the list is constructed backwards.

A segment of the necessary MATLAB code is presented below:

```
list = [];
manoeuvre1=struct('m', 6, 'x', 430.0230, 'y', 146.2181, 'z', 237.2146,
't', 589.6619, 'xm', 381.6167, 'ym', 158.7411);
manoeuvre2=struct('m', 8, 'x', 531.5585, 'y', 542.3869, 'z', 580.5756,
't', 1034.5687);
manoeuvre3=struct('m', 4, 'x', 580.0000, 'y', 580.0000, 'z', 580.0000,
't', 1089.5647, 'xm', 579.9931, 'ym', 529.9734);
milestone1=struct('x', 580.0000, 'y', 580.0000, 'z', 580.0000, 'manoeuvre1',
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);
list = [milestone1 list];
```

Where `list` is the list that contains all the necessary information to reach a milestone from a parent milestone, and `struct` creates a MATLAB structure with members. The members of a manoeuvre are:

- `m` denotes the manoeuvre to execute, either right turn (6), left turn (4) or forward (8) as defined in Section 4.5. The `-1` indicates a 'no manoeuvre' which is used when this manoeuvre is generated by the Steer (Subsection 4.9.1) algorithm which consists of only two manoeuvres.
- `x`, `y`, `z`, and `t` denotes the `x,y` and `z` coordinates of the end of a manoeuvre as well as the time by which this point must be reached.
- `xm`, and `ym` denotes the `x` and `y` coordinates of the midpoint of the turning circle.

The members of a milestone are:

- `x`, `y`, and `z` which denotes the `x,y` and `z` coordinates of the milestone.
- `manoeuvre1`, `manoeuvre2`, and `manoeuvre3` denotes the manoeuvres defined above.

### 7.2.1.2 Manoeuvre controllers

Next, the manner by which the path planner processes the above defined information is discussed. The information is in a list of milestones, where each milestone contains the information necessary to reach it from its parent milestone. The path planner block executes the entire path by simply processing the list one milestone at a time.

To reach a milestone, the path planner block executes a milestone's associated manoeuvres, and for this purpose manoeuvre specific controllers are used. The references the path planner block provides to the inner loop controllers are:

- A position vector consisting of a x, y and z coordinate  $[x_{\text{ref}}, y_{\text{ref}}, z_{\text{ref}}]$ .
- A velocity vector consisting of a forward and heave velocity  $[v_{xy\text{ref}}, v_{z\text{ref}}]$ .
- A heading reference  $\psi$ .

**Left turn manoeuvre** The first step is to calculate the heading that the vehicle should have:

$$\psi_{\text{curr}} = \text{atan2}(y_{\text{curr}} - y_{\text{mid}}, x_{\text{curr}} - x_{\text{mid}}) + \pi/2 \quad (7.2.1)$$

Where  $\psi_{\text{curr}}$  denotes the current heading of the vehicle,  $x_{\text{curr}}$  and  $y_{\text{curr}}$  denotes the current x and y position of the vehicle, and  $x_{\text{mid}}$  and  $y_{\text{mid}}$  are  $\mathbf{x}_m$  and  $\mathbf{y}_m$  as defined earlier. Next the heading at the end of the turn is determined:

$$\psi_{\text{end}} = \text{atan2}(y_{\text{stop}} - y_{\text{mid}}, x_{\text{stop}} - x_{\text{mid}}) \quad (7.2.2)$$

Where  $\psi_{\text{end}}$  is the heading of the vehicle at the end of the manoeuvre, and  $x_{\text{stop}}$  and  $y_{\text{stop}}$  are the coordinates of the end of the manoeuvre. Next the distance is determined between where the vehicle is at the moment and where the manoeuvre stops.

$$\text{distance} = \text{abs}(\psi_{\text{end}} - \psi_{\text{curr}}) \times r \quad (7.2.3)$$

Where  $\text{abs}()$  denotes the absolute value, and  $r$  is a constant representing the turning radius used when the path was planned by the motion planning algorithm. This distance is then used to calculate the velocity at which the vehicle must travel in order to reach  $x_{\text{stop}}, y_{\text{stop}}$  at the specified time  $\mathbf{t}$ .

$$\text{time}_{\text{left}} = \text{time}_{\text{end}} - \text{time}_{\text{elapsed}}, \quad (7.2.4)$$

$$v_{\text{forward}} = \text{distance} / \text{time}_{\text{left}}, \quad (7.2.5)$$

where  $\text{time}_{\text{end}}$  is the time the vehicle must reach the end of the manoeuvre, and  $\text{time}_{\text{elapsed}}$  is the elapsed time since the start of the simulation.

Next, proportional control is used to determine the reference velocity and position the path planner block provides the inner loop controllers. The heading reference is calculated somewhat differently.

$$v_{\text{err}} = v_{\text{forward}} - v_{\text{curr}}, \quad (7.2.6)$$

$$v_{xy\text{ref}} = v_{\text{forward}} \times P_v, \quad (7.2.7)$$

where  $v_{\text{err}}$  denotes the difference between the current velocity ( $v_{\text{curr}}$ ) of the vehicle and the velocity the vehicle should be travelling,  $P_v$  is the proportional gain, and  $v_{\text{ref}}$  is the reference velocity in the forward direction. The heading reference calculation simply adds a small fraction of a circle to the current heading:

$$\psi_{\text{ref}} = \psi_{\text{curr}} + P_\psi / (2\pi r), \quad (7.2.8)$$

where  $\psi_{\text{ref}}$  is the heading reference, and  $P_\psi$  is the small fraction of a circle added to the current heading. Lastly, the position reference is determined through proportional control:

$$x_{\text{est}} = r \times \cos(\psi_{\text{curr}}), \quad (7.2.9)$$

$$x_{\text{err}} = x_{\text{est}} - x_{\text{curr}}, \quad (7.2.10)$$

$$x_{\text{ref}} = r \times \cos(\psi_{\text{ref}}) + x_{\text{mid}} + x_{\text{err}} \times P_x, \quad (7.2.11)$$

$$y_{\text{est}} = r \times \sin(\psi_{\text{curr}}), \quad (7.2.12)$$

$$y_{\text{err}} = y_{\text{est}} - y_{\text{curr}}, \quad (7.2.13)$$

$$y_{\text{ref}} = r \times \sin(\psi_{\text{ref}}) + y_{\text{mid}} + y_{\text{err}} \times P_y, \quad (7.2.14)$$

where  $x_{\text{est}}$  and  $y_{\text{est}}$  are the estimated x and y positions where the vehicle should be based on its current heading,  $x_{\text{err}}$  and  $y_{\text{err}}$  are differences between where the vehicle should be and where it currently is,  $P_x$  and  $P_y$  are the proportional gain used for position, and  $x_{\text{ref}}$  and  $y_{\text{ref}}$  are the references the path planner block provides the inner loop controller.

As the vehicle does not translate in the z dimension (that is, it does not move up or down) during a left turn manoeuvre the position reference for the z dimension is simply its current z position.

The controller for executing a **Right turn manoeuvre** operates in a very similar manner to the controller for executing a Left turn manoeuvre.

**Forward manoeuvre** The first step in this controller is to determine where the vehicle should be at the current time:

$$t_{\text{diff}} = t_{\text{stop}} - t_{\text{prev}}, \quad (7.2.15)$$

$$t = (t_{\text{elapsed}} - t_{\text{stop}}) / t_{\text{diff}}, \quad (7.2.16)$$

$$x_{\text{est}} = x_{\text{stop}} + t \times (x_{\text{stop}} - x_{\text{prev}}), \quad (7.2.17)$$

$$y_{\text{est}} = y_{\text{stop}} + t \times (y_{\text{stop}} - y_{\text{prev}}), \quad (7.2.18)$$

$$z_{\text{est}} = -(z_{\text{stop}} + t \times (z_{\text{stop}} - z_{\text{prev}})), \quad (7.2.19)$$

where  $t_{\text{diff}}$  is the difference between the time associated with the end of this manoeuvre ( $t_{\text{stop}}$ ) and the time associated with the end of the previous manoeuvre ( $t_{\text{prev}}$ ). The variable  $t$  is a measurement of the amount of time already elapsed for the execution of the current manoeuvre, and  $t_{\text{elapsed}}$  is the elapsed time of the entire simulation. The variables  $x_{\text{est}}$ ,  $y_{\text{est}}$ , and  $z_{\text{est}}$  are the estimated x, y, and z positions where the vehicle should be according to the  $t_{\text{elapsed}}$ . The variables  $x_{\text{stop}}$ ,  $y_{\text{stop}}$ , and  $z_{\text{stop}}$  are the x, y, and z positions at the end of the manoeuvre. The variables  $x_{\text{prev}}$ ,  $y_{\text{prev}}$ , and  $z_{\text{prev}}$  are the x, y, and z positions at the end of the previous manoeuvre.

The next step is to calculate the heave angle:

$$\theta = \text{atan2}(z_{\text{stop}} - z_{\text{prev}}, \text{distance}_{xy}), \quad (7.2.20)$$



where  $\theta$  is the heave angle, and  $\text{distance}_{xy}$  is the total distance of the manoeuvre in the x, y plane. The heave angle is then used to determine the reference output for the z dimension:

$$z_{\text{step}} = \text{xyz}_{\text{step}} \times \sin(\theta), \quad (7.2.21)$$

$$z_{\text{err}} = z_{\text{est}} - z_{\text{curr}}, \quad (7.2.22)$$

$$z_{\text{ref}} = z_{\text{est}} - z_{\text{step}} + P_z \times z_{\text{err}}, \quad (7.2.23)$$

where  $z_{\text{step}}$  is a portion of the  $\text{xyz}_{\text{step}}$  which is used to determine the vehicle's position reference,  $z_{\text{err}}$  is the difference between where the vehicle should be and it currently is ( $z_{\text{curr}}$ ), and  $P_z$  is a proportional gain.

Next, the x and y position and heading references are calculated:

$$\psi = \text{atan2}(y_{\text{stop}} - y_{\text{prev}}, x_{\text{stop}} - x_{\text{prev}}), \quad (7.2.24)$$

$$\text{xy}_{\text{step}} = \text{xyz}_{\text{step}} \times \cos(\theta), \quad (7.2.25)$$

$$x_{\text{err}} = x_{\text{est}} - x_{\text{curr}}, \quad (7.2.26)$$

$$x_{\text{ref}} = x_{\text{est}} + \text{xy}_{\text{step}} \times \cos(\psi) + P_x \times x_{\text{err}}, \quad (7.2.27)$$

$$y_{\text{err}} = y_{\text{est}} - y_{\text{curr}}, \quad (7.2.28)$$

$$y_{\text{ref}} = y_{\text{est}} + y_{\text{step}} \times \cos(\psi) + P_y \times y_{\text{err}}, \quad (7.2.29)$$

where  $\psi$  is the heading angle between the end of the previous manoeuvre and the end of the current manoeuvre,  $\text{xy}_{\text{step}}$  is a portion of the  $\text{xyz}_{\text{step}}$  which is used to determine the vehicle's position reference,  $x_{\text{err}}$  and  $y_{\text{err}}$  is the difference between where the vehicle currently is and where it should be, and  $P_x$  and  $P_y$  are proportional gains. The  $\psi$  calculated here is also used as the heading reference.

Lastly, the references for the forward and heave velocity are calculated:

$$v_{\text{req}} = \text{distance}_{\text{rem}} / \text{time}_{\text{rem}}, \quad (7.2.30)$$

$$v_{\text{err}} = v_{\text{req}} - v_{\text{curr}}, \quad (7.2.31)$$

$$v_{\text{ref}} = v_{\text{req}} + P_v \times v_{\text{err}}, \quad (7.2.32)$$

$$v_{xy\text{ref}} = v_{\text{ref}} \times \cos(\theta), \quad (7.2.33)$$

$$v_{z\text{ref}} = v_{\text{ref}} \times \sin(\theta), \quad (7.2.34)$$

$$(7.2.35)$$

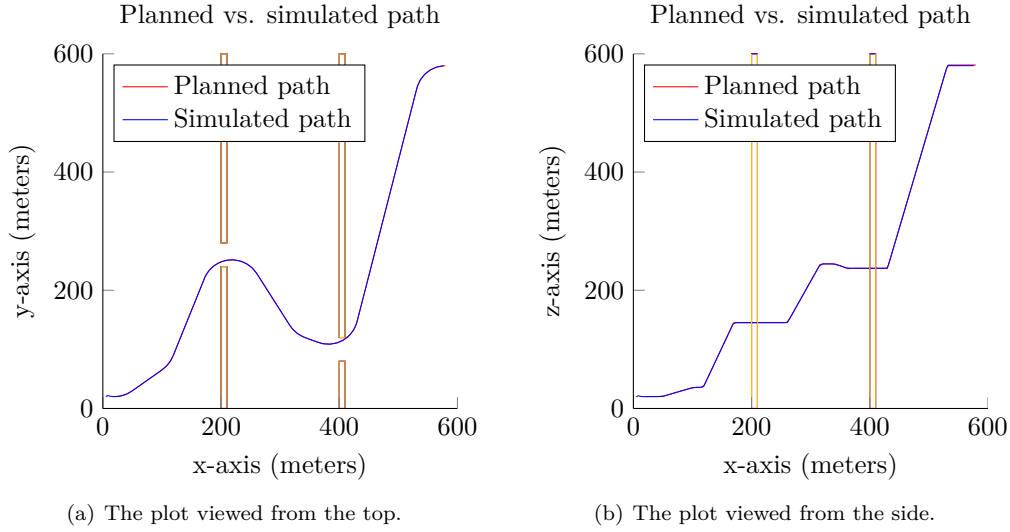
where  $v_{\text{req}}$  is the velocity required to reach the end of the manoeuvre on time,  $\text{distance}_{\text{rem}}$  and  $\text{time}_{\text{rem}}$  is the distance and time remaining from their current positions to the end of the manoeuvre,  $v_{\text{err}}$  is the difference between the required velocity ( $v_{\text{req}}$ ) and the current velocity ( $v_{\text{curr}}$ ) of the vehicle,  $v_{\text{ref}}$  is the total reference velocity,  $P_v$  is the proportional gain used for velocity,  $v_{xy\text{ref}}$  is the velocity reference in the x, y plane, and  $v_{z\text{ref}}$  is the velocity reference in the z dimension.

## 7.3 Results

### 7.3.1 Path following

The purpose of this chapter is to ascertain whether the manoeuvres used in this project enables a vehicle to remain within its envelope of operation while executing a planned path. In the previous section the Simulink model and the controllers of the vehicle are discussed as well as the necessary changes to enable execution of this project's manoeuvres. In this section results are presented that show the vehicle is able to execute a planned path within acceptable cross and along-track errors.

In Figure 7.5, the planned path is superimposed on a trace of the vehicle executing the planned path. The error between the planned path and the trace is not visible from these figures, however in Tables 7.1 and 7.2 the along- and cross-track errors are shown.



**Figure 7.5** – This figure shows a planned path superimposed on the simulated path of the vehicle.

Note that a path is represented by an array of xyz-coordinates, and each xyz-coordinate also has a time associated with it. To illustrate the along-track error, take for example xyz-coordinates  $(x_{\text{planned}}, y_{\text{planned}}, z_{\text{planned}})$  and  $(x_{\text{actual}}, y_{\text{actual}}, z_{\text{actual}})$ , both associated with time  $t_{\text{now}}$ . The coordinate  $(x_{\text{planned}}, y_{\text{planned}}, z_{\text{planned}})$  denotes where the vehicle should be at  $t_{\text{now}}$ , however the vehicle is actually located at the coordinate  $(x_{\text{actual}}, y_{\text{actual}}, z_{\text{actual}})$ . The along-track error is calculated as:

$$\text{along-track}_{\text{error}} = \sqrt{(x_{\text{planned}} - x_{\text{actual}})^2 + (y_{\text{planned}} - y_{\text{actual}})^2 + (z_{\text{planned}} - z_{\text{actual}})^2}. \quad (7.3.1)$$

Table 7.1 shows the along-track error. These errors are maximum values obtained during executing the path shown in Figure 7.5 and show that the error never exceeded 1 meter. Considering the size of the vehicle<sup>5</sup>, this error is acceptable.

| Dimension | Maximum Error (meters) | Mean Error (meters) | Standard deviation (meters) |
|-----------|------------------------|---------------------|-----------------------------|
| x         | 0.28895m               | 0.038632m           | 0.07401m                    |
| y         | 0.45459m               | 0.066087m           | 0.11926m                    |
| z         | 0.81075m               | 0.14734m            | 0.09524m                    |
| distance  | 0.97337m               | 0.16604m            | 0.1696m                     |

**Table 7.1** – Maximum, mean and standard deviation errors in x,y,z and time dimension (along-track) between the planned and travelled path.

<sup>5</sup>The reader is referred to Section 2.2 for details about the vehicle.

If the time dimension is disregarded<sup>6</sup>, the x,y and z distance that the vehicle is from the path is known as the cross-track error. The cross-track errors shown in Table 7.2 also decrease if the sampling time of the simulation is decreased.

| Dimension | Maximum Error (meter) | Mean Error (meters) | Standard deviation (meters) |
|-----------|-----------------------|---------------------|-----------------------------|
| x         | 0.11494m              | 0.028716m           | 0.015158m                   |
| y         | 0.22695m              | 0.080904m           | 0.029007m                   |
| z         | 0.27853m              | 0.10747m            | 0.025969m                   |
| distance  | 0.37722m              | 0.13755m            | 0.0418m                     |

**Table 7.2** – Maximum, mean and standard deviation errors in x,y,z dimension (cross-track) between the planned and travelled path.

Following the results from Tables 7.1 and 7.2, it is concluded that the manoeuvres chosen for this project are executable by the vehicle, and that the vehicle stays within its envelope of operation.

### 7.3.2 Cost function

For this project a cost function was determined to facilitate comparing two manoeuvres and determine which is more cost effective. This cost function is established using measurements from the Simulink simulation, specifically, measurements of the force the main and tail rotor exert while executing a specific manoeuvre. The goal of the cost function is to determine the cost of a manoeuvre during motion planning by the PRM or RRT\*, that is, it must provide an approximation of the cost to execute any given manoeuvre. In this subsection it is determined whether the cost function approximation is close to the Simulink cost of executing a path.

|               | Cost (relative units) |
|---------------|-----------------------|
| Simulink      | 6987                  |
| Cost function | 7393                  |
| Difference    | 5.1%                  |

**Table 7.3** – Difference in cost between Simulink simulation and that determined by the cost function.

In Table 7.3 the path costs calculated by the cost function and the Simulink simulation are shown, and the difference between the costs is slightly more than 5%, which is acceptable. These costs are calculated for the path shown in Figure 7.5.

## 7.4 Conclusion

This chapter concerns validating theoretical assumptions related to the manoeuvre choice in Section 4.5 as well the cost function introduced in Subsection 3.4.4. To validate these assumptions a Simulink simulation is used, where the controllers therein are software implementations of controllers used in real world flight tests of the vehicle.

<sup>6</sup>This means that the vehicle does not adhere to a specific time it must reach a x,y and z coordinate

In Section 7.2 the Simulink model is presented as well as the changes necessary to enable execution of manoeuvres.

In Section 7.3 results following from the Simulink simulation are shown. Specifically, Subsection 7.3.1 presents the maximum along and cross-track errors. The errors are within acceptable bounds which indicates that the Simulink vehicle model was capable of executing the given path where the path consisted of a sequence of manoeuvres. That is, the vehicle operated within its envelope of operation indicating that the theoretical assumptions regarding manoeuvre choices are sound. In Subsection 7.3.2 the (actual) cost of the path determined through measurements of the Simulink simulation are compared to the calculation of the path cost during motion planning via the cost function. It is seen that the calculated cost is close to the actual cost which indicates that the assumptions regarding the cost function are valid.

## Chapter 8

# Environments

This project forms part of a larger goal to achieve autonomous navigation where this project specifically focuses on the motion planning aspect thereof. The goal of this chapter is to show different environments and solutions to the respective problems they present to the motion planning algorithms. Refer to Section 4.2 for details about the software and hardware used in this project. For the planning, a turning radius of 50m is used, and the parameter  $\eta$  set to 641m.

### 8.1 Environment one

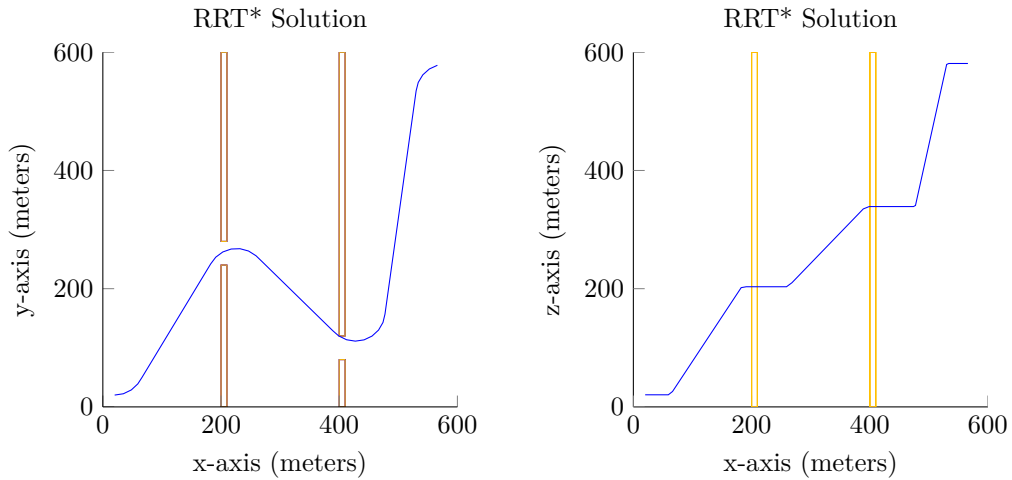
The first environment is designed to test the ability of an algorithm to find a path through a relatively small gap. As per the discussions of Sections 5.2.2 and 5.3.1, a small gap presents the motion planning algorithms with the problem of low connectivity between regions on either side of it. This means that only a few milestones in one region can reach milestones through the gap in another region. Note that a large part of the analysis for this environment is presented in Section 5.4.

This environment represents any real world situation where tight openings are present. This might be open doors inside a building, a situation where a ground based vehicle must move across a bridge, or where a helicopter must fly above a specific patch of land (e.g. the walls in the environment can represent a keep out zone where the helicopter can be shot down).

#### 8.1.1 Performance of the RRT\* vs the PRM

##### The RRT\* solution

The path shown in Figure 8.1 is generated by the RRT\* motion planning algorithm.

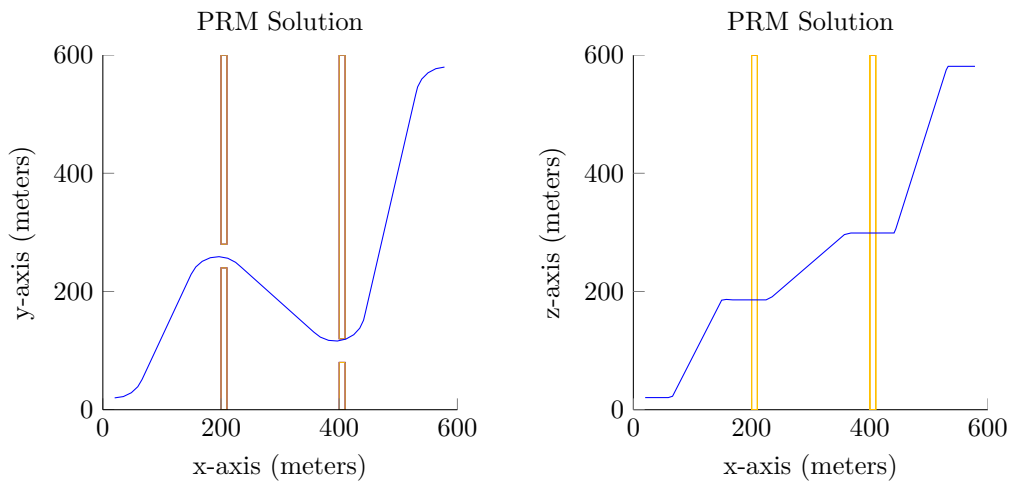


(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.1** – Path generated by the RRT\*.

### The PRM solution

The path shown in Figure 8.2 is generated by the PRM motion planning algorithm.



(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.2** – Path generated by the PRM.

### PRM and RRT\* solutions

The best paths found by both algorithms are almost identical and no clear distinction is observable.

### PRM and RRT\* histograms

Histograms for this solution, and their in depth analysis can be found in Section 5.4. From the histograms in Section 5.4, the RRT\* requires less CPU-time to find a path in this environment. The quality of the paths found by the RRT\* is also better than that of the PRM. From this, the RRT\* is the recommended motion planning algorithm to use in environments similar to environment shown in this section.

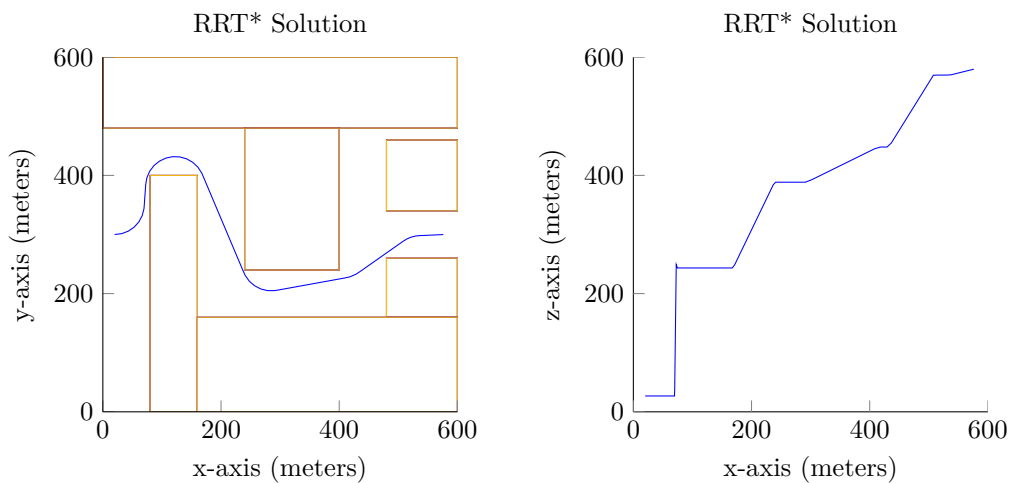
## 8.2 Environment two

The environment in the previous section had plenty of open space for manoeuvrability, which is not always the case. The environment in this section has little open space compared to the amount of free space of the previous environment. Another difference in this environment is that the corridors are too narrow for a turn, i.e. the diameter of a turn is 100m, and the corridors are only 80m wide.

This environment represents any real world situation where the vehicle has very little space to manoeuvre. This might be inside corridors of a building, or in urban environments wherein the vehicle must fly close to the ground (e.g. for surveillance purposes).

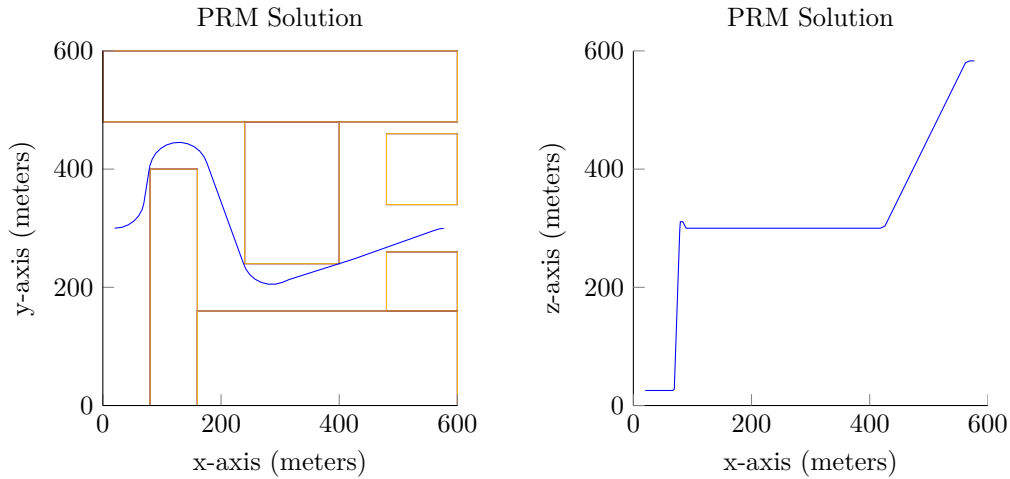
### 8.2.1 Performance of the RRT\* vs the PRM

#### RRT\* Solution



(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.3** – Path generated by the RRT\*.

**PRM Solution**

(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.4** – Path generated by the PRM.

**PRM and RRT\* solutions**

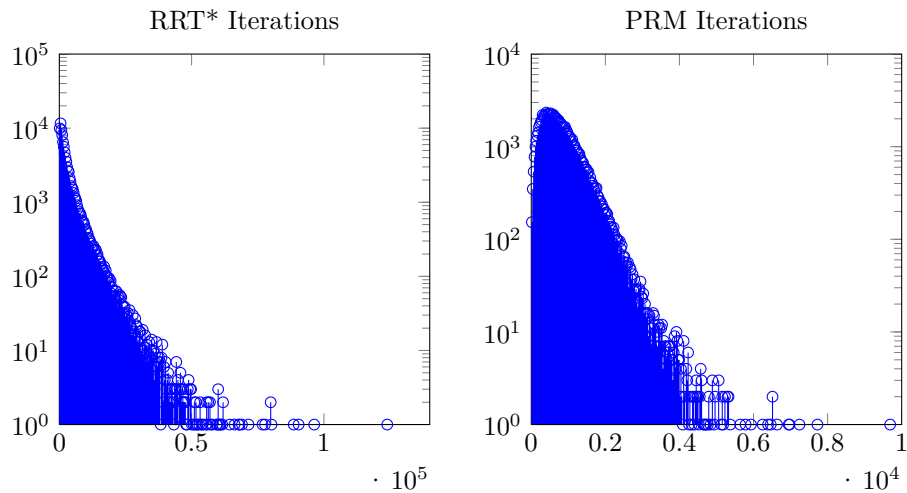
In this section, and the next, the number of corners a path very nearly touches is compared. A path is said to be hugging a corner when it very nearly touches the corner. In an environment with narrow corridors, an algorithm will possibly hug many corners if it struggles to add milestones to its tree. This is seen consistently throughout this and the next section.

The best path found by the RRT\* has much more curves when compared to the PRM solution. A possible reason for this is because the manoeuvres of the RRT\* are limited by  $\eta$ , and in environments with narrow corridors, it is easier for the RRT\* to find paths with lots of curves. The best path found by the PRM hugs four corners of the environment very tightly, compared to the three corners tightly hugged by the RRT\*.

In the side view, the RRT\* has a much more gradual climb, when compared to the PRM.

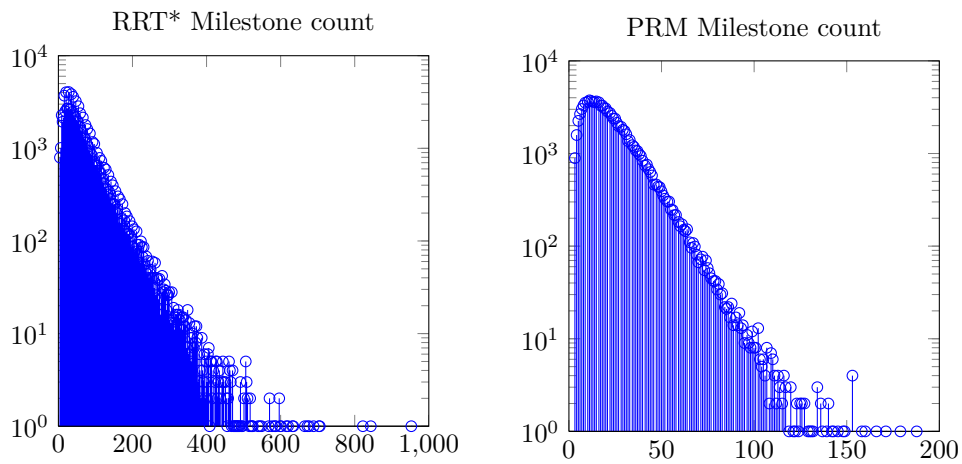


## PRM and RRT\* histograms



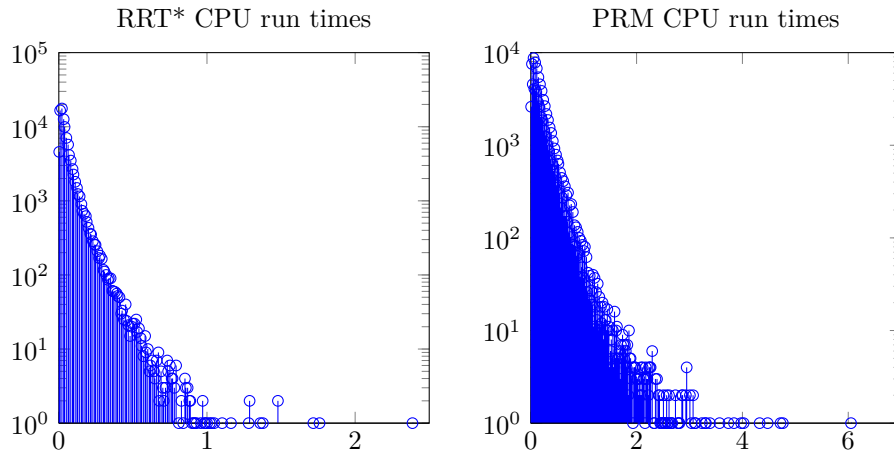
(a) RRT\*: Histogram showing the number of iterations required to find solutions in this environment. The maximum number of iterations is 123948, and the average is 3500. (b) PRM: Histogram showing the number of iterations required to find solutions in this environment. The maximum number of iterations is 9688, and the average is 815.

**Figure 8.5** – Iteration histograms of 100000 runs for both the PRM and RRT\* algorithms.



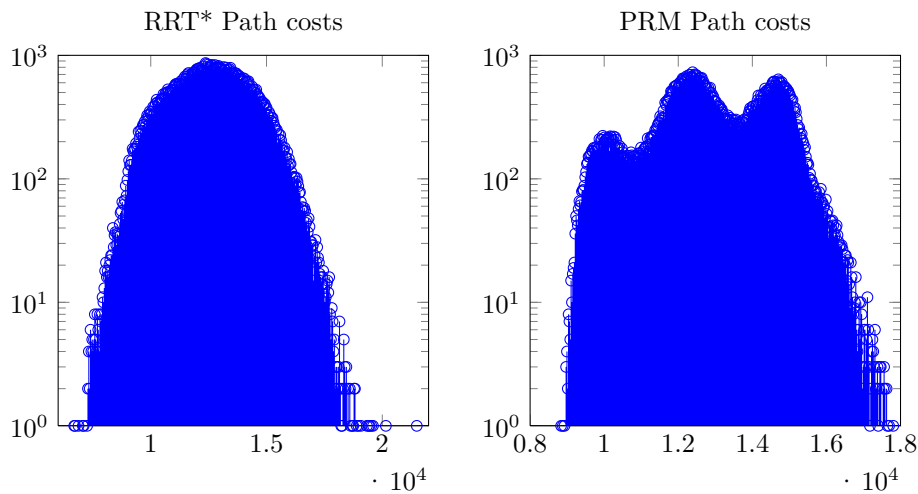
(a) RRT\*: Histogram showing the number of milestones required to find solutions in this environment. The maximum number of milestones is 953, and the average is 63. (b) PRM: Histogram showing the number of milestones required to find solutions in this environment. The maximum number of milestone is 188, and the average is 22.

**Figure 8.6** – Milestone histograms of 100000 runs for both the PRM and RRT\* algorithms.



(a) RRT\*: Histogram showing the CPU times required to find a solution. The maximum CPU time is 2.387 seconds and the average is 0.0555 seconds.  
 (b) PRM: Histogram showing the CPU times required to find a solution. The maximum CPU time is 6.053 seconds and the average is 0.2056 seconds.

**Figure 8.7** – CPU time histograms of 100000 runs for both the PRM and RRT\* algorithms.



(a) RRT\*: Histogram showing the Path costs of the solutions. The maximum cost is 21494 units and the average cost is 12608 units.  
 (b) PRM: Histogram showing the Path costs of the solutions. The maximum cost is 17804 units and the average cost is 12948 units.

**Figure 8.8** – Path cost histograms of 100000 runs for both the PRM and RRT\* algorithms.

From Figures 8.5(a), 8.5(b), 8.6(a), 8.6(b), it is seen that the RRT\* used much more iterations and milestones in comparison to the PRM.

When comparing the CPU-time (Figures 8.7(a), and 8.7(b)) required by the PRM and RRT\*, the RRT\* clearly finds solutions quicker on average, and the maximum time required by the RRT\* is also far shorter than that required by the PRM.

From Figures 8.8(a) and 8.8(b), it is seen that while the average path cost of the RRT\* is slightly lower than the PRM, the maximum path cost is significantly higher of the RRT\*. Interestingly, the histogram of the PRM forms three peaks, which is possibly due to there being three lines of possible solutions, where each line has a significantly different cost. This is because the manoeuvres of the PRM are not limited (as opposed to that of the RRT\*), and the area wherein the PRM must sample milestones to be able to connect them to its tree, is therefore much smaller than the area wherein the RRT\* may sample milestones and still connect them to its tree. This is evident from the much longer solve time by PRM, the much lower amount of milestones present in the tree of the PRM, and the histogram of the PRM having three clear peaks.

Looking at the above comparison, the RRT\* is the best choice to use in similar environments.

### 8.2.2 Summary

The data from the histograms in the previous subsection is summarised in Table 8.1.

|            | PRM     |         | RRT*    |         |
|------------|---------|---------|---------|---------|
|            | Maximum | Average | Maximum | Average |
| Iterations | 9688    | 815     | 123948  | 3500    |
| Milestones | 188     | 22      | 953     | 63      |
| CPU times  | 6.0530s | 0.2056s | 2.3870s | 0.0555s |
| Path Costs | 17804   | 12948   | 21494   | 12608   |

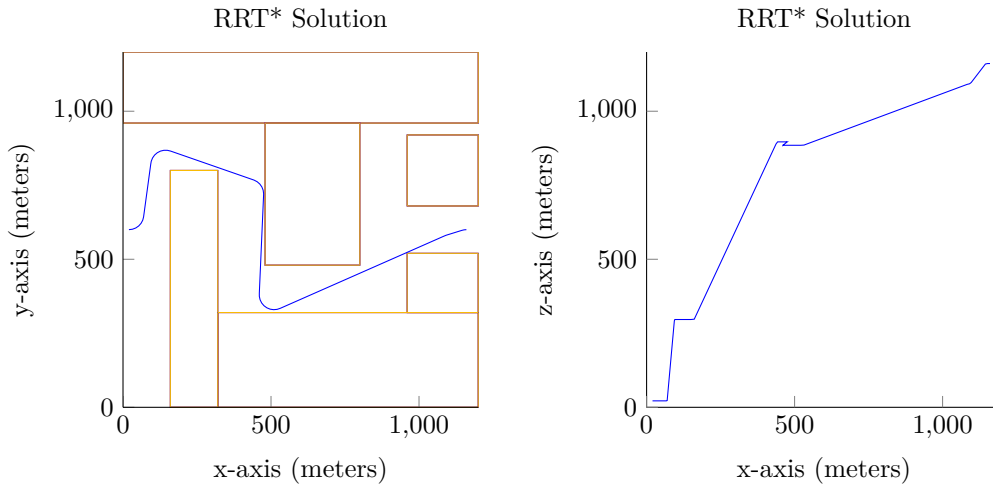
**Table 8.1** – Summary of the iteration, milestone, CPU time, and path cost histograms from Subsection 8.2.1.

### 8.3 Environment three

This environment is similar to the environment in Section 8.2, with the exception that everything in the environment is scaled up by a factor of two. The vehicle's turning radius, and  $\eta$  are kept the same.

### 8.3.1 Performance of the RRT\* vs the PRM

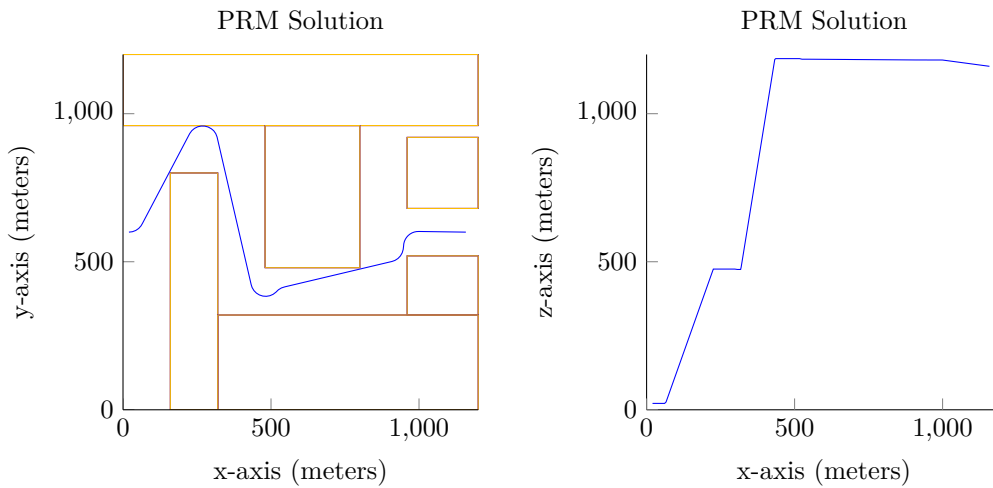
#### RRT\* Solution



(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.9** – Path generated by the RRT\*.

#### PRM Solution



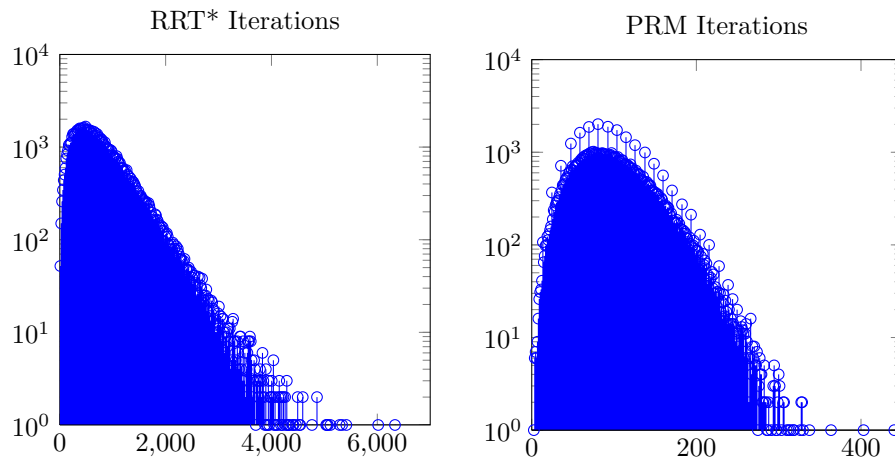
(a) A plot showing the top view of a solution in this environment. (b) A plot showing the side view of a solution in this environment.

**Figure 8.10** – Path generated by the PRM.

### PRM and RRT\* solutions

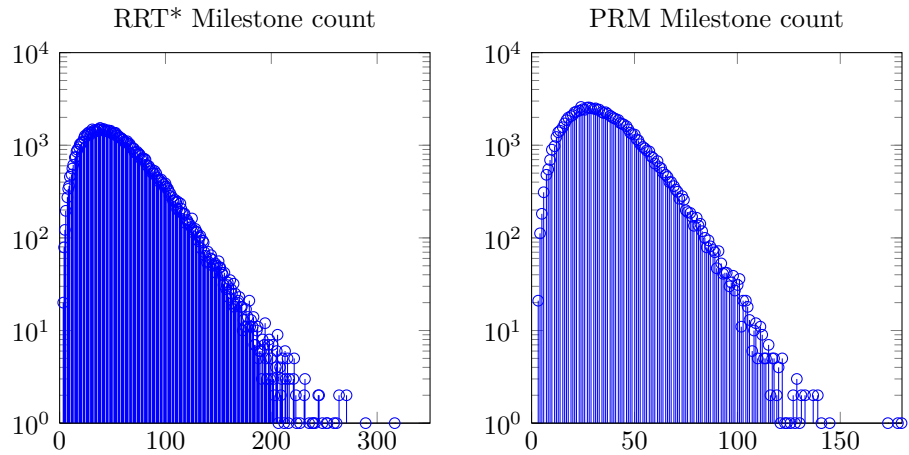
The best path found by the RRT\* differs from the path found in the previous section (same environment, but scaled down). In this section, the manoeuvrability of the vehicle isn't as tightly constrained, and the amount of corners that are tightly hugged are less (only two). For the PRM, the best path also differs from the previous section, and the number of tightly hugged corners is down to only two.

### PRM and RRT\* histograms



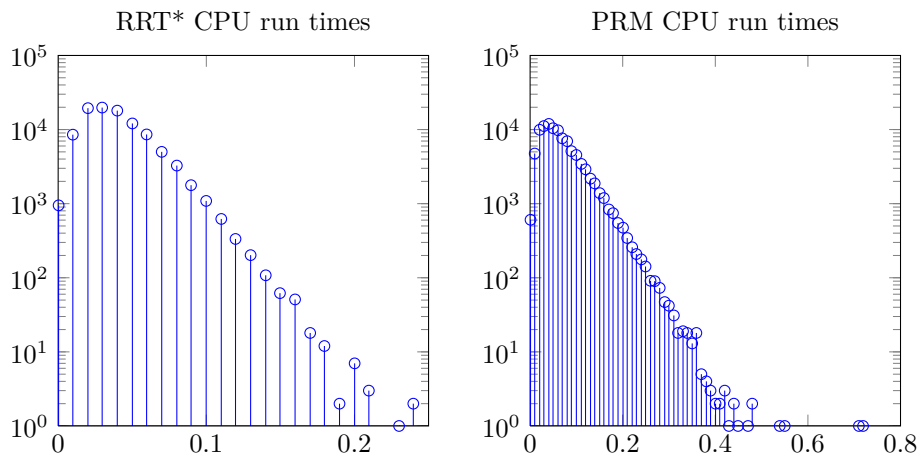
(a) Histogram showing the amount of iterations required to find solutions in this environment. The maximum iteration amount is 6334 and the average is 782. (b) Histogram showing the amount of iterations required to find solutions in this environment. The maximum iteration amount is 440 and the average is 98.

**Figure 8.11** – Iteration histograms of 100000 runs for both the PRM and RRT\* algorithms.



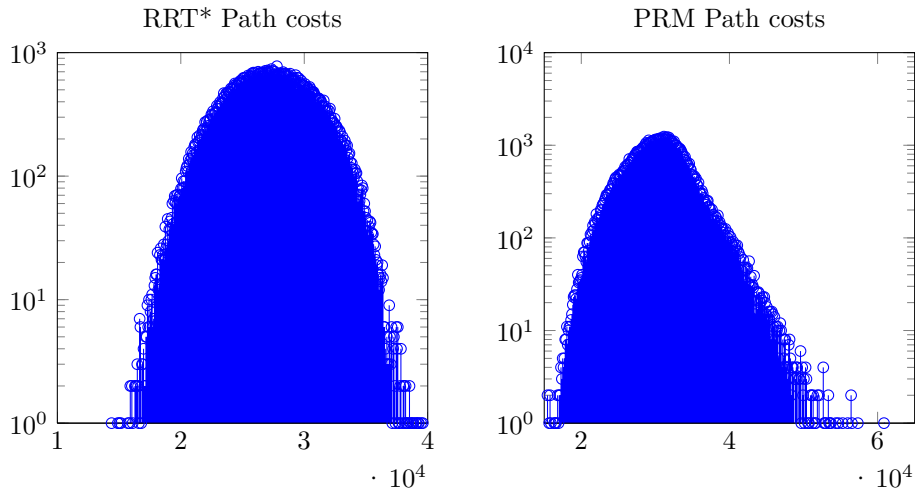
(a) Histogram showing the amount of milestones required to find solutions in this environment. The maximum milestone amount is 316 and the average is 55.  
 (b) Histogram showing the amount of milestones required to find solutions in this environment. The maximum milestone amount is 180 and the average is 35.

**Figure 8.12** – Milestone histograms of 100000 runs for both the PRM and RRT\* algorithms.



(a) Histogram showing the CPU times required to find a solution. The maximum CPU time is 0.2400 seconds and the average is 0.0400 seconds.  
 (b) Histogram showing the CPU times required to find a solution. The maximum CPU time is 0.72 seconds and the average is 0.0671 seconds.

**Figure 8.13** – CPU time histograms of 100000 runs for both the PRM and RRT\* algorithms.



(a) Histogram showing the Path costs of the solutions. The maximum cost is 39595 units and the average cost is 27247 units  
 (b) Histogram showing the Path costs of the solutions. The maximum cost is 60862 units and the average cost is 30496 units

**Figure 8.14** – Path cost histograms of 100000 runs for both the PRM and RRT\* algorithms.

From Figures 8.11(a), 8.11(b), 8.12(a), and 8.12(b), it is seen that the RRT\* uses much more iterations than the PRM; however, in contrast to the previous section where the same, but scaled down environment is used, the number of milestones in the RRT\* and PRM trees are not that different. This is because the RRT\* doesn't need as many milestones (when compared to the previous section) before it can find a path. The maximum milestones required for the PRM is even less than the previous section, even though the environment of this section is double in size.

When comparing the CPU-time (Figures 8.13(a) and 8.13(b)) to that of the previous section, the maximum CPU time required by the RRT\* and PRM is far lower.

From Figures 8.14(a) and 8.14(b), the paths found by the RRT\* have a lower average and maximum cost in comparison to the PRM. Comparing the path costs of this section to the previous section does not make sense as the lengths of the paths in this section are much longer. However, note that the histogram of the PRM has a single peak, more evidence that the PRM didn't struggle as much as in the previous section.

Looking at the above comparison, the RRT\* is the best choice to use in similar environments, independent of scale.

### 8.3.2 Summary

The data from the histograms in the previous subsection is summarised in Table 8.2.

|            | PRM     |         | RRT*    |         |
|------------|---------|---------|---------|---------|
|            | Maximum | Average | Maximum | Average |
| Iterations | 440     | 98      | 6334    | 782     |
| Milestones | 180     | 35      | 316     | 55      |
| CPU times  | 0.72s   | 0.0671s | 0.2400s | 0.0400s |
| Path Costs | 60862   | 30496   | 39595   | 27247   |

**Table 8.2** – Summary of the iteration, milestone, CPU time, and path cost histograms from Section 8.3.



## Chapter 9

# Conclusion

This chapter concludes the project with remarks on the work achieved throughout this document.

### 9.1 Project Scope

Chapter 1 establishes that this project is part of a larger project to achieve autonomous navigation, as well as establishing that this project firstly looks at problems presented by the path planner and conflict resolution modules within the greater autonomous navigation system architecture. For this project these problems are solved using a class of algorithms known as motion planning algorithms.

### 9.2 Motion planning algorithms

After defining the project scope, Chapter 3 presents this project's problem statement, with which it defines the problems presented by the path planner and conflict resolution modules. To solve the problem statement, a class of algorithms known as motion planning algorithms are proposed, for which several algorithms exist. Generally speaking, these algorithms plan a path from a given initial point to a given end point. These planning algorithms include:

- **Complete** algorithms which provides formal proof of guaranteeing to find a path, however they are proven to be computationally too expensive.
- **Almost Complete** algorithms which lack the proof of guaranteeing to find path, however they provide similar guarantees:
  - **Grid** based algorithms which guarantee resolution completeness, however they become computationally too expensive for high grid resolutions as well as high dimensional problems.
  - **Potential Field** based algorithms which use Voronoi vertex generation to avoid getting stuck in local minima. However, the vertex generation becomes computationally too expensive for environments containing many obstacles.
  - **Sampling** based algorithms which provide probabilistic completeness and have been successfully used in real world projects.

Sampling based algorithms are therefore used in this project, specifically the Probabilistic Roadmap Method (PRM) and the Rapidly exploring Random Tree\* (RRT\*).

### 9.3 Algorithm implementation

Next this project discusses in Chapter 4 the data structures used for the PRM and RRT\*, as well as presenting details of the implementations and improvements of the algorithms.

#### 9.3.1 Probabilistic Roadmap Method

The classic PRM is implemented with a small improvement for this project. The small improvement is to sort potential parent milestones according to the distance they are from a newly sampled milestone and only iterate through this sorted list until a parent can connect to the newly sampled milestone. While this does not guarantee an optimal parent for a newly sampled milestone it is still a good approximation of a locally distance-optimal parent. This improvement reduces the CPU time required by the algorithm.

#### 9.3.2 Rapidly exploring Random Tree\*

The basic RRT\* explores an environment until it happens to reach a goal area, which is undesirable. A proposed addition is to use the Local Planning Method (already developed for the PRM) to connect all newly reached milestones to the end milestone. This enables the RRT\* to reach the end milestone as soon as it is 'visible' from a known milestone, as well as reaching the goal at a specific position and heading. This also reduces the CPU time required by the algorithm.

### 9.4 Analysis

After the implementation and improvements to the RRT\* and PRM motion planning algorithms are discussed, the algorithms are analysed in a specific environment in Chapter 5. The techniques used for this analysis can easily be extended to other environments.

#### 9.4.1 Theoretical

This project looks at theoretically determining performance bounds for the PRM and RRT\*. For both algorithms their proof of probabilistic completeness is rewritten so that portions of it can be determined for a chosen environment using a computer. Using these rewritten forms, bounds on the number of milestones the PRM requires to guarantee finding a path (with a chosen probability) as well as the bounds on the number iterations the RRT\* requires (with a chosen probability) is found. The portion of the probabilistic proof required to determine the PRM's performance bound is also rewritten into a less conservative form. The theoretical performance bounds of the PRM and RRT\* is tested using a histogram analysis of the iterations and milestones required by the algorithms and it is seen that the theoretical results hold.

#### 9.4.2 Histogram

For this project a histogram analysis is also conducted. For this analysis histograms are constructed of 100000 algorithm runs whereby CPU time and Path costs are represented as a distribution. Refer to Section 4.2 for details about the hardware and software used in this project.

#### 9.4.2.1 CPU time

A histogram analysis of the CPU time the PRM and RRT\* requires to find a path reveals the PRM algorithm has an acceptably fast average solve time (0.0470 seconds), however its distribution's shape has a long tail ending at a maximum solve time of 1.598 seconds. The RRT\* has an even faster average solve time (0.0285 seconds), and a maximum solve time of only 0.330 seconds. The shape of the RRT\*'s histogram suggests that it is much more consistent in its behaviour regarding solve time.

#### 9.4.2.2 Path cost

A histogram analysis of the cost to execute a path found by the PRM and RRT\* algorithms shows that the PRM algorithm's path cost distribution has a sharp peak, and a higher average (16337 vs. 11877 units) relative to that of the RRT\* algorithm. The shape of the path cost distribution of the RRT\* algorithm suggests a much more consistent behaviour regarding the execution cost of a path found by the RRT\* algorithm.

### 9.5 Path Replanning

It was found that the RRT\* has a much more consistent behaviour regarding solve time as well as path costs, and therefore the RRT\* algorithm is extended to implement path replanning. Path replanning enables the algorithm to include new information about the environment post initial planning. This is successfully implemented and shown to work in Chapter 6. It is shown that replanning enabled the vehicle to start moving after only a short duration while still executing a very cost efficient path. It is also shown that a change in the environment is handled by the path replanning algorithm since information about the change can be included after the initial path planning.

### 9.6 Software simulation

In Chapter 7 a Simulink simulation is used to execute a path planned by the RRT\* algorithm by executing the manoeuvres defined in Section 4.5. It is determined that the vehicle is able to follow the path within small error bounds, indicating that the manoeuvre choices used by the Motion Planner is acceptable.

In Subsection 3.4.4 a cost function is described using measurements from the Simulink simulation. In Chapter 7 this cost function is tested by comparing the cost function calculated cost to execute a path to the cost of actually executing the path, and it is seen that the difference is within acceptable bounds.

## Appendix A

# MATLAB Code to execute a Path

```

list = [];
manoeuvre1=struct('m', 6, 'x', 430.0230, 'y', 146.2181, 'z', 237.2146,
't', 589.6619, 'xm', 381.6167, 'ym', 158.7411);
manoeuvre2=struct('m', 8, 'x', 531.5585, 'y', 542.3869, 'z', 580.5756,
't', 1034.5687);
manoeuvre3=struct('m', 4, 'x', 580.0000, 'y', 580.0000, 'z', 580.0000,
't', 1089.5647, 'xm', 579.9931, 'ym', 529.9734);
milestone1=struct('x', 580.0000, 'y', 580.0000, 'z', 580.0000, 'manoeuvre1',
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);
list = [milestone1 list];

manoeuvre1=struct('m', 6, 'x', 341.2132, 'y', 120.2741, 'z', 244.6028,
't', 498.7267, 'xm', 358.7956, 'ym', 167.0807);
manoeuvre2=struct('m', 8, 'x', 364.2043, 'y', 111.8723, 'z', 236.4507,
't', 520.0676);
manoeuvre3=struct('m', -1, 'x', 364.4549, 'y', 111.7786, 'z', 236.7108,
't', 520.0676, 'xm', 381.3661, 'ym', 158.8348);
milestone2=struct('x', 364.4549, 'y', 111.7786, 'z', 236.7108, 'manoeuvre1',
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);
list = [milestone2 list];

manoeuvre1=struct('m', 4, 'x', 260.6835, 'y', 228.0861, 'z', 145.2000, 't',
348.2027, 'xm', 218.2730, 'ym', 201.6031);
manoeuvre2=struct('m', 8, 'x', 316.5014, 'y', 140.4080, 'z', 244.7438, 't',
471.4675);
manoeuvre3=struct('m', 6, 'x', 317.5137, 'y', 138.8706, 'z', 244.8027, 't',
472.8529, 'xm', 358.6795, 'ym', 167.2595);
milestone3=struct('x', 317.5137, 'y', 138.8706, 'z', 244.8027, 'manoeuvre1',
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);
list = [milestone3 list];

manoeuvre1=struct('m', 6, 'x', 118.2914, 'y', 87.5823, 'z', 35.7337, 't',
104.0246, 'xm', 72.0998, 'ym', 106.7223);
manoeuvre2=struct('m', 8, 'x', 169.7299, 'y', 214.5113, 'z', 144.9956, 't',
250.1856);
manoeuvre3=struct('m', 6, 'x', 170.9340, 'y', 217.6972, 'z', 144.6374, 't',
252.5736, 'xm', 123.3904, 'ym', 233.2905);
milestone4=struct('x', 170.9340, 'y', 217.6972, 'z', 144.6374, 'manoeuvre1',

```

```
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);  
list = [milestone4 list];  
  
manoeuvre1=struct('m', 6, 'x', 48.7253, 'y', 29.0750, 'z', 20.1436,  
't', 25.5821, 'xm', 20.0000, 'ym', 70.0000);  
manoeuvre2=struct('m', 8, 'x', 100.7129, 'y', 65.7182, 'z', 35.3658,  
't', 80.1997);  
manoeuvre3=struct('m', -1, 'x', 100.9057, 'y', 65.8539, 'z', 35.2551,  
't', 80.1997, 'xm', 71.9071, 'ym', 106.5866);  
milestone5=struct('x', 100.9057, 'y', 65.8539, 'z', 35.2551, 'manoeuvre1',  
manoeuvre1, 'manoeuvre2', manoeuvre2, 'manoeuvre3', manoeuvre3);  
list = [milestone5 list];
```

## Bibliography

- [1] Groenewald, S.: Development of a rotary-wing test bed for autonomous flight. 2005.
- [2] Hough, W.: Autonomous aerobatic flight of a fixed wing unmanned aerial vehicle. 2007.
- [3] Nye, B.: Mars exploration rovers. 2005.  
Available at: [http://athena.cornell.edu/kids/bn\\_special\\_report.html](http://athena.cornell.edu/kids/bn_special_report.html)
- [4] van Daalen, C.E.: Conflict detection and resolution for autonomous vehicles. 2010.
- [5] Peddle, I.: Autonomous flight of a model aircraft. masters dissertation, stellenbosch university. 2005.
- [6] Coyle, S.: The art and science of flying helicopters, iowa state university press. 1996.
- [7] Rossouw, E.: Autonomous flight of an unmanned heli. 2008.
- [8] Gavrillets, V., Mettler, B. and Feron, E.: Dynamic model for x-cell 60 helicopter in low advance ration flight.
- [9] Medellín-Colombia: 2006.  
Available at: <http://www.control-systems.net>
- [10] Lozano-Perez, T. and Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. 1979.
- [11] Schwartz, J.T. and Sharir, M.: On the âpiano moversâ problem: Ii. general techniques for computing topological properties of real algebraic manifolds. 1982.
- [12] Reif, J.: Complexity of the moverâs problem and generalizations. 1979.
- [13] LaValle, S.M.: Planning algorithms. 2006.
- [14] Ansari, S., Ok, K., Gallagher, B. and Sica, W.: Planning with uncertainty for autonomous uav. 2011.
- [15] Frazzoli, E., Daleh, M.A. and Feron, E.: Real-time motion planning for autonomous vehicles. 2000.
- [16] Hsu, D., Kindel, R., Latombe, J.-C. and Rock, S.: Randomized kinodynamic motion planning with moving obstacles.
- [17] Karaman, S. and Frazzoli, E.: Incremental sampling-based algorithms for optimal motion planning. 2010.
- [18] Kavraki, L. and Latombe, J.: Randomized preprocessing of configuration space for fast path planning. 1994.

- [19] Kavraki, L., Svestka, P., Latombe, J. and Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. 1996.
- [20] LaValle, S.M. and Kuffner, J.J.: Randomized kinodynamic planning. 2001.
- [21] van den Berg, J., Ferguson, D. and Kuffner, J.: Anytime path planning and replanning in dynamic environments. 2006.
- [22] Lavalle, S.: Rapidly-exploring random trees: A new tool for path planning. 1998.
- [23] Kwak, J.: Rough terrain navigation for mars rovers. 2007.
- [24] Donald, B., Xavier, P., Canny, J. and Reif, J.: Kinodynamic motion planning. 1993.
- [25] Frazzoli, E., Dahleh, M.A. and Feron, E.: Robust hybrid control for autonomous vehicle motion planning.