COPYRIGHT AND USE OF THIS THESIS

This thesis must be used in accordance with the provisions of the Copyright Act 1968.

Reproduction of material protected by copyright may be an infringement of copyright and copyright owners may be entitled to take legal action against persons who infringe their copyright.

Section 51 (2) of the Copyright Act permits an authorized officer of a university library or archives to provide a copy (by communication or otherwise) of an unpublished thesis kept in the library or archives, to a person who satisfies the authorized officer that he or she requires the reproduction for the purposes of research or study.

The Copyright Act grants the creator of a work a number of moral rights, specifically the right of attribution, the right against false attribution and the right of integrity.

You may infringe the author’s moral rights if you:

- fail to acknowledge the author of this thesis if you quote sections from the work
- attribute this thesis to another author
- subject this thesis to derogatory treatment which may prejudice the author’s reputation

For further information contact the University’s Copyright Service.

sydney.edu.au/copyright
Provably-Correct Task Planning for Autonomous Outdoor Robots

Chanyeol Yoo,
BE (Hons 1, The University of Auckland)

A thesis submitted in fulfillment of the requirements of the degree of Doctor of Philosophy

Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

Submitted October 2014; revised December 2015
Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Chanyeol Yoo

3 December 2015
Abstract

Chanyeol Yoo
Doctor of Philosophy
The University of Sydney
December 2015

Provably-Correct Task Planning for Autonomous Outdoor Robots

Autonomous outdoor robots should be able to accomplish complex tasks safely and reliably while considering constraints that arise from both the environment and the physical platform. Such tasks extend basic navigation capabilities to specify a sequence of events over time. For example, an autonomous aerial vehicle can be given a surveillance task with contingency plans while complying with rules in regulated airspace, or an autonomous ground robot may need to guarantee a given probability of success while searching for the quickest way to complete the mission. A promising approach for the automatic synthesis of trusted controllers for complex tasks is to employ techniques from formal methods. In formal methods, tasks are formally specified symbolically with temporal logic. The robot then synthesises a controller automatically to execute trusted behaviour that guarantees the satisfaction of specified tasks and regulations. However, a difficulty arises from the lack of expressivity, which means the constraints affecting outdoor robots cannot be specified naturally with temporal logic. The goal of this thesis is to extend the capabilities of formal methods to express the constraints that arise from outdoor applications and synthesise provably-correct controllers with trusted behaviours over time.

This thesis focuses on two important types of constraints, resource and safety constraints, and presents three novel algorithms that express tasks with these constraints and synthesise controllers that satisfy the specification. Firstly, this thesis proposes an extension to probabilistic computation tree logic (PCTL) called resource threshold PCTL (RT-PCTL) that naturally defines the mission specification with continuous
resource threshold constraints; furthermore, it synthesises an optimal control policy with respect to the probability of success. With RT-PCTL, a state with accumulated resource out of the specified bound is considered to be failed or saturated depending on the specification. The requirements on resource bounds are naturally encoded in the symbolic specification, followed by the automatic synthesis of an optimal controller with respect to the probability of success. Secondly, the thesis proposes an online algorithm called greedy Büchi algorithm (GBA) that reduces the synthesis problem size to avoid the scalability problem. A framework is then presented with realistic control dynamics and physical assumptions in the environment such as wind estimation and fuel constraints. The time and space complexity for the framework is polynomial in the size of the system state, which is efficient for online synthesis. Lastly, the thesis proposes a synthesis algorithm for an optimal controller with respect to completion time given the minimum safety constraints. The algorithm naturally balances between completion time and safety. This work proves an analytical relationship between the probability of success and the conditional completion time given the mission specification. The theoretical contributions in this thesis are validated through realistic simulation examples.

This thesis identifies and solves two core problems that contribute to the overall vision of developing a theoretical basis for trusted behaviour in outdoor robots. These contributions serve as a foundation for further research in multi-constrained task planning where a number of different constraints are considered simultaneously within a single framework.
Acknowledgements

This thesis officially celebrates the end of my 19.5 years of student life across three countries. This day would not have come without the helps of my great supervisors and I would like to acknowledge them first. I would like to thank Professor Salah Sukkarieh who has accepted my PhD application from Auckland to Australian Centre for Field Robotics (ACFR). He has inspired me to consider the problems addressed in this thesis. With his supervision and insights in field robotics, I was not only focusing on theoretical work, but also the real outdoor robots used in practice. I also thank Dr. Robert Fitch for his great supervision that was full of advice and encouragement. In the days of depression and procrastination, his cheerful comments always encourage me to work harder to achieve above my limit. This thesis would not have been possible without his understanding and hard work.

My thanks also go to Ken, Abdallah, Bryan, Victor and Tariq who have been always friendly to a shy man from New Zealand. A special thank goes to Tariq for giving me tonnes of advices on PhD life. I also would like to thank Zhe, Jen Jen, Bryan and Nick for helping me with my research and proof-reading my papers.

To my family, thank you for your support over the years. Without their endless support, completing a PhD would not have been possible. A very special thank you to my lovely wife Dr. Amy Jisoo Kwon who has always been on my side. She has always given me good advices based on her own experience and made me happy with her warm words. She also did a great job of delivering my lovely son Seojin, whom makes me smile and happy at every moment of my life since his birth.
## Contents

Declaration i

Abstract iii

Acknowledgements v

Contents vii

List of Figures xiii

List of Tables xvii

List of Algorithms xix

List of Theorems xxi

Nomenclature xxiii

1 Introduction 1

1.1 Trusted Behaviour ................................................................. 2

1.2 Resource Constraints .............................................................. 4

1.2.1 Resource Threshold Constraints ............................................. 5

1.2.2 Fuel Constraints in Wind ...................................................... 6

1.3 Safety Constraints ................................................................. 6

1.4 Approaches to Outdoor Task Planning with Constraints .............. 8

1.4.1 Resource Constraints ........................................................... 8
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.2 Safety Constraints</td>
<td>9</td>
</tr>
<tr>
<td>1.4.3 Combinations of Constraints and Other Extensions</td>
<td>10</td>
</tr>
<tr>
<td>1.5 Contributions</td>
<td>11</td>
</tr>
<tr>
<td>1.6 Thesis Outline</td>
<td>12</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>13</td>
</tr>
<tr>
<td>2.1 Formal Methods</td>
<td>13</td>
</tr>
<tr>
<td>2.1.1 Formal Methods in Digital Systems</td>
<td>15</td>
</tr>
<tr>
<td>2.2 Formal Methods in Robotics</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 Task Planning with PCTL</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2 Task Planning with LTL</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Task Planning with Constraints</td>
<td>19</td>
</tr>
<tr>
<td>2.3.1 Resource Constraints</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2 Safety Constraints</td>
<td>21</td>
</tr>
<tr>
<td>2.4 Hybrid System Approach</td>
<td>23</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>24</td>
</tr>
<tr>
<td>3 Background</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Discrete-Time Markov Decision Process (DT-MDP)</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Model-Checking</td>
<td>26</td>
</tr>
<tr>
<td>3.3 Linear Temporal Logic (LTL)</td>
<td>27</td>
</tr>
<tr>
<td>3.3.1 Deterministic Büchi Automaton</td>
<td>28</td>
</tr>
<tr>
<td>3.3.2 Synthesis with LTL</td>
<td>29</td>
</tr>
<tr>
<td>3.4 Probabilistic Computation Tree Logic (PCTL)</td>
<td>32</td>
</tr>
<tr>
<td>3.4.1 Model-Checking with PCTL</td>
<td>34</td>
</tr>
<tr>
<td>3.4.2 Synthesis with PCTL</td>
<td>34</td>
</tr>
</tbody>
</table>
## 4 Task Planning with Resource Threshold Constraints

4.1 Resource Threshold Constraints .......................... 37
   4.1.1 Motivating Example .................................. 38

4.2 Discrete Time Resource-Dependent MDP (DTR-MDP) ....................... 40

4.3 Resource Threshold-PCTL (RT-PCTL) ................................ 42
   4.3.1 Definition of Piecewise-Constant Function .................... 43

4.4 Performance Evaluation of Control Policy ............................ 45
   4.4.1 PPF Solutions for Quantifiers \( \mathcal{X} \), \( \mathcal{U} \), and \( \mathcal{F} \) ................. 46
   4.4.2 Piecewise-Constant Control Policy ......................... 48
   4.4.3 Model-Checking an RT-PCTL State Formula .................... 50
   4.4.4 Calculation Example .................................... 50

4.5 Synthesis of Optimal Control Policies .............................. 52
   4.5.1 Synthesis for Composite Specifications ....................... 57
   4.5.2 Analysis ............................................. 60

4.6 Summary .................................................. 64

## 5 Task Planning with Fuel Constraints in Wind

5.1 Fuel Constraints in Wind .................................... 65

5.2 Models of Platform and Environments .............................. 66
   5.2.1 Breguet Range Equation ................................ 66
   5.2.2 Wind Interpolation with Gaussian Process Regression .......... 67
   5.2.3 UAV Dynamics and Controller Model ......................... 68

5.3 Synthesis for Continuous Trajectories ............................ 69
   5.3.1 Discrete Synthesis .................................... 69
   5.3.2 Continuous Execution ................................... 73
   5.3.3 Analysis ............................................. 76

5.4 Summary .................................................. 77
6 Task Planning with Safety Constraints

6.1 Safety Constraints ................................................. 79

6.1.1 Motivating Example .......................................... 80

6.1.2 Limitations of MDP Formulation ............................. 82

6.2 Expected Time to Accomplish Mission .......................... 84

6.3 Synthesis of Optimal Control Policy ............................ 88

6.3.1 Synthesis Example ........................................... 89

6.4 Summary ......................................................... 91

7 Applications of Task Planning with Constraints ................. 93

7.1 Resource Threshold Constraints ................................. 93

7.1.1 Reach the Goal and Avoid Danger ............................ 94

7.1.2 A Complex High-Level Mission Specification ............... 100

7.1.3 A Sequencing Mission Specification .......................... 101

7.1.4 Altitude Regulation in Large Environment with Realistic Thermal Model ........................................... 101

7.1.5 Surveillance and Contingency Planning with a Solar-Powered Aircraft with Emergency Flight Plan ............... 104

7.2 Fuel Constraints with Wind ...................................... 109

7.2.1 Reach Goal while Avoiding Danger with Direction Constraint 110

7.2.2 Surveillance Mission ........................................... 111

7.2.3 Large Scale Scenario ........................................... 112

7.2.4 GBA with Rapidly-Exploring Random Tree (RRT) ........... 115

7.3 Safety Constraints ............................................... 118

7.3.1 Avoid Slope and Water until Reaching the Goal State .......... 120

7.3.2 Avoid Slope, Water and Mud Adjacent to Water until Reaching the Goal State .................................... 121

7.4 Summary ......................................................... 122
8 Conclusion 123

8.1 Thesis Summary ................................................. 123

8.2 Contributions .................................................. 124

8.2.1 Resource Threshold Constraints .............................. 124

8.2.2 Fuel Constraints in Wind .................................... 125

8.2.3 Safety Constraints ............................................ 125

8.2.4 Applications .................................................. 125

8.3 Future Work ...................................................... 126

8.3.1 Unified Frameworks ........................................... 126

8.3.2 Dynamic and Uncertain Environments ....................... 127

8.3.3 Complex Models .............................................. 129

8.3.4 Guaranteed Satisfaction ..................................... 129

8.3.5 Algorithmic Improvements .................................. 130

Bibliography 131
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Overview of the research problems in this thesis</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>A simple tree of the temporal logic family</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>An overview of a hybrid system for task planning</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>A diagram of the model-checking approach</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Constructed Büchi automaton of LTL formula</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>The transition system with 9 states</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>The product automaton of Figure 3.2 and 3.3</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>A simple example environment and computation trees</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Optimal control policy and PPF for the simple scenario in Figure 4.1a</td>
<td>56</td>
</tr>
<tr>
<td>5.1</td>
<td>A simple environment and Büchi automaton</td>
<td>70</td>
</tr>
<tr>
<td>5.2</td>
<td>A transition from discrete state $s$ to $s'$ is shown with wind vectors</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>The UAV is to visit $s_1$ and $s_9$ infinitely often while avoiding $s_4$</td>
<td>75</td>
</tr>
<tr>
<td>6.1</td>
<td>Simple demonstration environment</td>
<td>80</td>
</tr>
<tr>
<td>6.2</td>
<td>Balancing the reward matrix in an MDP formulation with different sets of costs for danger states</td>
<td>81</td>
</tr>
<tr>
<td>6.3</td>
<td>Maximising the success probability with respect to the given PCTL formula over a set of atomic propositions without any constraints</td>
<td>83</td>
</tr>
<tr>
<td>6.4</td>
<td>Synthesis results with different safety constraints and the corresponding minimum probability of success</td>
<td>90</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.1</td>
<td>Glider dynamics in a hexagonal grid</td>
<td>95</td>
</tr>
<tr>
<td>7.2</td>
<td>Most probable path of a glider using the heuristic control policy in Equation 7.1</td>
<td>96</td>
</tr>
<tr>
<td>7.3</td>
<td>Success probability with respect to entering altitude at state $(s_{68}, d_1)$ for the exact PPF solution compared with the discrete resource representation</td>
<td>97</td>
</tr>
<tr>
<td>7.4</td>
<td>Most probable paths at several starting altitudes (9.4m, 9.6m and 9.8m), with probability of mission success as indicated</td>
<td>98</td>
</tr>
<tr>
<td>7.5</td>
<td>Optimal path and PPFs for different starting states</td>
<td>99</td>
</tr>
<tr>
<td>7.6</td>
<td>Most probable path using the optimal policy for the scenario in Section 7.1.2</td>
<td>100</td>
</tr>
<tr>
<td>7.7</td>
<td>Most probable path for the scenario in Section 7.1.3 using an optimal policy</td>
<td>102</td>
</tr>
<tr>
<td>7.8</td>
<td>Thermal map modelled with Gedeon’s thermal model presented in Equation 7.7</td>
<td>103</td>
</tr>
<tr>
<td>7.9</td>
<td>The UAV trajectory shown in 3D</td>
<td>104</td>
</tr>
<tr>
<td>7.10</td>
<td>Most probable path for a solar-powered surveillance aircraft without emergency flight plans</td>
<td>105</td>
</tr>
<tr>
<td>7.11</td>
<td>Most probable path for a solar-powered surveillance aircraft with emergency flight plans</td>
<td>106</td>
</tr>
<tr>
<td>7.12</td>
<td>Probabilities of satisfaction with respect to primary battery level at state $(s_{78}, d_4)$ within a 30-step time horizon</td>
<td>107</td>
</tr>
<tr>
<td>7.13</td>
<td>Wind vectors drawn on the environment sized 2000m by 2000m</td>
<td>109</td>
</tr>
<tr>
<td>7.14</td>
<td>Comparing two different algorithms in the same environment</td>
<td>110</td>
</tr>
<tr>
<td>7.15</td>
<td>A continuous execution of the surveillance mission</td>
<td>112</td>
</tr>
<tr>
<td>7.16</td>
<td>Accumulated elapsed travel and synthesis time with respect to iterations, with 10000 discrete states</td>
<td>114</td>
</tr>
<tr>
<td>7.17</td>
<td>Timeline for partial synthesis and execution</td>
<td>114</td>
</tr>
<tr>
<td>7.18</td>
<td>RRT-variant of the scenario in Section 7.2.2 for fast searching</td>
<td>116</td>
</tr>
<tr>
<td>7.19</td>
<td>RRT-variant of the scenario in Section 7.2.2 where the number of nodes must be greater than $10^4$</td>
<td>118</td>
</tr>
<tr>
<td>7.20</td>
<td>10x10 simulation environment with steep slope (red), water (blue), mud (brown) and goal (green) states</td>
<td>119</td>
</tr>
</tbody>
</table>
7.21 Synthesised control policies for different probability thresholds for Section 7.3.1 .................................................. 120

7.22 Synthesised control policies for different probability thresholds for Section 7.3.2 .................................................. 121
List of Tables

7.1 Task-level synthesis time for different numbers of discrete states . . . 113
7.2 Time analysis for partial synthesis with RRT . . . . . . . . . . . . . . 117
List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Evaluation of PPF with piecewise control policy</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Compute piecewise maximum function and index function with respect to array of piecewise functions</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Compute optimal control policy and PPFs</td>
<td>54</td>
</tr>
<tr>
<td>5.1</td>
<td>Synthesis of optimal sequence to next Büchi State</td>
<td>72</td>
</tr>
<tr>
<td>5.2</td>
<td>Synthesis of continuous trajectory from sequence of discrete states</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Overall execution of controller in continuous state space</td>
<td>76</td>
</tr>
<tr>
<td>6.1</td>
<td>Algorithm for conditional completion time $\tau(s \mid v), \forall s \in S$</td>
<td>86</td>
</tr>
<tr>
<td>6.2</td>
<td>Synthesis algorithm for optimal control policy $\pi^*$ (dynamic programming)</td>
<td>89</td>
</tr>
</tbody>
</table>
List of Theorems

1. Lemma (Bounded cardinality of subdomain set) ............... 61
2. Lemma (Convergence of PPF in special cases) ............... 62
3. Lemma (Convergence of PPF in general cases) ............... 62
4. Lemma (Maximum number of iterations before convergence) .... 63
4.1 Theorem (PPF convergence to unique fixed point) ............ 64
Nomenclature

List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACFR</td>
<td>Australian Centre for Field Robotics</td>
</tr>
<tr>
<td>CTL</td>
<td>computation tree logic</td>
</tr>
<tr>
<td>PCTL</td>
<td>probabilistic computation tree logic</td>
</tr>
<tr>
<td>RT-PCTL</td>
<td>resource threshold-PCTL</td>
</tr>
<tr>
<td>LTL</td>
<td>linear temporal logic</td>
</tr>
<tr>
<td>MDP</td>
<td>Markov decision process</td>
</tr>
<tr>
<td>MC</td>
<td>Markov chain</td>
</tr>
<tr>
<td>DT-MDP</td>
<td>discrete-time Markov decision process</td>
</tr>
<tr>
<td>DTR-MDP</td>
<td>discrete-time resource-dependent Markov decision process</td>
</tr>
<tr>
<td>DT-MC</td>
<td>discrete-time Markov chain</td>
</tr>
<tr>
<td>DTR-MC</td>
<td>discrete-time resource-dependent Markov chain</td>
</tr>
<tr>
<td>PPF</td>
<td>piecewise-constant probability function</td>
</tr>
<tr>
<td>MOMDP</td>
<td>mixed-observability MDP</td>
</tr>
<tr>
<td>POMDP</td>
<td>partially-observable MDP</td>
</tr>
<tr>
<td>PRM</td>
<td>probabilistic road map</td>
</tr>
<tr>
<td>GBA</td>
<td>greedy Büchi algorithm</td>
</tr>
<tr>
<td>GR(1)</td>
<td>generalised reactivity (1)</td>
</tr>
<tr>
<td>UAV</td>
<td>unmanned aerial vehicle</td>
</tr>
<tr>
<td>RRT</td>
<td>rapidly-exploring random tree</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Task-level planning is a long-standing problem in robotics. The tasks in this sense are generally more complex than simple navigation of moving from one location to another. For example, such tasks could include surveillance missions for unmanned aerial vehicles (UAVs) along with contingency planning, and search and track missions to find and follow a target of interest. In task planning, the objective is to automatically synthesise low-level controllers that satisfy a high-level specification. It is critical to ensure that such controllers are guaranteed to exhibit trusted behaviour, since failure to do so may cause catastrophic failure.

A promising approach towards the vision of task planning for autonomous robots involves formal methods. Formal methods are mathematical techniques for formal specification and verification of systems. They provide a natural way to define complex tasks in robotics using a class of logic called temporal logic. Using temporal logic, we can specify rich tasks [8, 49] or complex goals [11]. In addition to providing powerful means for representing tasks, formal methods also enable means to prove the correctness of a plan or policy that completes a task. By verifying a plan or policy against its logical specification, we can prove the absence of error, as opposed to testing or simulation which only shows the presence of error. Therefore formal methods are useful for task planning because safety-critical systems with a provably-correct controller can be synthesised automatically from a given task specification.
Task planning for robots in outdoor environments must consider constraints that arise from the environment and the physical platform. For example, a UAV may have to travel through wind while complying with altitude regulations, and an autonomous ground vehicle may have to minimise mission completion time while it ensures a certain level of safety and battery level. Failure to comply with such rules and constraints could lead to unqualified completion or destruction of the platform. For example, if the UAV flies below the legal minimum altitude, it may crash into buildings, and if the battery for the ground robot goes flat, the robot may not be able to complete the mission by itself.

Formal methods, however, are not currently applicable to systems with such constraints. Tasks described using temporal logic are specified symbolically, while most of the constraints that arise in outdoor applications cannot be specified symbolically. For example, it is not trivial to specify the minimum safety and maximum legal altitude for the UAV, and the minimum guaranteed level of safety for the ground vehicles along with the tasks. Also outdoor robots are under the influence of external disturbances and internal factors affecting the overall performance and dynamics of the platform. For example, dynamics of a petrol-powered UAV are affected by wind and its gross weight which decreases with fuel consumption.

This thesis addresses problems that arise in task planning for outdoor autonomous robots. In particular, it focuses on two of the most important constraints, resource and safety, that are vital to successful and safe mission completion, and presents three novel algorithms for expressing task-level specifications and synthesising controllers. Figure 1.1 illustrates an overview of the problems addressed in two parts. The following sections discuss the motivation for studying resource and safety constraints in this context.

1.1 Trusted Behaviour

The main motivation of this thesis is trusted behaviour of robots. Many outdoor robotic applications heavily rely on direct control from a human, or a human-in-the-
Figure 1.1 – Overview of the research problems in this thesis. In the overall vision of outdoor task planning with constraints, two important constraints are identified and addressed: resource and safety constraints.

loop, where a small portion of the system is automated and all the major decisions are made by human operators. The operator is required to make rational decisions with uncertain and limited information. For example, an aircraft with no communication from a ground station may have to find the closest airport in an emergency situation while ensuring the safety of passengers. In these applications, making a poor decision could lead to the loss of platform, property, reputation, and human casualties in the worst case.

Widespread use of fully autonomous robotic systems for outdoor applications will require a high level of trust. The autonomous systems should make a valid decision in the presence of limited, uncertain, and even corrupted information where the environment may have unexpected events. The system should accept any combination of inputs and return an acceptable output.

For safety-critical systems such as commercial aviation, it is important to certify [89] that the systems work for every possible combination of inputs. Generally, a certification is given after checking every line of code and testing/simulating all possibilities that could occur. However, such manual and exhaustive searches consume substantial resources and time. These approaches still do not guarantee the absence of error, as the certification process is prone to human error. Hence, we need a method that
automatically synthesises a provably-correct controller from a given high-level specification, which we refer to as a top-down approach. Such a method should be able to certify the controller in an automated manner. In this thesis, we consider synthesising a provably-correct high-level plan that can be given to a generic control unit, rather than generating provably-correct code.

1.2 Resource Constraints

The operation of autonomous outdoor robots heavily depends on resources. Often, resources are modelled as variables that are simply maximised. However in some scenarios, resources could act as constraints that affect the success of a mission, or could have complex dynamics that are not easily computed. In this section, we discuss two cases of resource constraints: resource threshold constraints and fuel constraints in wind.

Often, the goal of robots with resource constraints is to either maximise/minimise the expected gain/loss of a resource while the robot travels to reach its goal location. Such maximisation or minimisation problems can be solved using a number of motion planning techniques such as dynamic programming, rapidly-exploring random trees (RRTs) and probabilistic road maps (PRMs) with cost functions and transition uncertainty. In outdoor applications, however, the influence of resource constraints on a robotic mission could be complex. For example, a UAV may have to comply with minimum and maximum altitudes during a mission. In such cases, the methods of either maximising or minimising do not represent the desired behaviour. Also, the resource variable represented in the methods is often represented as an expected value over a range of values. As a result, expected value within the allowed bounds could include path fragments that violate the bound requirements, which should be avoided for safety-critical systems.

Complex fuel constraints that arise from external disturbances are also important resource constraints to consider. As outdoor robots often operate over long time-frames,
the efficient use of limited resources is critical for the completion of the overall mission. These outdoor robots often operate with external disturbances. The presence of such disturbances affects the dynamics of the aircraft and the level of fuel consumption. For example, the gross weight of a petrol-powered UAV would change as it travels. The change in weight affects the dynamics of the aircraft. The rate of fuel consumption is heavily affected by the wind in the environment. As capacity of fuel is an important factor for task completion, complex fuel constraints must be considered at the task level. In task-level planning, such fuel constraints could be considered by designing a hybrid system where the change in fuel becomes a symbolic variable and the task-level planner reacts to the change. However, the synthesis of such reactive controllers is computationally hard [74].

### 1.2.1 Resource Threshold Constraints

One of the important types of resource constraints is resource threshold constraints, where the values of resources are bounded. An intuitive application of a resource bound would be to constrain fuel or battery level above some minimum. However, some systems such as autonomous thermal gliders may have less obvious resource threshold constraints. In this case, the ‘resource’ of interest is altitude. The glider gains altitude, and hence energy, by exploiting favourable wind currents. Gliders must operate within a fixed altitude range for several reasons, including maintaining the safety of the platform and to comply with government regulations for autonomous flight. It may seem reasonable to model altitude discretely, but as with all discrete approximation it is then necessary to choose an appropriate resolution. Even with a very fine resolution, discrete approximation can lead to inaccurate evaluation of the safety criteria. For example, there may exist conditions where the length scale of wind features is less than the discretisation of altitude. A policy could then lead the glider into an unsafe wind gust yet is evaluated as safe. From the perspective of formal methods, there is no strong guarantee in the evaluation since such approximation may find the presence of certain behaviour, but not the absence of such behaviour.


1.2.2 Fuel Constraints in Wind

The rate of fuel consumption is heavily dependent on external forces acting on robots. Especially for outdoor robots that have limited opportunity for refuelling or recharging, such fuel constraints are an important type of resource constraints to consider. Fuel constraints are important for UAVs because violation can lead to catastrophic failure. We would like to specify tasks that guarantee safe operation such as returning to base when the fuel level drops below a threshold, and ensuring that a suitable landing site is reachable at all times in the event of an emergency. Important work in robotics has explored hybrid controllers where rich tasks are specified as linear temporal logic (LTL) formulas at a high-level discrete layer, and continuous controllers are designed or synthesised that execute the high-level behaviours [11, 83, 38]. Fuel constraints introduce a challenging case because it is undesirable to model such continuous values discretely in the high-level [94], yet task specifications must be able to encode behavioural goals with respect to these values. It is possible to treat this as a reactive task, where change in fuel level is viewed as a change in the environment to which the high-level controller must react. However synthesis of reactive controllers generally is computationally expensive [74], limiting its potential for online execution.

Designing low-level controllers is also challenging in this case because UAV dynamics depend on the gross weight, which decreases as fuel is burned (for non-electrically powered UAVs). Further, the behaviour of the UAV strongly depends on wind conditions such as tail winds and head winds. This optimal control problem, known as Zermelo’s problem, is a two-point boundary value problem typically solved numerically using shooting methods.

1.3 Safety Constraints

Safety is another important type of constraint considered in this thesis. Mobile robots navigating in outdoor terrain must consider various forms and sources of uncertainty that arise from incomplete knowledge of the environment, and non-deterministic in-
1.3 Safety Constraints

Interactions between the environment and the robot. In outdoor application areas such as agriculture [34], robots must traverse terrain with hazards that generally remain constant over time (e.g., steep slopes and static obstacles) and hazards that change over time (e.g., pools of standing water). Sloping terrain can complicate control due to wheel slip, and muddy areas can cause the robot to become stuck. Solving motion planning problems in such environments involves an inherent balance between risk and reward, where risky choices can lead to shorter paths at the expense of a greater chance of mission failure.

Previous work has proposed and validated sophisticated statistical methods for high quality mobility prediction given a prior map, known as a digital elevation map, that represents the slope of the terrain and obstacles within it [44]. However, principled methods for exploiting this high quality mobility prediction have not been fully explored. This mobility prediction can be viewed as a transition function. Typical motion planners balance risk and reward through a weighted cost function that encourages progress towards the goal while discouraging actions that could lead to collision or tipping over [59]. Tuning the weights of the cost function must be done manually, usually by inspecting sample paths.

The cost function approach is an indirect means of achieving the desired intent, which is to find safe paths. It would be useful to instead have a formal probabilistic measure of safety, or mission success, that informs path planning. Formal verification with temporal logic is a powerful tool that can provide probabilistic performance guarantees for stochastic systems. A temporal logic called probabilistic computation tree logic (PCTL) has been used to define temporal mission specifications from which optimal control policies can be synthesised automatically [55]. PCTL is based on the Markov decision process (MDP) formalism, but PCTL allows for symbolic task descriptions that would otherwise be manually encoded in the MDP reward structure. Unfortunately, existing model checking and synthesis methods for PCTL-specified tasks cannot be directly applied to risk/reward balanced motion planning because they maximise the probability of success, as opposed to path length with respect to a safety threshold. The challenge is to develop model checking and synthesis algorithms
for this bi-criteria objective.

1.4 Approaches to Outdoor Task Planning with Constraints

This section presents approaches to problems with resource and safety constraints. The resource part considers resource threshold constraints and fuel constraints in wind and proposes an extension to an existing logic and efficient synthesis algorithms. The safety part is based on a mathematical proof showing that safety and completion time are analytically related.

1.4.1 Resource Constraints

Resource Threshold Constraints

Existing forms of temporal logic are unable to formally express resource threshold constraints. Therefore the related model-checking and synthesis algorithms cannot address such critical properties in practice, and the lack of expressivity often leads to degradation of performance. Our approach to the problems is to extend PCTL to admit resource threshold constraints in continuous form. The extended logic resource threshold-PCTL (RT-PCTL) is presented which is not only able to formally represent high-level symbolic specifications, but also a constraint on an accumulated continuous-valued resource.

A piecewise-constant control policy is defined to specify control actions depending on the value of the accumulated resource at the time a state is entered. A piecewise-constant probability function (PPF) is also defined to represent the probability of mission success in a given state with respect to the value of the accumulated resource. A set of PPFs, one for each state, represents the formal performance guarantee for a given control policy over all possible paths. Algorithms for model-checking and synthesis are defined for RT-PCTL.
Fuel Constraints in Wind

Problems with fuel constraints in wind are addressed by presenting efficient algorithms for the synthesis of correct task-level behaviour from linear temporal logic (LTL) formulas for a UAV. A reactive task-level controller is coupled to a low-level flight controller through operational state variables. The operational state of the robot is modelled in continuous form in the flight control layer, and also represented symbolically in the task layer. The task layer reacts to changes in operational state, such as if the fuel level drops below a certain value, in a way that satisfies the given LTL task specification.

Reactive task-level synthesis is performed using a Büchi automaton [50], but not by constructing a product of automata as is typical. This approach drastically improves the efficiency of synthesis for the purpose of enabling online execution during flight. The main limitation of this approach is that efficiency gain comes at the cost of completeness. However, correctness at the task level is preserved.

The flight controller plans a path for the robot given wind velocity predictions interpolated from point estimates using Gaussian process regression. Change in gross weight of the robot due to fuel burn over time is modelled analytically using the well-known Breguet range equation. UAV dynamics are modelled using a set of non-linear differential equations and solved numerically.

1.4.2 Safety Constraints

A robot is often required to consider more than one mission requirement. One of the important bi-criteria cases would be safety and completion time. In this case, an analytical relationship between the safety and completion time must be studied. For this problem, the approach is to prove a mathematical relationship between a formal satisfaction guarantee and expected conditional completion time, given a PCTL specification. Based on this analytical relationship, a model-checking algorithm is presented where a given control policy is model-checked against the specification for
safety and completion time. The policy is first model-checked for safety, and the result is used to compute the completion time from the relationship. Furthermore, a synthesis algorithm is presented where the completion time is minimised while maintaining the given safety threshold. At every iteration of synthesis, an action for a state is chosen in such a way that the action with minimum completion time is chosen from the set of actions that guarantees safety above the given threshold.

1.4.3 Combinations of Constraints and Other Extensions

The approaches in this thesis aim to address the most important and basic problems that arise in task planning for outdoor robots. Looking further, there exist many problems with other types of constraints that are yet to be addressed. For example, due to sensor noise and uncertainty in the environment, resources are often modelled as a probability density function [21]. For this case, further study to extend the resource model is required. In some safety-related applications, it may be necessary to adjust the required level of safety during execution [55]. Also, in a multi-robot application, there could be communication constraints where a large number of robots are to share information [19]. Formal expression of the constraints would be necessary for better performance. Even more, an application may require combinations of these constraints in a unified framework. The framework should have a special form of temporal logic that is capable of expressing the different types of constraints in a single form. The framework should be able to synthesise a provably-correct controller in a push-button manner.

The algorithms presented in this thesis are useful in isolation with respect to their individual problem domains. However, the intention is that these algorithmic approaches will be combined and extended in the future to further increase the capability of trusted outdoor robot systems. For instance, our current approaches with resource threshold and safety constraints can be naturally combined together in a unified framework, and we provide discussion of such combinations in this thesis. Also, RT-PCTL can be further extended to model resources with a probability den-
The main contribution of this thesis is to extend formal methods for application to autonomous outdoor robots with resource and safety constraints. For resource constraints, problems with resource threshold and fuel constraints in wind are addressed; for safety constraints, the problem of bi-criteria optimisation for safety and completion time is addressed. The work presented with resource constraints appears in part in [94, 95] and the work with safety constraints appears in part in [96]. Specific contributions are as follows:

- Continuous representation of accumulated resource and threshold constraints for task planning. The constraints in resource bounds are naturally expressed with formal language, and the specification can be used for model-checking or synthesising an optimal controller with respect to the probability of success. An existing logic, PCTL, is extended to RT-PCTL which is formally defined with syntax and semantics. Formal proofs are given for correctness and computational complexity.

- Efficient synthesis algorithms for task planning in the presence of external forces. The work presents a novel approach for online synthesis of complex missions with consideration of complex models of wind, aircraft dynamics and fuel consumption.

- Proof for the analytical relationship between safety and conditional completion time with respect to PCTL formulas. Algorithms for model-checking and synthesis are developed based on the proof. This work contributes to the optimisation of bi-criteria problems.
• Evaluation of the theoretical contributions with simulated examples. The significance of the theoretical work is demonstrated with a number of examples to present its applicability in practice. The models of various types of autonomous outdoor robots, such as thermal gliders, aircraft with solar panel, autonomous ground vehicles and petrol-powered aircraft, are used in realistic environment settings.

1.6 Thesis Outline

The remainder of this thesis is organised as follows:

Chapter 2 presents related work in the use of formal methods, the types of temporal logic, the types of constraints and the complexity of controller synthesis.

Chapter 3 provides necessary background material in the areas of formal methods and motion planning.

Chapter 4 presents RT-PCTL, model-checking and synthesis algorithms for resource threshold constraints.

Chapter 5 presents an efficient synthesis algorithm called greedy Büchi algorithm (GBA) with the use of realistic models of fuel consumption and wind.

Chapter 6 proves an analytical relationship between safety and completion time, and presents model-checking and synthesis time for the bi-criteria objective.

Chapter 7 presents results from simulations conducted from the theoretical work presented.

Chapter 8 summarises and concludes the thesis with a discussion of important future research directions.
Chapter 2

Related Work

Interest in the application of formal methods to robotics problems has grown steadily over the past decade. In this chapter, we discuss relevant related work in this area and also discuss other approaches that consider resource and safety constraints. In Section 2.1, we survey relevant existing applications of formal methods and two types of temporal logic used in the thesis. In Section 2.2, we present general uses of formal methods in robotics. In Section 2.3, we present literature related to task planning with resource and safety constraints. Lastly in Section 2.4, existing work on using hybrid systems and their limitations are presented. The chapter concludes with a summary in Section 2.5.

2.1 Formal Methods

Formal methods are mathematical techniques for specifying the required properties of a system and verifying that a model of the system satisfies these properties. Formal methods involve both formal specification and verification. One approach for formal specification is to use temporal logic. Temporal logic is a class of logic that extends propositional or predicate logic with temporal properties [3]. It has been used extensively in embedded systems to specify required system properties over all possible sequences of inputs that are not possible using traditional propositional logic. System
properties include functional correctness, liveness, safety, fairness, reachability and real-time properties [13, 15, 22, 64]. Unlike a classical propositional logic, temporal logic is capable of expressing a behaviour of a system over time. Given a temporal logic specification and a model of a system, we can formally check the model with respect to the specification. The formal process is called model-checking. The critical difference between model-checking and testing/simulation is that model-checking is capable of detecting the absence of error whereas testing/simulation is only able to detect the presence of error. Therefore the formalism plays an important role in safety-critical systems.

Figure 2.1 shows a simple tree of the temporal logic family. Various forms of temporal logic have been proposed, including LTL [73] and CTL [32]. Neither is defined to include stochastic transition models in their basic forms. For temporal logic to represent real-world environments with sensor noise and actuation error, PCTL [53] was introduced to replace the non-determinism of CTL with probabilities. LTL is another widely used form of temporal logic that is suitable for specifying linear time
properties [73]. There exist other variants such as probabilistic LTL that includes a probability operator [4]. Also, there exists a superset of CTL* called $\mu$-calculus that is used in motion planning [43]. However, our main interest in this thesis is in using LTL and PCTL.

2.1.1 Formal Methods in Digital Systems

Formal methods have been used extensively in the area of digital circuits, and begun to gain attention in the embedded systems domains. Formal methods are important mathematical techniques for safety-critical systems and currently are widely employed in embedded systems. At the circuit-level, formal methods are used in pipelined CPUs [15] and sequential circuits [16] for expressing the properties of interest using temporal logic and verifying that the electric circuits operate as specified. Formal methods are also used in the automatic verification of programmes [86], where the written code is checked against the desired behaviours.

There exist a number of applications in medical devices where guaranteeing operational correctness is critical for the safety of humans. Such properties are specified and verified for medical monitoring systems [22]. A pacemaker system has been developed that uses a hybrid automaton to verify its functional correctness with a formal model of the heart [18].

Formal methods are also used extensively for distributed systems [79, 77] for the synthesis of task schedulers and synchronisation of multiple clocks. In [47], a formal study of an intelligent transport system with a stochastic model is discussed, where the objective is to develop standard communication networks that guarantee the safety of the vehicles, the infrastructure, and the humans involved.

2.2 Formal Methods in Robotics

The application of temporal logic to robotics problems has recently become an important topic of interest to the robotics community [8, 48, 27, 14, 12, 33, 52]. The
focus has been on systems where temporal logic is used for high-level discrete mission planning complemented by low-level planners or controllers operating in continuous or execution space [11, 83]. Existing work has used temporal logic for motion planning in uncertain environments with probabilistic guarantees [28, 84], in partially-known environments [36, 66], in non-deterministic environments [83, 58] and with multiple agents communicating to achieve a global goal [19, 46, 45, 93, 81, 7, 26, 42, 62]. Since temporal logic specifies truth over an infinite execution of a mission, it naturally expresses surveillance tasks in which robots visit a number of locations infinitely often while satisfying other rules [20, 78, 25]. Temporal logic can also express information gathering tasks in which an agent gathers information while satisfying other given specifications [41].

2.2.1 Task Planning with PCTL

PCTL is a popular form of branching-time temporal logic used in robotics. It is an extension of CTL where CTL’s non-determinism is replaced with probabilistic uncertainty [5]. The semantics of PCTL is defined over a Markov chain (MC) or Markov decision process (MDP). The difference between the classical MDP and the PCTL frameworks is that the objective in PCTL is driven by a symbolic specification, whereas the objective in the classical MDP is indirectly assigned by tuning a cost function.

With the expressivity of uncertainty, PCTL has been widely used in robotic applications with uncertainty or noise in robots or environments. For instance, PCTL has been used to synthesise optimal controllers with respect to the probability of mission satisfaction for a given PCTL formula [55, 56]. This work assumes a known transition uncertainty of robots in fully observable environments (i.e., no noise in observation). Such a framework is naturally extended to the cases with limited observability of environments such as partially-observable MDPs (POMDPs) and mixed-observability MDPs (MOMDPs) [68] where robots do not have perfect sensing and have limited knowledge of environments. Like in the fully observable environments, the objective
is also to find an optimal control policy that maximises the probability of satisfying a given specification. This thesis, instead, focuses on bi-criteria objectives where satisfaction probability and completion time are considered in a single framework.

PCTL’s expressive power has been used for system analysis in some existing work. The work in [39] analyses and reasons about the effects of sensor error on a satisfaction of a given PCTL formula. The proposed methods synthesise an optimal controller assuming perfect sensing and actuation, and then diagnose the effect on the mission success probabilistically with increasing level of error in sensing and actuation.

As PCTL is based on MC or MDP formulations, a reward structure is naturally included in the framework [53]. PCTL can also specify a requirement on expected reward. For example, a PCTL framework can return the expected amount of resource consumption [5] and use the results for model-checking. However, the reward structure in PCTL only considers the expected value at the end of the time horizon and thus is not suitable for a mission where success depends not only on the symbolic mission specification but also on the accumulated reward along the path. This thesis addresses realistic constraints on resources directly.

There exist continuous-time frameworks for PCTL [5, 67], where the transition uncertainty is modelled with an exponential distribution. Continuous-time PCTL is widely used in telecommunication and biology, where error rate in data transmission and evolution rate could be modelled with an exponential function [66]. However, since this model of transition does not reflect the behaviour of robots in general, it is yet to be used in robotics in practice.

### 2.2.2 Task Planning with LTL

LTL is a widely used form of temporal logic that specifies linear time properties [73]. Since the time semantics is linear, there exists a single successor at each moment in time in LTL (i.e., quantification branching is not allowed). In contrast, the time semantics in PCTL allows for branching [3]. Therefore, LTL can be used to reason about the truth of a linear sequence of events. Given temporal operators and Boolean
variables, a specification can be made and a system model can be verified or model-checked against the specification. The model-checking process can also be used to synthesise a set of control actions that satisfies the specification in a system.

**LTL** is useful for a system where tasks are to be satisfied over an infinite time horizon. Hence **LTL** is used very extensively in robotics for surveillance and persistent monitoring tasks, where a robot is required to visit regions of interest while complying with other rules over a long period of time repeatedly [61, 78, 20, 25, 50, 87]. Such tasks could be made far more complex than simple navigation tasks by combining and nesting more than one formula. The idea of persistent monitoring has also been extended to information gathering missions, where the goal of the robot is to maximise its information gain while complying with the rules encoded with **LTL** [41].

To specify a system property over a finite time horizon, a fragment of **LTL** called **syntactically co-safe LTL** could be used in which the satisfaction is in finite time [12].

The **LTL** synthesis problem is often formulated using an MDP where system transitions are probabilistic. The main focus in such a problem is to maximise the probability of satisfying a given formula [30]. For a partially observable environment, using a POMDP would describe constraints more precisely. However, synthesis of **LTL** formulas is undecidable for a POMDP [70, 76]. More precisely, the quantitative analysis problem, which is to ask if the probability of satisfying an objective is greater than a given threshold, is undecidable for probabilistic automata over finite words. However, for the practical case of finite-memory policies, the problem becomes decidable [17].

For faster and more efficient completion of tasks, the extension to the use of multiple robots has been studied [61, 35, 87, 6, 20]. The objective of the multi-robot approach is to achieve a goal that is hard to solve with a single robot, such as surveillance, and search and tracking [20]. The global goal is divided into sub-goals while avoiding collisions between the heterogeneous robots or all robots have the same goal.

The biggest challenge in using **LTL** in practice is the time complexity of synthesis. The time complexity is doubly-exponential for the general case [71, 75], but there does exist related work for synthesis of **LTL** formulas in practice. The fragment of
LTL known as generalised reactivity (1) (GR(1)) uses partial fragments of LTL and reactive controllers can be synthesised in polynomial time [71]. Automatic synthesis of control polices from LTL specifications has been proposed for various task planning and optimal control tasks in robotics. Pioneering work in reactive mission and motion planning with GR(1) formulas is presented in [50, 91, 20] and applied to single-robot and multi-robot scenarios. The restriction to GR(1) formulas means that control policies can encode reactions to global world events while retaining polynomial-time computation. However, it is important to note that the whole system is expressed in a GR(1) formula (i.e., a formula contains the details about transition system as well as the task specifications). Therefore the size of the formula is usually huge compared to a formula specified using standard LTL. Along with the GR(1) fragment, a receding horizon framework has been proposed to reduce the size of the synthesis problem into smaller sub-problems [25, 92, 41]. The framework guarantees correctness with respect to the given specification. This thesis also presents a novel algorithm for reducing the problem size of synthesis for a given LTL specification.

2.3 Task Planning with Constraints

The expressive power of temporal logic is useful for specifying complex robotic tasks. However, the expressivity is limited to symbolic specifications only. This section identifies two important constraints in robotics, resource and safety constraints, and presents related work.

2.3.1 Resource Constraints

Often, resource structures for robots are indirectly represented using cost functions. The function represents the amount of utility gained/lost when entering a state, or transiting from a state to another. The objective involving the cost functions is to minimise the expected cost or to maximise the expected reward over a finite or infinite horizon [90, 82, 42, 87, 29]. For example, the objective for a robot could be
to minimise the expected travel time to reach the goal. The cost function is also used to control the behaviour of robots indirectly [38, 2, 62]. Suppose an environment has regions to visit and avoid; the regions to visit are given positive values and the regions to avoid are given negative values. In the formulation, robots are attracted toward positive-valued regions, and repelled away from negative-valued regions. The objective in the indirect control approach is to maximise the expected cost and find the corresponding control policy. In [58], using a cost function approach, an action at each node is chosen by finding a node with the least number of child nodes that do not lead to a goal.

There is recent work that provides more realistic models of resources in practice. In [61], the use of limited battery life with multiple robots is considered, where the objective is to satisfy the given mission specification for high-level vehicle routing. Robots with limited battery life can visit charging stations when required. In [88], resource constraints for active sensing and recognition of objects are considered, where the resources for motion, number of measurements and bandwidth are limited. The objective is to find optimal actions for accurate measurements. However, the models of resources are simple extensions from basic cost function approaches. The constraints associated with the types of resources cannot be naturally expressed as a part of the mission specification. This thesis presents a novel extension to an existing logic that enables the expression of complex resource constraints.

There also exists a temporal logic called signal temporal logic (STL) [31, 65]. The logic is defined over continuously valued signals, where the value of a signal at a given time is a real value. In STL, predicates are defined over a real-valued function of a signal. Therefore some extensions could be made to the existing form to express resource constraints with its quantitative semantics. However, since the form of logic only allows the signals to be deterministic, STL is not a suitable form for a system with stochastic transitions.
2.3 Task Planning with Constraints

2.3.2 Safety Constraints

The safety of a robot is often related to uncertainty in the environment and system. The result of a given input to a robot cannot be known in advance. Therefore, the safety of the robot is not guaranteed for control inputs that are considered to be safe in the deterministic case. These violations could lead to catastrophic failure of the robot platform and cause damages or injuries. In robotics, such uncertainty in the environment and system is modelled with a system with transition uncertainty where an action in a state could lead to a number of possible future states.

PCTL is an ideal temporal logic to describe such a system with uncertainty since it can naturally express the existence of multiple possible outputs for a given input. In robotics, PCTL has been a popular logic for specifying behaviour, model-checking a controller and synthesising an optimal controller with respect to the probability of success (i.e., safety) [55, 68]. Uncertainty in transition has also been introduced to LTL in [29] where the controller is synthesised from an LTL formula considering the probability of satisfaction.

The objective of synthesis for controllers operating in uncertain environments has been to maximise the probability of success. However, in many practical scenarios, it is also important to consider other requirements such as completion time. Those approaches tend to sacrifice other important factors for maximum safety. In [58], the bi-criteria objective is to increase the chance of reaching the goal within a fixed amount of completion time. Our work presents bi-criteria algorithms that balance between safety and completion time.

The idea of model-checking has been used to reason about the safety property of controllers with respect to the given specifications. The work presented in [40] considers controllers that are guaranteed to satisfy given PCTL specifications when assuming perfect sensing and actuation, and then model-checks the controllers to find the satisfaction probability for different levels of uncertainty in sensing and actuation. In [72], when a model-checking algorithm finds a counter-example for violation of given safety properties, the proposed algorithms synthesise an increasingly-useful trajectory
Figure 2.2 – An overview of a hybrid system for task planning that consists of two layers: high-level planner and low-level controller. The initial inputs to the high-level planner are the mission specification and knowledge of environment. The planner synthesises a discrete, graph-based solution, and passes the solution to the low-level controller. The low-level controller then executes the discrete solution in a continuous environment. The controller obtains continuous-valued sensing data from the environment, and it passes discrete feedback to the high-level planner.

Formulating a problem using an MDP considers transition uncertainty of a system. The objective of solving such a problem is to find an optimal solution with respect to only one property of interest. In [80], a policy that guarantees the minimum expected average cost between two consecutive tasks is found. Similarly in [30], a maximally satisfying control policy is found for a system with stochastic transitions based on an MDP formulation. Our work using PCTL is different to the existing approaches in such a way that we consider two constraints at the same time. We guarantee the minimum level of safety while maximising the probability of success.
2.4 Hybrid System Approach

Task planning in robotics is based on a top-down and hierarchical structure [8, 51]. Such a hybrid system approach consists of two layers of control: a high-level planner for symbolic tasks, and a low-level controller for execution in continuous state space, as shown in Figure 2.2 [11, 83, 38, 92]. The figure illustrates a typical model of a hybrid system used in the robotics community. The high-level planner initially receives a task specification and environment information to synthesise a discrete solution. The solution is given to the low-level controller to execute the solution in a continuous environment. Continuous sensing data is then received from the environment, and the necessary information is fed back to the planner symbolically.

For the ease of computation and theoretical limitations, physical models of the real world (e.g., position and time) are discretised with a given resolution and interval, and assumptions are made with respect to the models (e.g., transition probability and wind models). For instance, if space of an environment is discretised too coarsely, a synthesis algorithm for the task-level planner may not find a solution even there exists one. There exists important work on synthesising a provably-correct controller for both discrete abstraction of the robot and continuous dynamics [9, 51]. However, due to inherent complexity in guaranteeing bisimulation and reachability properties, finding such a controller is not scalable for outdoor environments. Therefore, in this thesis, the optimality and completeness of the proposed algorithms are subject to the models of the system. Hence, an optimal task-level solution with respect to a discrete abstraction may not guarantee optimality in continuous models.

The class of temporal logic to use in the high-level planner depends on the type of expressivity required and properties of the environment. For example, LTL would be suitable for synthesis of a reactive controller for surveillance missions [61, 78, 20, 25, 50, 87], and PCTL would be suitable for a slippery environment (i.e., transition uncertainty) [55, 56, 68]. The type of controller for low-level execution also depends on the models and assumptions of the environment. A simple model of dynamics (e.g., differential driving) has been used for execution in a number of existing ap-
plications [51, 38, 54], and some existing work has presented non-linear models of dynamics with guaranteed execution of the controller satisfying a high-level specification [23, 61, 12].

These approaches assume that there exist no external forces such as wind acting on the robots. Other work synthesises controllers that guarantee the execution of an LTL formula for a class of dynamical systems [23, 92], but the level of coupling between low-level operational and task-level states of the robot is limited. For instance, the operational states from the low level such as continuous-valued fuel is not considered in the high level. As a consequence, such important information is not considered. This thesis considers this coupled case with continuous execution under the influence of a continuous wind field assuming realistic dynamics and fuel models.

2.5 Summary

In summary, we introduced a general overview of formal methods and applications in embedded systems and robotics. We presented how PCTL and LTL are used to express complex robotic missions and to synthesise a controller. We also presented how resource and safety constraints are addressed using various types of temporal logic.

We have pointed out a number of significant limitations of using formal methods in robotics. Firstly, the current form of temporal logic is yet to formally express complex constraints on resources. Secondly, the time complexity of synthesis is intractable in general and also the models of systems and environments are too simple for real application. Lastly, formal methods cannot express bi-criteria objectives where two different types of properties should be considered simultaneously.

This thesis proposes algorithms that address the limitations and presents results with examples. It addresses the problems in three parts: problems with resource threshold constraints, fuel constraints in wind, and safety constraints.
Chapter 3

Background

In this chapter, we present necessary background material. In Section 3.1, we discuss the discrete-time Markov decision process (DT-MDP) and discrete-time Markov chain (DT-MC). In Section 3.2, we introduce the concepts of model-checking in formal methods. In Sections 3.3 and 3.4, we discuss the temporal logic used in this thesis: probabilistic computation tree logic (PCTL) and linear temporal logic (LTL). More details of the contents can be found in [5].

3.1 Discrete-Time Markov Decision Process (DT-MDP)

PCTL is based on the DT-MDP formalism. A labelled DT-MDP $\mathcal{M}$ is defined with a tuple $\langle S, A, P, R, AP, L \rangle$, where $S$ is a finite set of states, $A$ is the set of available actions, $P_{ss'}$ denotes the probability of transitioning from state $s$ to $s'$ in discrete time with action $a \in A(s)$, $R : S \to \mathbb{R}$ is a reward function, $AP$ is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function that assigns atomic propositions for each state.

A DT-MDP $\mathcal{M}$ is reduced to a discrete-time Markov chain (DT-MC) $\overline{\mathcal{M}}$ when an action is uniquely chosen for each state, such that $\pi : S \to A$. The DT-MC is defined
Figure 3.1 – A diagram of the model-checking approach. Requirement and system descriptions are formally specified using automata-based models, and the models are given to a model-checker. The model-checker examines all possible combinations of the requirement and system states to search for a violation of the requirements. When a violation is detected, the model-checker returns a counter-example. A counter-example is a trace of inputs that leads to the violation.

by a tuple \((S, P, R, AP, L)\) in which \(P : S \times S \to [0, 1]\) where \(\sum_{s' \in S} P(s, s') = 1, \forall s \in S\). Those states with \(P(s, s) = 1\) are called absorbing states.

3.2 Model-Checking

Model-checking is a verification technique that explores all possible system states in a brute-force manner [3]; given the formal models of requirements and system, it performs an exhaustive search to check if the given requirements hold true for the system model. Since model-checking systematically explores all possible combinations of the system states and input variables, it provides the absence of error as opposed to simulation or testing that only provides the presence of error. The diagram in Figure 3.1 illustrates an overview of model-checking.

Model-checking-based verification is a push-button technique whereby verification is performed automatically given formal models of the requirements and system. This allows the inputs of model-checking to be formalised very easily; hence the model-checking approach can be quickly implemented. This is in contrast to theorem proving, in which the verification on the given system is semi-automated.
When violation of a desired property is detected during model-checking, the model-checker terminates and returns ‘not satisfied’. Since model-checking explores all possible combinations of system states and inputs, the trace of inputs that caused the violation can be acquired explicitly. Such a trace of inputs is called a *counter-example* as shown in Figure 3.1 [5]. Counter-examples have a large number of applications and one notable example is debugging: the system designer can use the trace to understand which inputs caused the problem and diagnose which part of the system should be checked. A counter-example can also be used for synthesis of a controller. When the negated formal specification is given to the model-checker, the counter-example is the trace that satisfies the un-negated (i.e., original) specification.

### 3.3 Linear Temporal Logic (LTL)

LTL is an extension of propositional logic that expresses and reasons about the behaviour of systems over time [3]. The syntax of LTL is

\[
\varphi ::= \text{true} \mid p \mid \neg \varphi \mid \varphi \lor \varphi \mid \bigcirc \varphi \mid \varphi U \varphi,
\]

where $\varphi$ is an LTL formula over a set of atomic propositions $AP$, $p$ is an element of atomic propositions $AP$, $\neg \varphi$ is a negation of formula $\varphi$, $\varphi \lor \varphi$ is a disjunction of LTL formulas, $\bigcirc \varphi$ is a temporal model operator for the LTL formula $\varphi$ in the next state which is read as *next*, and $\varphi U \varphi_2$ is a temporal operator for the satisfaction of formula $\varphi_1$ until $\varphi_2$ is satisfied which is read as *until*. Note that by De Morgan’s laws, conjunction of LTL formulas $\varphi \land \varphi$ can be made with negation and disjunction, as well as implication ($\varphi_1 \Rightarrow \varphi_2$) and equivalence ($\varphi_1 \Leftrightarrow \varphi_2$). With the *until* operator, additional temporal operators such as *in future* and *always* can be derived: $\Diamond \varphi = \text{true} U \varphi$ and $\Box \varphi = \neg \Diamond \neg \varphi$ respectively.

If a run $\sigma$ satisfies a formula $\phi$, we represent the satisfaction as $(\sigma, i) \models \phi$ where $i \in \mathbb{N}$
Background

is a time instance. With the definitions, the semantics for LTL is

\[(\sigma, i) \models p \iff p \in \sigma[i]\]

\[(\sigma, i) \models \neg \varphi \iff (\sigma, i) \not\models \varphi\]

\[(\sigma, i) \models \varphi_1 \lor \varphi_2 \iff (\sigma, i) \models \varphi_1 \text{ or } (\sigma, i) \models \varphi_2\]

\[(\sigma, i) \models \bigodot \varphi \iff (\sigma, i + 1) \models \varphi\]

\[(\sigma, i) \models \varphi_1 U \varphi_2 \iff \exists k \geq i, (\sigma, k) \models \varphi_2 \text{ and } \forall j \in \{x \in \mathbb{N} \mid i \leq x < k\}, (\sigma, j) \models \varphi_1,\]

where \(\sigma[i]\) is an alphabet \((\in 2^{AP})\) at time \(i\).

LTL is used to express a variety of robotic tasks such as coverage, sequencing, conditions and avoidance. For example, \(\Diamond \text{room}_1 \land \Diamond \text{room}_2 \land \Diamond \text{room}_3\) denotes that \(\text{room}_1\), \(\text{room}_2\) and \(\text{room}_3\) are reachable in any order (coverage), \(\Diamond (\text{room}_1 \land \Diamond (\text{room}_2 \land \Diamond \text{room}_3))\) denotes that ‘\(\text{room}_2\) is reachable after \(\text{room}_1\) is reached, and then \(\text{room}_3\) is reachable’ (sequencing), \((\text{room}_1 \Rightarrow \bigodot \text{room}_2)\) denotes that \(\text{room}_2\) will be visited immediately if currently in \(\text{room}_1\), and \(\neg \text{danger} U \text{room}_1\) denotes that there is no danger until reaching \(\text{room}_1\). More complex missions can be expressed with nesting, conjunction/disjunction and negation of multiple LTL formulas as defined in the syntax. For example, a surveillance mission can be written as \(\square (\Diamond \text{room}_1 \land \Diamond \text{room}_2)\) which denotes \(\text{room}_1\) and \(\text{room}_2\) are always visited infinitely often.

### 3.3.1 Deterministic Büchi Automaton

From a given LTL formula over a set of atomic propositions, a Büchi automaton that accepts only the satisfying traces can be constructed. Note that there exist LTL formulas that do not admit deterministic Büchi automata, however all admit deterministic Rabin automata. In this thesis, our interest is in constructing and using deterministic Büchi automata.

A deterministic Büchi automaton \(B\) is a tuple \(\langle Q, q_0, \Sigma, \delta, F \rangle\), where \(Q\) is a finite set of states, \(q_0 \in Q\) is an initial state, \(\Sigma = 2^{AP}\) is a set of input alphabets defined over
a set of atomic propositions $AP$, $\delta : Q \times \Sigma \rightarrow Q$ is a deterministic transition relation and $F \subseteq Q$ is a set of accepting states.

In order to solve for the truth of an infinite sequence of states over an LTL formula, an equivalent Büchi automaton is built which accepts all and only the infinite sequences of words $\omega$ where $\omega_i \in \Sigma$ satisfies the given formula. An infinite sequence is said to be accepted by a Büchi automaton if and only if the accepting states are visited infinitely often.

A Büchi automaton for an LTL formula $\Box \Diamond a \land \Box \Diamond b$ is shown in Figure 3.2 where $q_1$ is an initial state and $q_2$ is an accepting state. Any word (or sequence) of infinite length that visits $q_2$ would be an accepting word. For example, a word $\omega = aabab...$ with $ab$ repeated is an accepting sequence of the formula since the accepting state $q_2$ is visited infinitely often.

### 3.3.2 Synthesis with LTL

There are a number of approaches for LTL synthesis depending on the types of determinism considered. For a deterministic transition system and a deterministic Büchi automaton, Dijkstra’s algorithm can be used to find a satisfying run. For a deterministic transition system and a non-deterministic Büchi automaton, we find a counterexample from a model-checker. For a non-deterministic transition system and a deterministic Büchi automaton, a game-theoretic approach can be used to solve (i.e., playing a Büchi game). In this section, we present the case for a deterministic
transition system and Büchi automaton using Dijkstra’s algorithm.

A controller is synthesised from an LTL formula over a set of atomic propositions by constructing a product automaton. A product automaton is the product of a transition system and a Büchi automaton with a tuple

\[ \mathcal{P} = \mathcal{T} \otimes \mathcal{B} \]

\[ = (S \times Q, (s_0, q_0), \mathcal{A}, AP, \Sigma, \delta, F), \]

where \( S \times Q \) is the set of product states, \((s_0, q_0)\) is the initial state for the product, \( \mathcal{A} \) is the discrete set of actions at a state, \( \delta : S \times Q \times \mathcal{A} \rightarrow S \times Q \) is a deterministic transition relation and \( F = \{(s, q) \in S \times Q \mid q \in F_Q\} \) is the set of accepting states.

Suppose we have a transition system in Figure 3.3 and the Büchi automaton in Figure 3.2. The deterministic transition system has 9 states representing the position of the robot. State \( s_1 \) and \( s_9 \) are labelled with \( a \) and \( b \) respectively and \( s_3 \) is the initial state. The product automaton is shown in Figure 3.4, where the initial state is \((3,1)\) and the accepting states are \((2,2)\), \((6,2)\) and \((9,2)\). Note that \((9,2)\) is the only infinitely reachable accepting state. Different colours of states represent different states in the Büchi automaton. An action causing a transition in the product automaton is labelled on each edge.
The initial state for the automaton is $(3, 1)$ and the accepting states are $(2, 2)$, $(6, 2)$ and $(9, 2)$. Note that $(9, 2)$ is the only infinitely reachable accepting state. Edges are labelled with numbers representing actions to take. The product states with same Büchi states have the same colour. The objective is to find a sequence of actions visiting the accepting states infinitely often. One way is to use a graph search algorithm.

The objective of synthesis is to find a trace of actions over time that visits the accepting product states $(s, q) \in F$ infinitely often. One possible method is to use a graph search algorithm such as Dijkstra’s algorithm [24]. We find a sequence of actions that leads to one of the accepting states from an initial product state. Once an accepting state is reached, then we find another sequence that leads to the accepting state cyclically. Using this method, one of the accepting states is visited infinitely often with the two sequences, usually referred to as prefix and suffix. Alternatively, these terms can also be referred to as transient and steady-state sequences. Often, edges of the automaton are labelled with values that represent transition cost such as distance and energy spent between two states. With these costs, the aim is then to find a control sequence that minimises the average sum of the costs over an infinite time horizon, or that minimises the sum over the suffix. If we had a deterministic transition system and a non-deterministic Büchi automaton, we can synthesise a controller by finding counter-examples [50]. Note that the construction of a Büchi automaton is exponential in the size of the formula. Since the product automaton is a cross product of the Büchi automaton and transition system, the synthesis complexity mainly comes from
the construction of the Büchi automaton.

### 3.4 Probabilistic Computation Tree Logic (PCTL)

Probabilistic computation tree logic is a temporal logic extended from computation tree logic (CTL) which replaces non-determinism with uncertainty. In CTL, the model-checking of path formulas is done by evaluating the existence of either all or some paths that satisfy the given specification. Using PCTL, specifications with a probabilistic measure of satisfaction can be expressed through qualitative reasoning.

The syntax for PCTL is defined as

\[
\Phi ::= \text{true} \mid a \mid \neg \Phi \mid \Phi \land \Phi \mid P_{\preceq \lambda}[\phi] \\
\phi ::= X\Phi \mid \Phi U \leq k \Phi,
\]

where \( \Phi \) is a state formula over a set of atomic propositions, \( \phi \) is a path formula, \( a \) is an atomic proposition, \( \preceq \) represents probabilistic inequality (i.e., \( \preceq \in \{<,\geq,\leq,>\} \)), \( \lambda \in [0,1] \) is the probability threshold and \( k \in \mathbb{N} \cup \{\infty\} \) is a time limit. Atomic propositions \( AP \) are the symbolic properties of interest in each state. Note that \( U \) is an unbounded ‘until’ operator such that \( U \equiv U \leq \infty \). The satisfaction relation for any state \( s \in S \) is defined by

\[
s \models \text{true}, \forall s \in S \\
s \models a \iff a \in L(s) \\
s \models \neg \Phi \iff s \not\models \Phi \\
s \models \Phi_1 \land \Phi_2 \iff s \models \Phi_1 \land s \models \Phi_2 \\
s \models P_{\preceq \lambda}[\phi] \iff \text{Prob}(s,\phi) \preceq \lambda,
\]

where \( \text{Prob} \) specifies the probability of satisfying the given path formula \( \phi \) at state \( s \).
The satisfaction relation for any path formula is defined by

\[ \omega \models X \Phi \iff \omega[1] \models \Phi \]
\[ \omega \models \Phi_1 U^{\leq k} \Phi_2 \iff \omega[i] \models \Phi_2 \land \omega[j] \models \Phi_1, \quad (3.6) \]
\[ \exists 0 \leq i \leq k, \forall 0 \leq j < i, \]

where \( \omega \in \text{Path}(s) \) is a path with a sequence of states and \( \text{Path}(s) \) is a set of all finite and infinite paths starting from state \( s \). In this notation, we have \( \omega[i] \) for the \( i \)-th state in the sequence with \( \omega[0] = s \in S \). Note that \( \omega \) used in PCTL is for a path whereas that used in LTL is an infinite sequence of words.

A set of PCTL operators consist of boolean and temporal operators. Boolean operators are the operators from propositional logic such as \( \neg \) and \( \land \), representing negation (‘not’) and conjunction (‘and’), respectively. Temporal operators are used to express path formulas including ‘\( \neq X \)’ and ‘\( U \)ntil’. Additional operators could be made from the existing operators, as shown below:

\[ \text{false} \equiv \neg \text{true} \]
\[ \Phi_1 \lor \Phi_2 \equiv \neg(\neg \Phi_1 \land \neg \Phi_2) \]
\[ \Phi_1 \Rightarrow \Phi_2 \equiv \neg \Phi_1 \lor \Phi_2 \]
\[ F^{\leq k} \Phi \equiv \text{true} U^{\leq k} \Phi \]
\[ G^{\leq k} \Phi \equiv \neg F^{\leq k} \neg \Phi, \quad (3.7) \]

where \( \lor \) is disjunction (‘or’), \( \Rightarrow \) is implication (‘if then’), \( F \) and \( G \) are temporal operators for ‘sometime in ‘\( F \)uture’ and ‘\( G \)lobally’ (or ‘always’) respectively. Since the negation of a path formula is not permitted, we denote \( P_{\lambda}[G \Phi] \) as \( P_{\lambda - \lambda}[F \neg \Phi] \) where \( \preceq \equiv \succeq, \preceq \equiv \leq, \preceq \equiv \gg \) and \( \succeq \equiv \ll \). The ‘\( \neq X \)’ operator can be defined using \( \bigcirc \), ‘\( G \)lobally’ using \( \Box \), and ‘\( F \)uture’ using \( \Diamond \) as they are used in LTL. Although each symbol is identical to the other pair, we use such different notations to distinguish between LTL and PCTL formulas.
3.4.1 Model-Checking with PCTL

In PCTL, the output of a model-checking algorithm over a DT-MC is the set of states satisfying the given state formula \( \Phi \). The function ‘Sat’ returns the set of satisfying states for the input formula of the form shown below:

\[
\begin{align*}
\text{Sat}(s) &= S \\
\text{Sat}(a) &= \{ s \mid a \in L(s) \} \\
\text{Sat}(\neg \Phi) &= S \setminus \text{Sat}(\Phi) \\
\text{Sat}(\Phi_1 \land \Phi_2) &= \text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2) \\
\text{Sat}(\mathcal{P} \mathcal{w} \lambda[\phi]) &= \{ s \in S \mid \text{Prob}(s, \phi) \succcurlyeq \lambda \}.
\end{align*}
\]

The probability function \( \text{Prob} \) is calculated by value iteration. For ‘until’ \( U \) and ‘next’ \( X \) operators, the value iteration equations are shown below

\[
\begin{align*}
\text{Prob}(s, X\phi) &= \sum_{s' \in \text{Sat}(\Phi)} P_{ss'} \\
\text{Prob}(s, \Phi_1 U \leq k \Phi_2) &= \\
&= \begin{cases} 
1 & \text{if } s \in \text{Sat}(\Phi_2) \\
0 & \text{else if } k = 0 \text{ or } s \in \text{Sat}(\neg \Phi_1 \land \neg \Phi_2) \\
\sum_{s' \in S} P_{ss'} \cdot \text{Prob}(s', \Phi_1 U \leq k-1 \Phi_2) & \text{otherwise}
\end{cases}
\end{align*}
\]

where \( P_{ss'} \) is the transition probability from \( s \) to \( s' \). Note that \( \text{Prob} \) converges to a unique solution for \( k = \infty \).

3.4.2 Synthesis with PCTL

An optimal controller is synthesised based on a Markov decision process (MDP) formulation, with respect to the probability of satisfying a given PCTL specification over a set of atomic propositions. The synthesis based on the formulation is to find
an action for each state that maximises the probability of success. For path formulas, we have the following MDP formulations:

\[
\begin{align*}
\Prob^*(s, \mathcal{X}\Phi) &= \max_{a \in A(s)} \sum_{s' \in \text{Sat}(\Phi)} P^a_{ss'} \\
\Prob^*(s, \Phi_1 \mathcal{U} \leq k \Phi_2) &= \begin{cases} 
1 & \text{if } s \in \text{Sat}(\Phi_2) \\
0 & \text{else if } k = 0 \text{ or } s \in \text{Sat}(\neg \Phi_1 \land \neg \Phi_2), \\
\max_{a \in A(s)} \sum_{s' \in S} P^a_{ss'} \cdot \Prob^*(s', \Phi_1 \mathcal{U} \leq k-1 \Phi_2) & \text{else} 
\end{cases}
\end{align*}
\]

(3.10)

where \( A(s) \) is the set of actions for state \( s \) and \( P^a_{ss'} \) is the transition uncertainty from \( s \) to \( s' \) with action \( a \in A(s) \). For a path formula \( \phi \), the optimal control policy with respect to the satisfaction probability is \( \pi^*(s) = \arg \max_{a \in A(s)} \Prob^*(s, \phi) \). Problems formulated in this MDP formulation can be solved with value iteration.
Chapter 4

Task Planning with Resource Threshold Constraints

In this chapter, we focus on the problem of resource threshold constraints, which is the first instance of resource constraints considered in this thesis. Limitations of existing temporal logic are addressed and algorithms are proposed with formal analysis for a newly extended logic. In Section 4.1, we present a problem definition and a motivating example. In Section 4.2, we define the discrete-time resource-dependent Markov decision process (DTR-MDP). We then extend an existing temporal logic, probabilistic computation tree logic (PCTL), to resource threshold-PCTL (RT-PCTL) in Section 4.3, and provide model-checking and synthesis algorithms with respect to the probability of satisfying a given RT-PCTL formula in Sections 4.4 and 4.5. Section 4.6 summarises the chapter.

4.1 Resource Threshold Constraints

The objective is to build a formal framework for: 1) expressing a mission with the constraints, 2) model-checking, and 3) synthesising a controller satisfying the specification. The essential part of the framework is an extended form of temporal logic that is capable of formally specifying resource threshold constraints. The framework
is required to formally express complex and possibly non-intuitive threshold requirements for resources. The requirements are to be expressed with formal language that is also capable of expressing symbolic tasks. A mission specification with formally defined threshold requirements is to be model-checked for satisfaction of the mission. Such satisfaction depends on both symbolic task and threshold requirements. Furthermore, the framework should be capable of synthesising an optimal controller for a given specification with respect to the probability of success. Again, the controller should comply with the threshold requirements.

### 4.1.1 Motivating Example

To illustrate the need for resource threshold constraints, a simple environment with three states is given in Figure 4.1a. The agent can move left or right with probability 0.8 of moving as intended, and probability 0.2 of self-transition. Attempting to move past the left or right boundary always results in self-transition. The agent starts in state $s_1$ with entering resource value 0 and attempts to reach goal state $s_3$ while maintaining the value of the accumulated resource within the bound $h = [0, 5]$. For example, when the agent with no resource starts in state $s_1$, it gains the resource of 1.21. Therefore the accumulated resource becomes 1.21. When the agent visits state $s_2$, it loses the resource (i.e., $-2.16$). Hence the accumulated resource becomes $-0.95$. Note that the transition is stochastic in that the next state given a state and an action is not known in advance. An *initial accumulated resource* is the accumulated resource just before entering a state and a *final accumulated resource* is the accumulated resource after the resource gain or loss. The objective of the agent (i.e., reaching the goal state) can be expressed using PCTL. Since the semantics of PCTL is defined over a discrete-time Markov decision process (DT-MDP), we can consider this example as a problem of solving a DT-MDP.

Formulating this example in PCTL [53], it is possible to compute the probability of satisfaction of a property over an indefinite number of paths with no consideration of resource bounds, shown in Figure 4.1b. The probability is computed to be 0.9728 af-
4.1 Resource Threshold Constraints

![Diagram showing computation trees with and without resource constraints.](image)

(a) Simple environment

(b) Probabilistic computation tree without resource constraints

(c) Probabilistic computation tree with resource constraints

**Figure 4.1** – A simple example environment and computation trees. An example environment where the values represent the resource gained/lost when entering the states (a) and associated computation trees without (b) and with (c) resource constraints.
ter four time steps. However, the resource structure in standard PCTL is only able to consider ‘expected accumulated resource after $k$ time steps’, ‘expected instantaneous state resource at $k$ time steps’, and ‘expected accumulated state resource before satisfying a formula over a set of atomic propositions.’ Since these properties all compute the expectation of the resource at or after a certain number of time steps, the standard PCTL formulation is unable to determine if the accumulated resource within a path ever violates the bounds. As a result, the computation tree keeps branching from the state within the path that already went below threshold. Note that although the final resource at the end is above the threshold, the mission is considered to be unsuccessful if any state within the path does not satisfy the constraint.

The computation tree with a threshold constraint (Figure 4.1c) shows that branching terminates at a state when the accumulated resource goes below zero. The probability of success in this case is 0.1536. From Figure 4.1c it is shown intuitively that the successful path within four time steps is the one in which the agent stays in state $s_1$ for two time steps. Hence there is a need for a control policy structure and evaluation function that depend on the accumulated resource. We define such an evaluation function in Section 4.4.

4.2 Discrete Time Resource-Dependent MDP (DTR-MDP)

For the development of the framework, we start by defining a mathematical model of the resource threshold constraints we consider. Similar to the way that the semantics of PCTL is defined over DT-MDP, the semantics of RT-PCTL is defined over DTR-MDP.

We extend a labelled DT-MDP to a labelled DTR-MDP that is defined by a tuple $(S, s_0, X, x_0, A, P, r, h, L, AP)$ with a finite set of states $S$, an initial state $s_0 \in S$, resource space $X \subseteq \mathbb{R}$, an initial accumulated resource $x_0 \in X$, a finite set of available actions $A$ and a set of resource-dependent transition probability functions $P : S \times
4.2 Discrete Time Resource-Dependent MDP (DTR-MDP)

\[ X \times A \times S \rightarrow [0,1] \]  
where \( \sum_{s' \in S} P_{ss'}^a(x) = 1, \forall s, s' \in S, \forall a \in A(s), \forall x \in X \). More precisely, \( P_{ss'}^a(x) = \mathbb{P}(s_{t+1} = s' | s_t = s, x_t = x, a_t = a) \). More details will be shown in Section 4.4.2. Note that the value \( x \in X \) represents the accumulated resource. The function \( r \) returns instantaneous and transition resource gains for a state and between states respectively. Scalar \( r_s \in X \) is an instantaneous resource gained when entering state \( s \), and \( r_{ss'} \in X \) is a transition resource gained while transiting from state \( s \) to state \( s' \). When there is a transition from state \( s \) with accumulated resource \( x \) to another state \( s' \), the updated accumulated resource is \( (x + r_s + r_{ss'}) \).

\( h : S \rightarrow \mathbb{R}^2 \) represents a set of resource bounds for each state after completion of a transition. A parenthesis in the interval expression represents a saturation limit, and a square bracket represent a hard limit. For example, \( h = [h^l, h^u] \) denotes that the mission fails (violates a hard limit) when the accumulated resource is below \( h^l \), and that the accumulated resource saturates at \( h^u \). We use \( h_s \) for state-specific resource bounds for state \( s \in S \) where \( h_s^u \) is an upper bound and \( h_s^l \) is a lower bound. We also define that \( (h_s - r = [h_s^l - r, h_s^u - r]) \) where the brackets are replaced with those from \( h_s \). A typical case where such limits apply would be a vehicle with fuel constraints, where the vehicle is immobilised when it runs out of fuel and is unable to have more fuel than its capacity. Function \( L : S \rightarrow 2^{AP} \) is a labelling function that assigns atomic propositions \( AP \) for each state \( s \in S \). The main difference between DT-MDP and DTR-MDP is that the the transition probability function depends on both the accumulated continuous-valued resource and the action.

The DTR-MDP is reduced by a control policy \( \pi \) to a labelled discrete-time resource-dependent Markov chain (DTR-MC) which is defined by a tuple \( \langle S, s_0, X, x_0, P, R, h, L, AP \rangle \). The control policy is defined as \( \pi : S \times X \rightarrow A \) and transition probability is defined as \( P : S \times X \times S \rightarrow [0,1] \).
4.3 Resource Threshold-PCTL (RT-PCTL)

The syntax for RT-PCTL is defined as

$$\Phi ::= \text{true} \mid a \mid \neg \Phi \mid \Phi_1 \land \Phi_2 \mid P^{x:h}_{\in\lambda} [\phi]$$

$$\phi ::= \mathcal{X} \Phi \mid \Phi_1 U \leq k \Phi_2$$

where $\Phi$ is a state formula,

$\phi$ is a path formula,

$a$ is an atomic proposition,

$\mathcal{X} \in \{<, \leq, =, \geq, >\}$,

$x \in \mathbb{R}$ and $\lambda \in [0, 1]$ and $k \in \mathbb{N}$,

(4.1)

where the symbols and operators are similar to those of PCTL in Chapter 3.4. Likewise, derivations of logical and temporal operators are similar.

We define the $P$-operator, $P^{x:h}_{\in\lambda} [\phi]$ that represents the truth of a path formula $\phi$ over a set of atomic propositions with respect to the probability inequality $[\mathcal{X} \lambda]$, initial resource $x$, and resource bounds $h$. $Prob_s(x, h, \phi)$ indicates the probability of satisfying the path formula $\phi$ with initial resource $x$ over the set of paths $\text{Path}(s)$ while maintaining the accumulated resource within the bounds. Note that $x$ is the accumulated resource before transition. The short form of the function is $Prob^k_s(x)$ for a given $\phi$, time step $k$ where $Prob_s(x)$ is the probability function with $k = \infty$. The resource bound in the short form is $h_s$ which represents the bound for the state $s$.

Section 4.4 will describe an analytical solution to calculate the probability functions.

The satisfaction relation $|$ is defined for state formulas over a set of atomic propo-
sitions by:

\[ s \models a \iff a \in L(s) \]
\[ s \models \neg \Phi \iff s \not\models \Phi \]
\[ s \models \Phi_1 \land \Phi_2 \iff s \models \Phi_1 \land s \models \Phi_2 \] (4.2)
\[ s \models P^{x,h}_{s,\lambda}[^φ] \iff \text{Prob}_s^k(x) \not\equiv \lambda, \forall x \not\equiv x \text{ and } x \in h_s - r_s \]
\[ s \models P^{x,h}_{s,\lambda}[^φ] \iff \text{Prob}_s^k(x) \not\equiv \lambda, \forall x \in h_s - r_s. \]

Given a path \( \omega \) in \( \mathcal{M} \), the satisfaction relation is defined:

\[ \omega \models X\Phi \iff \omega[1] \models \Phi \]
\[ \omega \models \Phi_1 U^k \Phi_2 \iff \omega[i] \models \Phi_2 \land \omega[j] \models \Phi_1, \]
\[ \exists 0 \leq i \leq k, \forall 0 \leq j < i. \] (4.3)

Two examples are shown below:

- \( P^{=2;[0,10]}_1[X\text{danger}] \) : ‘the probability of reaching the danger state in the next time step is less than 0.05. The initial accumulated resource is 2 and the accumulated resource should be between 0 and 10 with upper saturation constraint’.

- \( P^{>0.8}_2[P^{>0.9}_1[F e U^{<200} \text{goal}]] \) : ‘The probability of reaching e with any resource between 0 and 100 should be greater than 0.9 until goal is reached. The mission has to be accomplished within a 200-step time horizon with resource bounds between 0 and 100, and the probability of satisfying the mission is greater than 0.8’.

### 4.3.1 Definition of Piecewise-Constant Function

This section defines the properties and operations on piecewise-constant functions used throughout the section. Some of the operations are standard. Note that the function is defined using ‘if − elseif’ statements where the upper statement is treated before the following. We use the short form ‘elif’ instead of ‘else if.’
Suppose $f(x)$ is a piecewise-constant function,

$$
f(x) = \begin{cases} 
    p_1' \text{ if } x > c_1' \\
    \vdots \\
    p_k' \text{ if } x > c_k' \\
    \vdots \\
    p_n' \text{ if } x > c_n' \\
    0 \text{ else}
\end{cases},
$$

where $k \in \mathcal{N}$ and $c_{k+1}' < c_k', \forall k \in \mathcal{N}$.

**Addition**

$$
f(x) + g(x) = \begin{cases} 
    \vdots \\
    f(c) + g(c) \text{ if } x > c \\
    \vdots \\
    0 \text{ else}
\end{cases}
$$

where $c \in (c' \cup c^g)$

**Shift**

$$
f(x + c) = \begin{cases} 
    p_1 \text{ if } x > c_1 - c \\
    \vdots \\
    p_n \text{ if } x > c_n - c \\
    0 \text{ else}
\end{cases}
$$
4.4 Performance Evaluation of Control Policy

To evaluate a control policy, we analytically solve a piecewise-constant probability function (PPF) at each state. This function represents the probability of mission
success in a given state with respect to the value of the accumulated resource. In this section, we show how to compute PPFs with respect to quantifiers $X$ (next), $U$ (until), and $F$ (future) for cases with hard limits and saturation limits.

### 4.4.1 PPF Solutions for Quantifiers $X$, $U$, and $F$

The quantifier $X$ specifies a path property from a state where $[Xp]$ denotes ‘property $p$ holds in the next transition’. The solution for computing the PPF is shown in Equation 4.10 for a single-action control policy $\pi(s)$. Note that $\text{Prob}_s(x, h, X\Phi)$ is a piecewise-constant function defined in 4.3.1 that returns the probability of holding the property $\Phi$ in the next transition starting from the state $s$ with the entering accumulated resource of $x$ while maintaining the accumulated resource within $h_s = [h^l_s, h^u_s]$. $P^{\pi(s)}_{ss'}$ denotes the transition probability from $s$ to $s'$ for an action $\pi(s)$.

\[
\text{Prob}_s(x, h, X\Phi) = \begin{cases} 
0 & \text{if } x > h^u_s - r_s \\
\sum_{s' \in \text{Sat}(\Phi)} P^{\pi(s)}_{ss'} \cdot \text{Prob}^0_{s'}(x') & \text{elif } x > h^l_s - r_s \\
0 & \text{else}
\end{cases}
\tag{4.10}
\]

where $\text{Prob}^0_{s'}(x) = \begin{cases} 
0 & \text{if } x > h^u_{s'} - r_{s'} \\
1 & \text{elif } x > h^l_{s'} - r_{s'} \\
0 & \text{else}
\end{cases}$

where $x' = x + r_x + r_{ss'}$.

The quantifier $U$ specifies the satisfaction of a property $\Phi$ along the path until it ends with another property $\Psi$, formally written as $[\Phi U \Psi]$. The PPF is shown in Equation 4.11 for a single-action control policy $\pi(s)$. Note that the formula $[F^{\leq k}\Psi]$ is identical to $[\text{true}U^{\leq k}\Psi]$. 

4.4 Performance Evaluation of Control Policy

\[ \text{Prob}_s(x, h_s, \Phi U^{k+1} \Psi) \]
\[
= \begin{cases} 
0 & \text{if } x > h_s^u - r_s \\
\sum_{s' \in S} P_{ss'}^{\pi(s)} \cdot \text{Prob}_{s'}^k(x', \Phi U^{k} \Psi) & \text{elif } x > h_s^l - r_s, \forall s' \in \text{Sat}(\Phi \land \neg \Psi), \\
0 & \text{else}
\end{cases}
\]

\[ \text{Prob}_s(x, h_s, \Phi U^{k+1} \Psi) = \begin{cases} 
0 & \text{if } x > h_s^u - r_s \\
1 & \text{elif } x > h_s^l - r_s, \forall s \in \text{Sat}(\Psi), \\
0 & \text{else}
\end{cases} \quad (4.11) \]

\[ \text{Prob}_s(x, h_s, \Phi U^{k+1} \Psi) = 0, \forall s \in \text{Sat}(\neg \Phi \land \neg \Psi) \]

where \( x' = x + r_s + r_{ss'} \).

We also illustrate the case with resource saturation constraints. For example, \( h = [0, 10] \) denotes that the accumulated resource should not go below 0 and will saturate at 10. General equations for upper and lower-saturated constraints are shown in Equation 4.12.

\[ \text{Prob}_s^{k+1}(x) \]
\[
= \begin{cases} 
\sum_{s' \in S} P_{ss'}^{\pi(s)} \cdot \text{Prob}_{s'}^k(h_s^u + r_{ss'}) & \text{if } x > h_s^u - r_s \\
\cdots & \text{elif } x > h_s^l - r_s, \forall s' \in \text{Sat}(
eg \Phi \land \Psi), \\
\cdots & \text{else}
\end{cases} \quad (4.12) \]

\[ \begin{align*}
\text{Prob}_s^{k+1}(x) & = \sum_{s' \in S} P_{ss'}^{\pi(s)} \cdot \text{Prob}_{s'}^k(x + r_s + r_{ss'}) & \text{elif } x > c_s^j - r_s \\
& = \sum_{s' \in S} P_{ss'}^{\pi(s)} \cdot \text{Prob}_{s'}^k(h_s^l + r_{ss'}) & \text{else}
\end{align*} \]
4.4.2 Piecewise-Constant Control Policy

The choice of action at a given state $s$ in our formulation depends on the value of the accumulated resource $x$. We represent this as a piecewise-constant control policy $\pi(s, x)$:

$$
\pi(s, x) =
\begin{cases}
  a_1 & \text{if } x > c^1_s \\
  a_2 & \text{elif } x > c^2_s \\
  \vdots & \vdots \\
  a_{n-1} & \text{elif } x > c^{n-1}_s \\
  a_n & \text{else}
\end{cases}
$$

(4.13)

where $A(s)$ is a set of possible actions at state $s$ and $a_i \in A(s)$.

The PPF with respect to a piecewise control policy $\pi(s, x)$ is shown in Equation 4.14:

$$
Prob^{k+1}_s(x) =
\begin{cases}
  0 & \text{if } x > h^n_s - r_s \\
  \sum_{s' \in S} P^{\pi(s,x)}_{ss'} \cdot Prob^k_s(x + r_s + r_{ss'}) & \text{elif } x > c^1_s - r_s \\
  \vdots & \vdots \\
  \sum_{s' \in S} P^{\pi(s,x)}_{ss'} \cdot Prob^k_s(x + r_s + r_{ss'}) & \text{elif } x > c^{i+1}_s - r_s \\
  \vdots & \vdots \\
  \sum_{s' \in S} P^{\pi(s,x)}_{ss'} \cdot Prob^k_s(x + r_s + r_{ss'}) & \text{elif } x > c^i_s - r_s \\
  \sum_{s' \in S} P^{\pi(s,x)}_{ss'} \cdot Prob^k_s(x + r_s + r_{ss'}) & \text{elif } x > h^1_s - r_s \\
  0 & \text{else}
\end{cases}
$$

(4.14)

For upper or lower saturated cases, the top or the bottom lines are replaced with $Prob^{k+1}_s(h^n_s + r_{ss'})$ and $Prob^{k+1}_s(h^1_s + r_{ss'})$ respectively.

The algorithm for evaluating a PPF with respect to a piecewise control policy is shown...
Algorithm 4.1: Evaluation of PPF with piecewise control policy

**Input:** RT-PCTL path formula, control policy and environment settings

**Output:** set of PPFs for all states over $K$ time steps ($Prob$)

1. $Prob^0_s(x) \leftarrow 0, \forall s \in S \setminus S^{yes}$
2. $Prob^0_s(x) \leftarrow \begin{cases} 0 & \text{if } x > h_u^s - r_s \\ 1 & \text{elif } x > h_l^s - r_s, \forall s \in S^{yes} \\ 0 & \text{else} \end{cases}$
3. for $k = 1$ to $K$ do
4. for $s \in S$ do
5. $Prob^k_s(x) \leftarrow \begin{cases} 0 & \text{if } x > h_u^s - r_s \\ \sum_{s' \in S} P_{ss'}^{(s,x)} \cdot Prob^{k-1}_{s'}(x + r_s + r_{ss'}) & \text{elif } x > c^1_s - r_s \\ \vdots & \vdots \\ \sum_{s' \in S} P_{ss'}^{(s,x)} \cdot Prob^{k-1}_{s'}(x + r_s + r_{ss'}) & \text{elif } x > h_l^s - r_s \\ \sum_{s' \in S} P_{ss'}^{(s,x)} \cdot Prob^{k-1}_{s'}(x + r_s + r_{ss'}) & \text{else} \end{cases}$
6. if upper saturated then
7. $Prob^k_s(x) \leftarrow \begin{cases} \text{Prob}^k_s(h_u^s + r_{ss'}) & \text{if } x > h_u^s - r_s \\ \vdots \end{cases}$
8. end if
9. if lower saturated then
10. $Prob^k_s(x) \leftarrow \begin{cases} \vdots \end{cases}$
11. end if
12. end for
13. if $\max_{s \in S, x \in \mathbb{R}} (Prob^k_s(x) - Prob^{k-1}_s(x)) < \epsilon$ then
14. break
15. end if
16. end for
17. return $Prob$

In Algorithm 4.1. For $[\mathcal{X} p], S^{yes} = \text{Sat}(p), S^2 = S \setminus S^{yes}, K = 1$ and for $[p_1 \cup^K p_2], S^{yes} = \text{Sat}(p_2), S^{no} = \text{Sat}(\neg p_1 \land \neg p_2), S^2 = S \setminus (S^{yes} \cup S^{no})$. The value $a_k$ is the control action at $c^*_s$ where $c^*_s$ is a subdomain in control policy $\pi(s, x)$. 
4.4.3 Model-Checking an RT-PCTL State Formula

Like in PCTL, the output of a model-checking algorithm is the set of states satisfying the given state formula over a set of atomic propositions, as presented in Chapter 3. Similarly, the function ‘Sat’ returns the set of states satisfying the input state formula as shown below:

\[
\begin{align*}
\text{Sat}(s) &= S \\
\text{Sat}(a) &= \{ s \mid a \in L(s) \} \\
\text{Sat}(\neg \Phi) &= S \setminus \text{Sat}(\Phi) \\
\text{Sat}(\Phi_1 \land \Phi_2) &= \text{Sat}(\Phi_1) \cap \text{Sat}(\Phi_2) \\
\text{Sat}(\mathcal{P}_{x:h}^{\leq k}[\phi]) &= \{ s \in S \mid \text{Prob}_s^k(x) \equiv p, \forall x \equiv x \text{ and } x \in h - r_s \} \\
\text{Sat}(\mathcal{P}_{x:h}^{\leq 4}[\phi]) &= \{ s \in S \mid \text{Prob}_s^k(x) \equiv p, \forall x \in h - r_s \}.
\end{align*}
\]

(4.15)

4.4.4 Calculation Example

We illustrate the calculation of a PPF using the simple scenario shown earlier in Figure 4.1a. A purely state-based policy for this example is shown in Equation 4.16, where \( \text{Prob}_1(x, h, \mathcal{F}^{\leq 4}s_3) \) and \( \pi(s_1) \) are the PPF and policy at state \( s_1 \) within four time steps. \( \text{Prob}_1(x, h, \mathcal{F}^{\leq 4}s_3) \) represents the PPF of the logic specification to reach state \( s_3 \) with accumulated resource \( x \) within four time steps with resource constraint \( h \) at state \( s_1 \). The policy is simply to move right at states \( s_1 \) and \( s_2 \). Since we constrain the accumulated resource to be above zero and below five, the agent is likely to fail if it enters state \( s_2 \) with zero accumulated resource; transitioning from state \( s_1 \) to
state \( s_2 \) results in accumulated resource \(-0.95\).

\[
\pi(s_1) = \text{Right} \\
\pi(s_2) = \text{Right}
\]

\[
\begin{align*}
&0.0000 \quad \text{if } x > +3.79 \\
&0.7680 \quad \text{elif } x > +3.11 \\
&0.6400 \quad \text{elif } x > +2.58 \\
&0.7936 \quad \text{elif } x > +1.90 \\
&0.0000 \quad \text{if } x > +3.79 \\
&0.7680 \quad \text{elif } x > +3.11 \\
&0.6400 \quad \text{elif } x > +2.58 \\
&0.7936 \quad \text{elif } x > +1.90 \\
&0.1536 \quad \text{elif } x > -0.26 \\
&0.0256 \quad \text{elif } x > -1.21 \\
&0 \quad \text{else}
\end{align*}
\]

(4.16) \( Prob_1(x, h, F^{\leq 4}s_3) = \)

In contrast, evaluation of the piecewise control policy is shown in Equation 4.17, where the policy is also resource dependent (i.e., \( \pi(s, x) \)). Here, the action mapping varies with the initial accumulated resource and the agent attempts to go left until its accumulated resource exceeds \(+1.0\). With such a policy, the probability of mission success has jumped from 0.1536 to 0.768. Note that the resource values can be any real number.
\[ \pi(s_1, x) = \begin{cases} 
   \text{Right} & \text{if } x > +1.0 \\
   \text{Left} & \text{else}
\end{cases} \]

\[ \pi(s_2, x) = \text{Right}, \forall x \in \mathbb{R} \]

\[ \text{Prob}_1(x, h, \mathcal{P}^{\leq 4}s_3) = \begin{cases} 
   0.0000 & \text{if } x > +3.79 \\
   0.7680 & \text{elif } x > +3.11 \\
   0.6400 & \text{elif } x > +2.58 \\
   0.7936 & \text{elif } x > +1.90 \\
   0.7680 & \text{elif } x > +1.37 \\
   0.7936 & \text{elif } x > +1.00 \\
   0.7680 & \text{elif } x > -0.21 \\
   0.6400 & \text{elif } x > -1.21 \\
   0 & \text{else}
\end{cases} \]  

4.5 Synthesis of Optimal Control Policies

In this section we present a synthesis algorithm for RT-PCTL. The purpose of synthesis is to generate an optimal piecewise control policy \( \pi^* \) for an RT-PCTL path formula \( \phi \) over a set of atomic propositions. The synthesis maximises the probability of satisfying the given path formula over a set of atomic propositions for every continuous-valued accumulated resource. The accumulated resource is bounded by resource constraint \( h \). The specification for synthesis is denoted as \( P^{x,h}_{\text{max}}[\phi] \) and the optimal control policy would be:

\[ \pi^*(s, x) = \begin{cases} 
   \vdots \quad \vdots \\
   \arg \max_{a_i \in A(s)} \sum_{s' \in S} P_{ss'}^{a_i} \cdot \text{Prob}_{s'}(x + r_s + r_{ss'}) & \text{elif } x > x_i \\
   \vdots \quad \vdots
\end{cases} \]

(4.18)
Algorithm 4.2: Compute piecewise maximum function and index function with respect to array of piecewise functions

Input: Array of piecewise functions $F = \{f_1, \cdots, f_N\}$ where $F[k] = f_k$

Output: Max. piecewise function $f_{\text{max}}(x)$ and max. index $\mu_{\text{max}}(x)$ for all $x \in \mathbb{R}$

1: $c \leftarrow$ ordered union of all subdomains
2: for all $i \in \{1, 2, \cdots N\}$ do
3: \hspace{1em} $p_i \leftarrow \max_{k=1,2,\cdots N} F_a(c_i + \epsilon)$
4: \hspace{1em} $\mu_i \leftarrow \arg \max_{k=1,2,\cdots N} F_a(c_i + \epsilon)$
5: end for
6: $f_{\text{max}}(x) = \begin{cases} 
\vdots 
\text{elif } x > c_i \\
\vdots 
\end{cases}$
7: $\mu_{\text{max}}(x) = \begin{cases} 
\vdots 
\text{elif } x > c_i \\
\vdots 
\end{cases}$
8: return $f_{\text{max}}(x)$ and $\mu_{\text{max}}(x)$

Likewise, the PPF for the optimal solution is in the form of:

$$Prob^*_s(x) = \begin{cases} 
\vdots 
\max_{a_i \in A(s)} \sum_{s' \in S} P_{ss'}^{a_i} \cdot Prob^*_s(x + r_s + r_{ss'}) \text{ elif } x > x_i .
\end{cases} \quad (4.19)$$

Note that $Prob^*_s(x)$ denotes the converged optimal PPF and $Prob^*_s(x)$ denotes the optimal PPF for the $k$-th iteration.

The synthesis algorithm is shown in Algorithm 4.3. For every value of accumulated resource $x$, the action chosen is the one that maximises the probability of satisfaction. The key part of the algorithm is in the calculation of a piecewise maximum function $f_{\text{max}}(x)$ from an array of piecewise functions $F = \{f_1, \cdots, f_N\}$. At every iteration, the PPFs for all actions available are computed for each state. Then the
Algorithm 4.3: Compute optimal control policy and PPFs

**Input:** RT-PCTL path formula, and DTR-MDP

**Output:** Optimal piecewise control policy and PPF

1: \( Prob^0_s(x) \leftarrow 0, \forall s \in S \setminus S^{yes} \)
2: \( Prob^0_s(x) \leftarrow \begin{cases} 
0 & \text{if } x > h^u_s - r_s \\
1 & \text{elif } x > h^l_s - r_s, \forall s \in S^{yes} \\
0 & \text{else} 
\end{cases} \)
3: for \( k = 1 \) to \( K \) do
4:   for \( s \in S^c \) do
5:     for \( a \in A(s) \) do
6:       \( F_a(x) \leftarrow \begin{cases} 
0 & \text{if } x > h^u_s - r_s \\
\sum_{s' \in S} P^{a}_{ss'} \cdot Prob^s_{s'}(x + r_s + r_{ss'}) & \text{elif } x > h^l_s - r_s \\
0 & \text{else} 
\end{cases} \)
7:     if upper saturated then
8:       \( F_a(x) \leftarrow \begin{cases} 
F_a(h^u_s + r_{ss'}) & \text{if } x > h^u_s - r_s \\
\vdots & \text{else} 
\end{cases} \)
9:     end if
10:    if lower saturated then
11:      \( F_a(x) \leftarrow \begin{cases} 
\vdots & \text{if } x > h^l_s - r_s \\
F_a(h^l_s + r_{ss'}) & \text{else} 
\end{cases} \)
12:   end if
13: end for
14: \( Prob_s(x) \leftarrow \max_{a \in A(s)} F_a(x), \forall x \in \mathbb{R} \)
15: \( \pi(s, x) \leftarrow \arg \max_{a \in A(s)} F_a(x), \forall x \in \mathbb{R} \)
16: end for
17: if \( \max_{s \in S, x \in \mathbb{R}} (Prob^k_s(x) - Prob^{k-1}_s(x)) < \epsilon \) then
18:   break
19: end if
20: end for
21: return \( \pi^* \leftarrow \pi \) and \( Prob^K_s \leftarrow Prob \)

maximum function returns the greatest value of probability at a given accumulated resource from the array of PPFs at a state. The algorithm also returns the actions to take at a given accumulated resource in order to maximise the probability.

For the given array of piecewise functions \( F \), the piecewise maximum function \( f_{max}(x) = \)
max\(_{n\in\{1,\ldots,N\}}\) \(F_n(x), \forall x \in \mathbb{R}\) is defined as:

\[
 f_{\text{max}}(x) = \begin{cases} 
 \vdots \\
 \max(\{f_1(c_{i-1}), \ldots, f_n(c_{i-1})\}) \quad \text{elif } x > c_i, \\
 \vdots 
\end{cases}
\]  

(4.20)

where \(c\) is the ordered union set of all subdomains in the array of piecewise functions. The algorithm for maximising the given array of piecewise functions is shown in Algorithm 4.2. Lines 14 and 15 of Algorithm 4.3 use Algorithm 4.2 to compute a piecewise maximum function \(Prob_s(x)\) from an array of piecewise functions \(F = \{f_{a_1}, \ldots, f_{a_N}\}, \forall x \in \mathbb{R}\) where \(a_i \in \mathcal{A}(s)\).

We demonstrate the synthesis of an optimal control policy for the environment in Figure 4.1a from state \(s_1\) for all accumulated resources. The specification would be denoted as \(P_{x_{\text{max}[F_{s_3}]}}\).

Figure 4.2 shows the optimal PPF with respect to the probability of success where the optimal control policy is:

\[
\pi^*(s_1, x) = \begin{cases} 
 \emptyset \quad \text{if } x > +3.79 \\
 Right \quad \text{if } x > +0.95 \\
 Left \quad \text{if } x > -1.21 \\
 \emptyset \quad \text{else}
\end{cases}
\]  

(4.21)

This equation can be further simplified to:

\[
\pi^*(s_1, x) = \begin{cases} 
 Right \quad \text{if } x > +0.95 \\
 Left \quad \text{else}
\end{cases}
\]  

(4.22)

The grey line in Figure 4.2 is the PPF for the single-action policy of going left only and the black line is the PPF for going right only. We obtain the optimal PPF and policy by computing the maximum over these two components.
Figure 4.2 – Optimal control policy and PPF for the simple scenario in Figure 4.1a at state $s_1$. The agent is to go left until the accumulated resource becomes greater than 0.95, and then go right. Simulation results are average values taken over $10^4$ Monte Carlo runs per accumulated resource value.

We also show average values from Monte Carlo simulation runs ($10^4$ runs per data point) for the purpose of validation. A robot in simulation is given the optimal control policy and the set of starting accumulated resources. The star points represent the result for the given accumulated resource with the control policy in Equation 4.22. We observe that the simulated points follow the optimal PPF.
4.5 Synthesis of Optimal Control Policies

4.5.1 Synthesis for Composite Specifications

A set of RT-PCTL state formulas over a set of atomic propositions can be combined together with boolean operators and nested, which we call a composite formula. Such combinations allow for more expressiveness in specifying larger problems with resource constraints.

Composite Specifications with Nested Formulas

Nesting of formulas is allowed by RT-PCTL syntax. There is existing work that addresses how to synthesise sub-optimal controllers for nested PCTL formulas \[57\]. In our work, we synthesise optimal controllers using the algorithms presented in Section 4.5.

We first find an optimal solution for the innermost formula wrapped in a $P$-operator. We then find the set of states satisfying the probability inequality ($\prec \lambda$) and the range of starting resources ($\prec x$) as shown in Section 4.4.3. The set of satisfying states are labelled with a new atomic proposition and the inner formula is replaced with the atomic proposition for the synthesis of the outer formula. The synthesis of nested formulas stops when the outermost formula is synthesised. A set of independent control policies $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ are generated and each control policy $\pi_i$ corresponds to a formula.

Suppose we have a formula $P_{\text{max}}^{x,h}[P_{>0.3}^{S,h}[F\Phi_1]^{\pi_1} U \Phi_2]^{\pi_2}$ that says ‘reach $\Phi_2$ while travelling through the states that have more than 0.3 probability of reaching $\Phi_1$ with initial resource of 5’. We find an optimal solution for the innermost formula $F\Phi_1$ and the corresponding control policy $\pi_1$. Then, the set of states with the satisfaction probability greater than 0.3 given a starting resource 5 are labelled with a new atomic proposition $\Theta_1$, where $\text{Sat}(\Theta_1) = \{s \in S \mid \text{Prob}_s(5) > 0.3 \text{ and } \pi = \pi_1\}$. The original formula can be re-written as $P_{\text{max}}^{x,h}[\Theta_1 U \Phi_2]^{\pi_2}$, which is again synthesised with the algorithm presented in Section 4.5. The synthesis of the composite formula returns two independently-operating control policies $\pi_1$ and $\pi_2$. Note that only one policy is executed at a given time.
Execution of a control policy given a composite specification is different to that using an atomic specification presented in Section 4.5 since there is more than one control policy. Similar to a control policy for an atomic specification that maps a state and resource to an action, we have a global control policy for a composite specification that maps from a state, resource and operational mode to an action. An operational mode \( \mu \subseteq \mathcal{E} \) is a set of events from system and environment that holds true at a given time. For example, the finite set of all possible events could be \( \mathcal{E} = \{\text{‘normal’}, \text{‘engine failure’}\} \). Each control policy \( \pi_i \in \Pi \) is correct with respect to its corresponding atomic formula because only one policy is executed at a given time, and each is correct independently.

The motivation for considering composite specifications of this form comes in part from contingency planning for systems subject to significant failure modes, where a remedial course of action can be taken in the event of such a failure. In [57], a control policy is also constructed for a PCTL composite specification. However, a controller with such a policy is not conditional to operational mode. Therefore it cannot be used to react to events such as ‘engine failure’ as we consider here.

Given a set of control policies \( \Pi \) and a set of conditions \( C = \{C_1, C_2, \cdots, C_{n-1}\} \), we formally define the global control policy as

\[
\pi(s, x, \mu) = \begin{cases} 
\pi_1(s, x) & \text{if } \mu \models C_1 \\
\pi_2(s, x) & \text{else if } \mu \models C_2 \\
\vdots & \\
\pi_n(s, x) & \text{otherwise}
\end{cases} \tag{4.23}
\]

Note that upper condition is prioritised over lower condition. For instance, if both \( C_1 \) and \( C_2 \) hold true, then we execute \( \pi_1 \). Since only one policy runs at a time, there is no conflict between the policies.

From the example above, let \( \Phi_1 \) be an atomic proposition for emergency landing zones and \( \Phi_2 \) be an atomic proposition for destination for an aircraft. Overall, we would like to make sure that the aircraft reaches its destination while having sufficient
chance to reach the landing zones in case of emergency. The initial operational mode of the system would be to run $\pi_2$ which guides the aircraft to reach the destination while passing through the states guaranteed to satisfy the emergency safety condition (i.e., $\mathcal{P}_{h>0.3}[\mathcal{F}\Phi_1]$). In case of emergency while executing $\pi_2$ (e.g., engine failure), the operational mode of the system is switched to execute $\pi_1$. The overall control policy for such behaviour is formally given as

$$
\pi(s, x, \mu) = \begin{cases} 
\pi_1(s, x) & \text{if } \mu \models \text{"Emergency (e.g., engine failure)"} \\
\pi_2(s, x) & \text{otherwise}
\end{cases}
$$

(4.24)

As we synthesise a set of policies for every operational state of the system as opposed to that of plans, the aircraft would always find an action to execute when switching between operational modes if and only if the probability of satisfying the formula after switching is greater than zero. Formally, for any operational state $(s, x)$ given a control policy, there exists an action to execute if and only if $\text{Prob}_s(x, h_s, \phi_i) > 0$. If the probability of satisfying the desired formula is not greater than zero, then switching to the formula is prohibited (i.e., $\pi(s, x) = \emptyset$).

A synthesis of each nested formula is independent of others. This is to avoid searching for every possible combination of control actions that satisfies an inner and an outer formula, which is known to cause exponential growth in the complexity of synthesis [57]. It is important to note that each synthesis is optimal with respect to the probability of success given the inputs. For instance, a solution for an outer formula is optimal given the set of states labelled (e.g., $\Theta_1$) from the synthesis of an inner formula.

**Composite Specifications with Boolean Operators**

Path formulas wrapped in $\mathcal{P}$-operators can be combined by using boolean operators including conjunction, disjunction, negation and their combinations. Like in the synthesis of nested formulas, a control policy is synthesised for each atomic path formula. For example, the formula $\mathcal{P}_{max}[\phi_1]^{\pi_1} \land \mathcal{P}_{max}[\phi_2]^{\pi_2}$ has two independent control
policies $\pi_1$ and $\pi_2$, where each of them is optimal with respect to its corresponding formula. Again, the operational mode of the system governs which control policy is to be executed in a given situation.

### 4.5.2 Analysis

In this section, we prove the correctness and the computational complexity of the synthesis algorithm. In particular, we prove that value iteration converges to a unique optimal solution in a finite time horizon by showing the convergence of the set of subdomains.

For a PPF in the form of Equation 4.14, the set of subdomains in the PPF at timestep $k + 1$ for state $s$ is represented as

$$C_s^{k+1} = \{ c \in \{ h_s^l - r_s, h_s^u - r_s \} \cup \bigcup_{s' \in S^s} (C_{s'}^k - r_s) \mid h_s^l - r_s \leq c \leq h_s^u - r_s \}, \quad (4.25)$$

where $S^s$ is the set of immediate states reachable from state $s$ and the base case is given as

$$C_s^0 = \{ h_s^l - r_s, h_s^u - r_s \}. \quad (4.26)$$

Note that $(C - r) = \{ c_0, c_1, \cdots c_n \} - r = \{ c_0 - r, c_1 - r, \cdots c_n - r \}$.

In this section, any $h$ used without a subscript is defined as a general bound. For simplicity, we use the following notation:

$$\frac{C}{h} = \{ c \in C \mid h^l \leq c \leq h^u \}$$

$$\frac{C}{h - a} = \frac{C}{(h - a) \cap (h - b)}$$

$$= \{ c \in C \mid (h^l - a \leq c \leq h^u - a) \text{ and } (h^l - b \leq c \leq h^u - b) \}$$

$$= \{ c \in C \mid h^l - \min (\{a, b\}) \leq c \leq h^u - \max (\{a, b\}) \},$$

where $A \cap B$ represents the intersection of two bound constraints $A$ and $B$ with
scalars $a$ and $b$. For example, $\{0,5\} \cap \{2,7\} = \{2,5\}$. With this notation, Equation 4.25 can be represented as:

$$C^k_{s+1} = \frac{(h_s - r_s) \cup \bigcup_{s' \in S^s} (C^k_{s'} - r_s)}{h_s - r_s}$$

$$= (h_s - r_s) \cup \bigcup_{s'} \frac{h_s - r_{s'} - r_s}{h_s - r_s} \cup \bigcup_{s''} \frac{h_s - r_{s''} - r_s - r_s}{(h_s - r_s) \cap (h_s - r_{s'} - r_s)} \cup \cdots \cup \bigcup_{s''} \frac{h_s - r_{s''} - r_s - \cdots - r_s}{(h_s - r_s) \cap \cdots \cap (h_s - r_{s'} - \cdots - r_{s'k-1})}.$$  \hfill (4.28)

By generalising further that $S^s = S$ for all states (i.e., all states are immediately reachable from any state), the numerator of each fraction becomes the linear combination of $R$. With the generalisation, we have a general set of subdomains for any state represented as:

$$C^k_{s+1} = \bigcup_{s' \in S^s} (C^k_{s'})$$  \hfill (4.29)

where

$$C^k_{s+1} = \frac{h_s \cup C^k_{s+1}}{h_s}.$$  \hfill (4.30)

For the proof of boundedness of number of subdomains, we consider three cases:
1) $r_s > 0, \forall s \in S$, 2) $r_s < 0, \forall s \in S$ and 3) $r_s \in \mathbb{R}, \forall s \in S$.

**Lemma 1** (Bounded cardinality of subdomain set). The cardinality of the set of subdomains $C^k_s$ at time $k$ is always less than $|S|^k$.

**Proof.** Let $\overline{C}^k_s$ be the set of subdomains without the bound constraints, represented
as

$$C^k_s = h_s \cup \bigcup_{s' \in S} C^{k-1}_{s'}.$$  

(4.31)

The cardinality of the set is

$$|C^k_s| = |h_s \cup \bigcup_{s' \in S} C^{k-1}_{s'}| = |\bigcup_{s' \in S} C^{k-1}_{s'}| \leq |S| \cdot |C^{k-1}| \leq |S|^k.$$  

(4.32)

Since $|C^k_s| \leq |C^k| \leq |\overline{C^k}|$, the number of subdomains in $C^k_s$ is always less than $|S|^k$. □

Lemma 2 (Convergence of PPF in special cases). If all $r \in R$ are positive or negative real numbers, the set of subdomains in a PPF converges within a finite number of iterations.

Proof. Suppose we have the following equation (Equation 4.33) where $C$ is an arbitrary set of real numbers with a finite number of elements, $h$ is a bound constraint and $r_i$ are positive real numbers:

$$C \subseteq \left( h - r_1 \right) \cap \left( h - r_1 - r_2 \right) \cap \cdots \cap \left( h - r_1 - \cdots - r_n \right).$$  

(4.33)

Since all elements of $r_i$ are positive, the bound constraint on the set $C$ is $\{h^l - r_1, h^u - r_1 - r_2 - \cdots - r_n\}$ as shown in Equation 4.27. As all $r_1$ are positive, the upper bound decreases monotonically. Therefore there exists a finite value of $n$ where the upper bound goes below the lower bound. In such case, the resulting set would be an empty set. Substituting Equation 4.33 into Equation 4.28, there exists a finite value $k$ such that the cardinality of $C^{k+1}_s$ and $C^k_s$ are equal. Without loss of generality, we can also conclude that the set converges for the case where all elements in $R$ are negative. □

Lemma 3 (Convergence of PPF in general cases). If all $r \in R$ are rational numbers...
or all quotients of all possible pairs \((r_i, r_j) \in R\) are rational, the set of subdomains in a PPF converges within a finite number of iterations.

**Proof.** Let \(p = \{p_1, p_2\}\) be a pair of two numbers chosen from \(R\). If the elements of the pair are all positive or all negative, the set of subdomains converges by Lemma 2. If a negative and a positive number are chosen for \(p\), the smallest positive linear combination is the greatest common divisor (gcd) from number theory. Therefore, the number of elements added to the set of subdomains for the pair is finite if and only if the quotient is rational since the gcd only exists when the quotient is rational. □

**Lemma 4** (Maximum number of iterations before convergence). The maximum number of iterations before the convergence of the set of subdomains in each state is

\[
k_{\text{max}} = \begin{cases} 
\frac{\max_{s \in S} h_s^u - h_s^l}{\min_i |r_i|} & \text{all } r \in R \text{ positive or negative} \\
\frac{\max_{s \in S} h_s^u - h_s^l}{\min_{i,j} \gcd(r_i, r_j)} & \text{otherwise}
\end{cases}
\] (4.34)

where \(n\) is the cardinality of \(R\) and \(\gcd(r_i, r_j) \in R\) is the greatest common divisor of a pair \((r_i, r_j)\) with \(r_i > 0\) and \(r_j < 0\).

**Proof.** Let \(s^* = \arg \max_{s \in S} h_s^u - h_s^l, x \in [h_s^l, h_s^u]\) and \(lc^k(R)\) be a set of linear combinations such that \(\{e \in x - a_1 r_1 - a_2 r_2 - \cdots - a_n r_n \in [h_s^l, h_s^u] \mid a_i \in \mathbb{N}, a_i \geq 0, \sum_{i=1}^n a_i = k, r_i \in R\}\). If all \(r \in R\) are positive real numbers, the non-empty \(lc^k(R)\) with the longest \(k\) is \(\{x - k \min_i r_i\}\) for a given \(x\). Since the upper bound for \(x\) is \(h_s^u\), the longest \(k\) before the set of linear combinations becomes an empty set is \(k_{\text{max}} = \frac{\max_{s \in S} h_s^u - h_s^l}{\min_i |r_i|}\). Without loss of generality, we have \(k_{\text{max}} = \frac{\max_{s \in S} h_s^u - h_s^l}{\min_{i,j} \gcd(r_i, r_j)}\) if \(r \in R\) are all positive or all negative real numbers.

If the set \(R\) contains a combination of positive and negative real numbers, the smallest positive linear combination is the minimum difference between any two subdomains that are made from a pair \((r_i, r_j) \in R\). Since the smallest positive linear combination is the greatest common divisor, the maximum number of iterations is \(\frac{\max_{s \in S} h_s^u - h_s^l}{\min_{i,j} \gcd(r_i, r_j)}\) where all possible pairs \(\{r_i, r_j\}\) satisfy \(r_i > 0\) and \(r_j < 0\), and the quotient should be a rational number by Lemma 3. □
With the proof that the set of subdomains converges to a finite set within a finite number of iterations and the proof that the cardinality of the set of subdomains is bounded, we have the following theorem:

**Theorem 4.1** (PPF convergence to unique fixed point). The PPF converges to a unique fixed point.

**Proof.** From Lemma 1 and Lemma 4, it is shown that the number of subdomains in each state converges to a finite set in a finite number of iterations. Considering each subdomain as a state, a DTR-MDP can be reduced to a DT-MDP. As DT-MDPs converge to a unique fixed point using the value iteration method, the PPF also converges.

From Lemma 4, the number of states in a DTR-MDP reduced to a DT-MDP is proportional to $|S|^{|h|}$, where $|S|$ is the number of states and $|h|$ is the width of bound constraints. Since the time complexity of solving a DT-MDP is polynomial in the number of states, the time complexity of solving a DTR-MDP, and hence synthesising an optimal policy, is polynomial in $|S|^{|h|}$ and linear in the size of formula $|\Phi|$. Note that the width of resource bound $|h|$ may be a large number, but as it is independent of the number of states $|S|$, the time complexity is still polynomial in the number of states.

### 4.6 Summary

In this chapter, we have discussed the task planning problem with resource threshold constraints. A new extended logic RT-PCTL has been formally defined with syntax and semantics, and the related model-checking and synthesis algorithms have been proposed with mathematical analysis of complexity. The approach of extending a logic for the threshold constraints is novel in that the constraints can be directly specified in a formal manner. The following chapter discusses the other type of important resource constraints: *fuel constraints in wind.*
Chapter 5

Task Planning with Fuel Constraints in Wind

In this chapter, we continue our treatment of resource constraints with focus on fuel constraints in wind. We define the problem and introduce realistic models of fuel combustion, wind fields and unmanned aerial vehicle (UAV) dynamics in Sections 5.1 and 5.2. We then present synthesis algorithms which consist of discrete synthesis and continuous execution and provide complexity analysis in Section 5.3. We conclude this chapter with a summary in Section 5.4.

5.1 Fuel Constraints in Wind

The objective of the problem with fuel constraints in wind is to synthesise a controller that satisfies a high-level specification in the presence of external forces. In particular, we are interested in a petrol-powered UAV travelling in winds. The presence of wind affects the rate of fuel consumption and the dynamics of the UAV significantly. However, there exists no known analytical solution for the optimal fuel consumption and the trajectory of the UAV in the presence of wind, which is also known as Zermelo’s problem. Therefore, in order to make a rational decision, the wind conditions must
be known at the task level and there must be a coupling between the task and low levels.

Suppose we have a UAV in an environment with a complex mission such as surveillance and sequencing under the influence of wind and fuel constraints. The UAV is required to synthesise a task-level planner for the mission and a low-level controller for actuation. In this section, we develop a two-layered online synthesis algorithm which consists of discrete synthesis and continuous execution.

In discrete synthesis, the environment and the UAV dynamics are discretised and wind vectors are approximated for each discrete state. The algorithm is to find a sequence of discrete states satisfying the task-level mission specification. In particular, the sequence minimises the fuel consumption, with the presence of wind affecting the fuel consumption. We present an efficient algorithm to plan at the task-level with the given complex mission specification so that the planning can be done online. In continuous execution, the sequence of discrete states is realised with continuous dynamics of the UAV and the influence of continuous wind. In this chapter, we focus on a practical flight mission where the UAV is to start a landing procedure when the fuel goes below a certain threshold. Therefore we have a mission of the form ‘If the fuel level is above a threshold, mission $\phi$ is taken. If not, a landing procedure $\phi_e$ starts’. This form of mission is chosen for the purpose of solving the problem. The proposed algorithm can easily be extended further to express a general form.

In the following section, we introduce the fuel model of the UAV and the interpolation methods for the wind field. Lastly the UAV dynamics and the controller model are defined.

### 5.2 Models of Platform and Environments

#### 5.2.1 Breguet Range Equation

The UAV operates at constant altitude and air speed, subject to wind currents. Since fuel is consumed over time, the change in the mass of the fuel affects the flight
dynamics of the UAV significantly. The relationship between the ground distance travelled and the mass of fuel is represented by the Breguet range equation

\[ d_g = v_g \cdot C_a \cdot \log \frac{M_i}{M_f}, \]  

(5.1)

where \( d_g \) is the ground distance travelled, \( v_g \) is the ground velocity, and \( M_i \) and \( M_f \) are the initial and the final mass of the fuel respectively. We have \( C_a = I_{sp} \cdot L/D \) where \( I_{sp} \) is the specific impulse, and \( L/D \) is the lift-to-drag ratio.

With the presence of a tail wind, the ground velocity is re-written as \( v_g = v_a + v_w \) where \( v_a \) is the air velocity and \( v_w \) is the tail wind velocity. The mass after travelling an infinitesimal ground distance (\( dx \)) or time (\( dt \)) is shown as

\[ M_f = M_i \cdot \exp\left(-\frac{dx}{(v_a + v_w) \cdot C_a}\right) \]

\[ = M_i \cdot \exp\left(-\frac{dt}{C_a}\right). \]  

(5.2)

### 5.2.2 Wind Interpolation with Gaussian Process Regression

Continuous wind vectors are interpolated using Gaussian process regression given a number of observation points. The wind vector is assumed to be time-invariant and noise-free [60]. We use the typical squared exponential covariance function \( k(x, x') = \exp(-\frac{1}{2\lambda}||x - x'||^2) \) where \( \lambda \) is a length scale. Suppose we are interested in a wind vector at a point \( x^* \) with a number of observation points \( X \) and the corresponding observed wind vectors \( Y \). We have the following equation:

\[ V_w(x^*) = K(x^*, X)[K(X, X)]^{-1}Y, \]  

(5.3)

where \( K \) is the covariance matrix with components \( k(x, x') \) for all \( x, x' \in X \). Note that \( x \) and \( y \) dimensions are independent and share the same \( \lambda \). Optimising the values of the length scale is not the focus of this work; assuming that the wind field is smooth and does not vary rapidly, a large value is suitable for the interpolation.

In particular, we assume that the wind field values are spatially correlated and that
the length scale is approximately the distance between two nearest observations. Given an environment discretised into a grid with a set of discrete states $S$, the mean wind vector for each discrete state is

$$ V_w[s] = \frac{\int_{x \in X_s} V_w(x) \cdot dx}{\prod_{d=1}^{D} \|X_s^d\|}, \quad (5.4) $$

where $s \in S$ is a discrete state, $X_s$ is the extent of cell $s$ where $\|X_s^d\|$ is the length of the cell in the $d$-dimension.

### 5.2.3 UAV Dynamics and Controller Model

The UAV operating at a constant altitude and a constant airspeed $v_a$ is affected by wind currents. Therefore the state vector for the UAV is $X = [x, y, \psi]^T$ where $\psi$ is the heading angle. The control input is $u = \psi' \in U \subset \mathbb{R}$ where $U$ is a finite set of possible turn rates. If the UAV is moving in a wind field represented with a function $V_w(x, y)$ as interpolated in Equation 5.4, the dynamics of the UAV become

$$ x'(t) = v_a \cdot \cos(\psi(t)) + V_{wx}(x(t), y(t)) $$

$$ y'(t) = v_a \cdot \sin(\psi(t)) + V_{wy}(x(t), y(t)) $$

$$ \psi'(t) = u(t) \in U, \quad (5.5) $$

where $V_{wx}$ and $V_{wy}$ are the tail wind in the $x$ and $y$ axis respectively. Since the system of differential equations is non-linear, we solve them numerically:

$$ x[t + \Delta t] = (v_a \cdot \cos(\psi[t]) + V_{wx}(x[t], y[t])) \cdot \Delta t + x[t] $$

$$ y[t + \Delta t] = (v_a \cdot \sin(\psi[t]) + V_{wy}(x[t], y[t])) \cdot \Delta t + y[t] $$

$$ \psi[t + \Delta t] = u[t] \cdot \Delta t + \psi[t]. \quad (5.6) $$
5.3 Synthesis for Continuous Trajectories

5.3.1 Discrete Synthesis

As mentioned in Section 5.1, the normal flight mission is to be aborted when the fuel level is below a threshold and a landing procedure should then begin. The mission is expressed in linear temporal logic (LTL) as:

$$\Box((\neg e_\alpha \Rightarrow \phi) \land (e_\alpha \Rightarrow \phi_e)),$$

(5.7)

where $\phi$ is an LTL formula over a set of atomic propositions for the normal flight mission and $\phi_e$ is a formula over a set of atomic propositions for the emergency flight plan. The formula for the emergency plan is $[\neg d_{\text{land}} U g_{\text{land}}]$ where $d_{\text{land}}$ is a proposition to avoid and $g_{\text{land}}$ is a proposition to reach in the landing procedure. The symbol $e_\alpha$ is a signal produced by the low-level controller when the fuel goes below a threshold $\alpha$. Note that we solve for $\phi$ and $\phi_e$ separately. More details about solving the formula are shown in Section 5.3.2.

We discretise a continuous space in environment into a rectangular uniform grid of $xy$-discrete position states $S$, where $X_s$ is a vector for the size of each discrete state $s \in S$. The size of each discrete state is chosen heuristically. For each discrete state, we calculate the mean wind vector as in Equation 5.4. Each discrete state is labelled with symbolic propositions based on the mission. 3D position states could also be considered, but we focus on the 2D case here for simplicity.

We propose a greedy Büchi algorithm (GBA) to find a sequence of discrete states that minimises fuel consumption at the task level. The algorithm is optimal in one Büchi horizon, where $n$ Büchi horizons refers to $n$ transitions in Büchi states. The sequence is minimum fuel consuming for one transition in the Büchi automaton.

A Büchi automaton is generated from a given LTL formula $\phi$ over a set of atomic propositions. From a discrete state $s$ and Büchi state $q$, we find a sequence of discrete states that produces a finite word $\omega$ to transit to the next Büchi state $q'$. The sequence
Consider an example environment shown in Figure 5.1a and a Büchi automaton in Figure 5.1b of formula $\Box(\neg c U a) \land \Box(\neg c U b)$ over a set of atomic propositions (i.e., ‘visit $a$ and $b$ infinitely often while avoiding $c$’). From the Büchi automaton with initial state $q_0$, the set of valid input alphabets are expressed as $a \land b$, $a \land \neg c \land \neg b$ and $\neg a \land \neg c$, where the first two expressions allow transiting to the next available Büchi state (i.e. $trans(q_0) = a \land b \lor a \land \neg c \land \neg b$ and $stay(q_0) = \neg a \land \neg c$).

The problem is then formulated as a finite-horizon MDP with a deterministic transition model. The state space of the MDP is equivalent to the discrete environment representation $S$, the action space of the MDP corresponds to movement between adjacent environment states (a 4-connected grid in this example), and the reward function of the MDP corresponds to fuel consumption. Given a Büchi state $q$, the discrete states $L(s_{trans}) \in trans(q)$ are to be reached while moving through the states $L(s_{stay}) \in stay(q)$ while minimising fuel consumption. Solving the MDP pro-

Figure 5.1 – A simple environment and Büchi automaton. (a) Simple example environment shown discretised into a $3 \times 3$ grid with continuous wind vector field and mean wind vector for each discrete state (bold arrows). States $s_1$, $s_9$ and $s_4$ are labelled with $a$, $b$ and $c$ respectively. (b) A deterministic Büchi automaton for this LTL formula $\Box(\neg c U a) \land \Box(\neg c U b)$ where $a$ and $b$ have to be visited infinitely often while avoiding $c$. of discrete states generated is optimal in the discrete space with respect to fuel consumption with mean wind vectors. The advantage of our approach over the typical approaches [78, 50] of building a product automaton is discussed in Section 5.3.3.
5.3 Synthesis for Continuous Trajectories

Figure 5.2 – A transition from discrete state $s$ to $s'$ is shown with wind vectors $V_w[s]$ and $V_w[s']$. In the approximation of fuel consumption, we assume that the UAV moves between centres of states. The UAV has a tail wind (i.e., same direction) when in state $s$ and a head wind (i.e., opposite direction) when in state $s'$.

vides an optimal sequence to transit from the Büchi state $q$ to another $q' \in Q$. For example, starting from $s_3$ and $q_0$, the goal is to reach $s_1$ while avoiding $s_4$ with minimum fuel consumption. One possible sequence would be $s_3s_6s_5s_2s_1$ where the produced word $\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset a$ is a finite prefix of an accepting word.

The mean wind vectors are computed as in Equation 5.4 using Gaussian process regression. Suppose a transition is made from a discrete state $s$ to adjacent state $s'$ as shown in Figure 5.2. Since the UAV is moving horizontally, the wind vectors affecting the movement in the $x$ direction are denoted as $V_{wx}[s]$ and $V_{wx}[s']$. Therefore, the fuel equation for travelling between the centres of $s$ and $s'$ from Equation 5.2 can be re-written as

$$M_1 = M_0 \cdot \exp\left(-\frac{d_g \cdot (v_a \pm V_{wd}[s]) \cdot C_a}{(v_a \pm V_{wd}[s]) \cdot C_a}\right) \cdot \exp\left(-\frac{d_g \cdot (v_a \pm V_{wd}[s']) \cdot C_a}{(v_a \pm V_{wd}[s']) \cdot C_a}\right),$$

(5.8)

where $\pm V_{wd}[s]$ is the tail wind in the direction of UAV movement. If the direction of $V_{wd}[s]$ is opposite to the UAV movement, then the value becomes negative. For example, the wind vector in state $s$ from Figure 5.2 is positive since it is a tail wind whereas the vector in state $s'$ is negative since it is blowing against the movement of the UAV.

We solve the MDP using value iteration. Given a discrete state $s$, Büchi state $q$ and
Task Planning with Fuel Constraints in Wind

Algorithm 5.1: Synthesis of optimal sequence to next Büchi State

```latex
function seq ← GetSequence(s₀, q₀, B, Q_{seq})

1: ∀s ∈ S, F[s] ← \begin{cases} 
1 & \text{if } L(s) ∈ B\text{-}trans(q₀) \text{ and } q' ∉ Q_{seq}\setminus q \\
0 & \text{otherwise} 
\end{cases}

2: repeat
3:   F ← F
4: for all s ∈ B\text{-}stay(q₀) do 
5:   \{F[s], π[s]\} ← \max_{d ∈ D} F[s'] \cdot \exp\left(-\frac{d_g}{C_a} \cdot \frac{1}{v_a ± V_{wd}[s]} + \frac{1}{v_a ± V_{wd}[s']}\right)
6: end for 
7: until min(|F − F|) < ϵ
8: seq ← get sequence from s₀ by following π
9: return seq
```

a set of approximated wind vectors, we solve the following equation:

\[
F[s] = \begin{cases} 
F[s'] \cdot \exp\left(-\frac{d_g}{C_a} \cdot \left(\frac{1}{v_a ± V_{wd}[s]} + \frac{1}{v_a ± V_{wd}[s']}\right)\right) & \text{if } L(s) ∈ stay(q) \\
1 & \text{if } L(s) ∈ trans(q) \text{ and } q' ∉ Q_{seq}\setminus q \\
0 & \text{otherwise} 
\end{cases} 
\]  

(5.9)

where \( F[s] \) is the proportion of fuel remaining when entering the destination and \( q' \) is the next Büchi state when transiting from \( s \) to \( s' \). Based on Equation 5.9, we solve for \( F^*[s] = \max_{d ∈ D} F[s] \) and \( π^*[s] = \arg\max_{d ∈ D} F[s] \) where \( d ∈ D \) is a head direction, \( π^* \) is an optimal control policy and \( Q_{seq} \) is a set of visited Büchi states. Note that the set of Büchi states already visited, \( Q_{seq} \), is not to be re-visited until reaching an accepting state \( F ⊆ Q \). The optimal sequence is calculated by following the control policy from an initial discrete state. Pseudocode is listed as Algorithm 5.1.
5.3 Synthesis for Continuous Trajectories

5.3.2 Continuous Execution

In this section, we present how the sequence of discrete states is realised in a continuous environment with realistic dynamics in the presence of wind. To solve the non-linear system of differential equations in Equation 5.5, we assume a discrete number of available control inputs (turn rates) $U = \{-a_n, \ldots, -a_1, 0, a_1, \ldots, a_n\}$ deg $s^{-1}$.

Given an initial state of the UAV $x = [x_0, y_0, \psi_0]^T$ at a discrete state $s \in S$ and the optimal sequence from discrete synthesis, we iteratively forward integrate all available control inputs $u \in U$ to create a set of candidate trajectories that reach the boundary $X_s$ of the current discrete state. After each control propagation, a trajectory is pruned if the next discrete state is not the next state in the discrete sequence. Since the number of candidate trajectories grows after each iteration, we limit the number of those trajectories by selecting $N$ least-fuel-consuming candidates and prune all others. Therefore we have at most $N$ trajectories as opposed to $|U|^K$ where $K$ is the total number of sequences throughout the mission. The value of $N$ can be chosen practically; $N$ is treated as a parameter of the algorithm.

After all control inputs have been propagated, we consider the trajectory with the least fuel consumption and check the fuel level at the end of this trajectory. If the fuel left is below the specified threshold, then a new discrete sequence following a landing procedure is synthesised. If not, each candidate trajectory starts a new iteration by applying all control inputs. Once a trajectory reaches the end of the discrete sequence, the next sequence is synthesised as shown in Section 5.3.1.

The algorithm for following the sequence is shown in Algorithm 5.2 where $g \in G$ denotes a candidate trajectory with the UAV position, discrete state, Büchi state and fuel level. The overall realisation is presented in Algorithm 5.3 where $\phi$ is an LTL formula for normal operation, $\phi_e$ is a formula for a landing procedure, and $\alpha$ is a fuel level threshold to execute the landing procedure. As $\phi_e$ is for a landing procedure, all the accepting states in the Büchi automaton $B_e$ must be absorbing states.

Suppose we have an example shown in Figure 5.3 where the objective is to visit $a$ and $b$ infinitely often while avoiding $c$. The initial candidate trajectory starts at a
Algorithm 5.2: Synthesis of continuous trajectory from sequence of discrete states

function $G_{\text{new}} \leftarrow \text{SynthesiseTraj}(G_0, seq, U, N)$

1: $G \leftarrow G_0$
2: for $i \leftarrow 1$ to $\text{seq.length} - 1$ do
3:     $s_{\text{Next}} \leftarrow \text{seq}[i]$
4:     $G_{\text{new}} \leftarrow \{\emptyset\}$
5:     for all $g \in G$ and $u \in U$ do
6:         $g_{\text{new}} \leftarrow \text{ApplyControlInput}(g, u)$
7:         if $g_{\text{new}}$ terminates at $\text{seq}[i + 1]$ then
8:             $G_{\text{new}}.\text{add}(g_{\text{new}})$
9:         end if
10:     $G_{\text{new}} \leftarrow \text{N-best } g \in G_{\text{new}}$
11: end for
12: end for
13: return $G_{\text{new}}$

state $[0.5, 2.5, 30\text{deg}]^T$ in which the discrete state is $s_3$. The initial sequence given from the discrete synthesis is $s_3s_6s_5s_2s_1$. Figure 5.3a shows the result of applying all possible turn rates $U = \{-6, -4, -2, 0, 2, 4, 6\} \text{deg s}^{-1}$ until the trajectory hits the boundary of the next discrete state. Note that the bold black line is the least-fuel-consuming trajectory. Since $s_6$ is preceded by $s_3$ in the sequence given, all trajectories remain for the next iteration. We select the best control action that minimises fuel consumption and the best turn rate of $-6 \text{deg s}^{-1}$ is found for the first discrete state $s_3$.

In the next iteration in Figure 5.3b, the candidate trajectories terminate at discrete state $s_5$ where all other trajectories terminating at other states are abandoned. Note that the number of candidate trajectories is limited to 10 in this example. At the 5th iteration in Figure 5.3e, a new sequence is synthesised from the discrete synthesis after reaching the goal discrete state with the target input alphabet $a \in \text{trans}(q_0)$.

From the 5th to 8th iterations, the optimal sequence is $s_1s_2s_3s_6s_9$. The trajectories at the 28th iteration are shown in Figure 5.3i.
5.3 Synthesis for Continuous Trajectories

Figure 5.3 – The UAV is to visit $s_1$ and $s_9$ infinitely often while avoiding $s_4$. The optimal sequence of discrete states is given prior to synthesising a continuous trajectory. The environment size is $3000m \times 3000m$ with 9 discrete states. The airspeed is $5ms^{-1}$ and the available turn rates are $\{-6, -4, -2, 0, 2, 0, 6\}$ deg $s^{-1}$. The least fuel consuming trajectory is plotted with a bold black line and candidate trajectories are shown in red. The maximum number of candidate trajectories is limited to 10.
Algorithm 5.3: Overall execution of controller in continuous state space

\begin{verbatim}
function g* ← Execute(x_0, φ, φ_e, α, N)
1: Q_seq ← ∅
2: B ← ConstructBüchiAutomaton(φ)
3: B_e ← ConstructBüchiAutomaton(φ_e)
4: g_0 ← init(x_0, s_0 ← GetDiscreteState(x_0), q_0 ← B.q_0, fuel_0 ← 1)
5: G.add(g_0)
6: g* ← g_0
7: repeat
8: if g*.fuel ≥ α then
9: seq ← GetSequence(g*.s, g*.q, B, Q_seq)
10: G ← SynthesiseTraj(G, seq, U, N)
11: g* ← arg min G.fuel
12: else
13: seq_e ← GetSequence(g*.s, B_e.q_0, B_e, Q_seq)
14: G ← SynthesiseTraj(G, seq_e, U, N)
15: g* ← arg min G.fuel
16: end if
17: if g*.q ∈ B.F then
18: Q_seq ← ∅
19: else
20: Q_seq.add(g*.q)
21: end if
22: Execute g*
23: until g*.fuel < α
24: return g*
\end{verbatim}

5.3.3 Analysis

The time complexity of constructing a Büchi automaton B from an LTL formula φ is $O(2^{||φ||})$ [3]. The value iteration algorithm in Equation 5.9, known to have the complexity $O(poly(||S||))$ [69], is run to find an optimal sequence of discrete states for a transition in the Büchi automaton. Note that $poly(n)$ means ‘polynomial in n’. Since transition to any visited Büchi state is prohibited before reaching an accepting state, the maximum number of Büchi state transitions to reach an accepting state is $|Q| - 1$ where Q is a set of Büchi states and $|Q| < 2^{||φ||}$. The maximum number of candidate trajectories in trajectory synthesis is restricted to N. The time complexity of solving
an MDP with deterministic transitions, as in our case, is $O(|S|^2)$ [63]. Therefore the overall time complexity of solving for a single Büchi transition is $O(|S|^2 + |S| \cdot |U| \cdot N)$.

As GBA does not construct a product automaton, a locally optimal sequence of discrete states can be acquired online as opposed to constructing a product automaton and searching exhaustively. Synthesis for a single Büchi transition is efficient, so for state spaces of reasonable size this synthesis can feasibly be performed during the execution of the previous transition. Further, changes in the environment (such as time-variant wind predictions) do not affect the Büchi automaton and thus the cost of its construction is only incurred once (at the start of the mission).

The space complexity is $O(|S| + N \cdot |U| + 2^{|\phi|})$. We need $|S|$ space to solve value iteration, $N \cdot |U|$ to find the best trajectory and $2^{|\phi|}$ to construct a Büchi automaton. Note that typical synthesis algorithms require a construction of a product automata with size $O(|S| \cdot 2^{|\phi|})$ [78].

Although the size of the Büchi automaton is exponential in the size of a formula, the formula is often relatively small compared to the size of the discrete state space. If formula size is assumed to be constant, the space complexity is $O(|S| + N \cdot |U|)$.

5.4 Summary

This chapter has discussed the task planning problem with fuel constraints in the presence of wind. We have proposed a framework with an efficient algorithm called GBA for online synthesis by reducing the synthesis problem size. The algorithm has been coupled with low-level controllers with realistic models. Along with Chapter 4, this chapter has addressed issues in task planning with resource constraints. In Chapter 6, we address the other class of constraints considered in this thesis: safety constraints.
Chapter 6

Task Planning with Safety Constraints

In this chapter, the need for balancing safety and completion time objectives is addressed and proofs and algorithms are proposed for model-checking and synthesis. In Section 6.1, we define the problem in a Markov decision process (MDP) formulation and address limitations of this formulation. In Section 6.2, we derive the analytical relationship between safety and completion time for a given probabilistic computation tree logic (PCTL) formula in two ways. In Section 6.3, we present a synthesis algorithm for an optimal controller for the bi-criteria objective. Section 6.4 summarises the chapter.

6.1 Safety Constraints

The problem with safety constraints is to synthesise a provably-correct controller for a high-level specification with bi-criteria objectives. In particular, we are interested in the synthesis of a controller considering both completion time and probability of completion (i.e., safety). Often, these objectives are addressed independently, which results in either a too conservative or aggressive controller. In some approaches, the objectives are considered in a single framework. However, balancing the objectives is
Figure 6.1 – Simple demonstration environment, originally presented in [85]. State in green is the goal state \( s_{23} \), states in red are with danger \( s_{13}, s_{16}, s_{18} \) and state in black is obstacle \( s_8 \) while others in white are free-to-move states. The set of available actions are to move up, down, left and right. There is 80% probability of desired transition and 10% probability of moving \( \pm 90 \) degrees in the intended direction. When moving toward the boundary or an obstacle, an agent is bounced back to the original state.

not trivial, and it is difficult to evaluate the performance of the resulting controller. Therefore, the relationship between completion time and safety must be proved analytically with respect to a high-level specification and then a synthesis algorithm should be derived from the proof.

6.1.1 Motivating Example

Suppose we have a simple environment shown in Figure 6.1. The environment is formulated as an MDP as described in Chapter 3. The environment consists of 25 states where each state is labelled with an atomic proposition. Green, red and black states represent goal, danger and obstacle states respectively while the white states are the free states. The set of available actions for all states \( s \in S \) is \( \mathcal{A}(s) = \{ \text{right, up, left, down} \} \). There is 80% probability of successful transition and 10% probability of transiting \( \pm 90 \) deg in the intended direction. An agent is bounced back
### 6.1 Safety Constraints

Figure 6.2 – Balancing the reward matrix in an MDP formulation with different sets of costs for danger states. The values on each cell represent the expected sums of rewards/costs.
to its starting state when it hits the boundary or an obstacle.

The objective of the MDP formulation is for the agent to reach the goal state while avoiding any danger states. Since entering any danger states leads to mission failure, it is important to ensure a certain level of safety by finding a control policy that reduces the probability of entering these states in the presence of actuation noise. At the same time, expected completion time of the mission could be another important measure of performance. If safety is the only factor considered in the synthesis, the behaviour of the agent would be too conservative; if only completion time is considered, the behaviour would be too aggressive. Also, in order to provide sound reasoning for choice of a control policy, the performance of the factors should be measurable and comparable. Hence, it is imperative to quantify the performance.

6.1.2 Limitations of MDP Formulation

With a conventional MDP formulation, an optimal path is generated by maximising the future expected sum of rewards, in which danger states are given negative rewards and the goal states are given positive rewards. Figure 6.2 shows the optimal paths generated with different costs for danger states with discounting factor of 0.5.

Each of the results in Figure 6.2 shows the optimal path when collisions with the danger states were rewarded with $-1, -10, -100, -1000$. The results show that increasing the goal-to-obstacle ratio generates a more conservative policy. Decreasing the ratio generates a policy that takes less time to reach the goal. Although the reward for each state can be adjusted to balance between being conservative and risky, the choice of this ratio is not intuitive.

The task can be written as $\phi = [\neg \text{danger} U \text{goal}]$ using PCTL over a set of atomic propositions. With this mission specification, the optimal path with respect to the probability of mission success can be synthesised by solving Equation 6.1. Note that $\text{Prob}^*(s, \phi)$ is the probability of satisfying the path formula $\phi$ over a set of atomic
propositions at state $s$ for an optimal case:

$$Prob^*(s, \phi) = \max_{a^* \in A(s)} \sum_{s' \in S} T(s, s') \cdot Prob^*(s', \phi).$$  \hspace{1cm} (6.1)

We use a short form $Prob^k(s)$ to denote the probability of satisfaction at state $s$ and timestep $k$ and $Prob(s)$ for the probability in optimal case.

Figure 6.3 shows the optimal path with respect to the PCTL specification over a set of atomic propositions. The policy generated guarantees the maximum probability of accomplishing the mission in an infinite time horizon and has intuitive quantitative values. However this PCTL optimal policy is conservative since it only considers safety while other real robotic applications are also constrained with other aspects such as completion time. For instance at state $s_{22}$ (i.e., the state right below the goal state in green), the optimal action is to hit the wall until entering the goal with the probability of 10%. Such an action is too extreme in practice since the choice of action only considers safety.
6.2 Expected Time to Accomplish Mission

We propose a model-checking method to evaluate the expected number of time steps to accomplish a given mission specification written in PCTL along with a probabilistic satisfaction guarantee. The objective is to check if a state is satisfied with the safety constraint given in the form of mission success probability.

The expected time for mission success from state $s$ is defined with notation $\tau(s)$. Since any paths that failed to complete a mission should not be considered, we define $\tau(s \mid v)$ as the expected time from state $s$ given success $v$. The formal definition of $\tau(s \mid v)$ is shown in Equation 6.2 where the evaluation is done with a given control policy $\pi$:

$$\tau(s \mid v) = \mathbb{E}_\pi[\# \text{ of time steps } | s, v]$$
$$= \mathbb{E}_\pi[k \mid s, v]$$
$$= \sum_{k=0}^{\infty} k \cdot \mathbb{P}_s(\text{becoming successful at } k \mid v)$$
$$= \sum_{k=0}^{\infty} k \cdot \mathbb{P}_s(v^k \bigcap_{i=0}^{k-1} \neg v^i \mid v). \tag{6.2}$$

The probability $\mathbb{P}_s(v^k \bigcap_{i=0}^{k-1} \neg v^i \mid v)$ is the probability of being successful at time $k$ while not being so before time $k$, given that the mission as a whole is successful. The relationship between this probability and the success probability from PCTL is shown
in Equation 6.3.

\[
\mathbb{P}_s(v^k \bigcap_{i=0}^{k-1} \neg v^i \mid v) \\
= \frac{\mathbb{P}_s(v \mid v^k \bigcap_{i=0}^{k-1} \neg v^i) \cdot \mathbb{P}(v^k \bigcap_{i=0}^{k-1} \neg v^i)}{\mathbb{P}_s(v)} \\
= \frac{\mathbb{P}_s(v^k \bigcap_{i=0}^{k-1} \neg v^i)}{\mathbb{P}_s(v)} \\
= \frac{\mathbb{P}_s(v^k \bigcap_{i=0}^{k-1} \neg v^i) + \sum_{i=0}^{k-1} \mathbb{P}_s(v^i \bigcap_{j=1}^{i-1} \neg v^j) - \sum_{i=0}^{k-1} \mathbb{P}_s(v^i \bigcap_{j=0}^{i-1} \neg v^j)}{\mathbb{P}_s(v)} \\
= \frac{\sum_{i=0}^{k} \mathbb{P}_s(v^i \bigcap_{j=0}^{i-1} \neg v^j) - \sum_{i=0}^{k-1} \mathbb{P}_s(v^i \bigcap_{j=0}^{i-1} \neg v^j)}{\mathbb{P}_s(v)} \\
= \frac{\text{Prob}^k(s) - \text{Prob}^{k-1}(s)}{\text{Prob}(s)}
\]

Substituting Equation 6.2 into Equation 6.3 results in

\[
\tau(s \mid v) = \sum_{k=1}^{\infty} k \cdot \frac{\text{Prob}^k(s) - \text{Prob}^{k-1}(s)}{\text{Prob}(s)}.
\]

The algorithm for computing \(\tau(s \mid v)\) is shown in Algorithm 6.1. It terminates when the maximum element-wise difference between the current and previous \(\tau(s \mid v)\) is less than a small value \(\varepsilon\). Using Algorithm 6.1, control policy \(\pi\) can be evaluated to quantitatively measure the expected time to successfully complete a mission and the probability of success for a PCTL mission specification. The result is then used to model-check the safety constraints on the given control policy by checking if the success probability exceeds a given probability threshold. Formally, if \(\text{Prob}(s) > \alpha\) holds true, then the state \(s\) is said to satisfy the safety constraint.

Consider a three-state stochastic transition system where an agent starts in state \(s_0\). The objective is to reach \(s_1\) which is reachable with the probability of 0.8 at a given
Algorithm 6.1: Algorithm for conditional completion time $\tau(s \mid v), \forall s \in S$

Input: $\pi$
Output: $\tau(s \mid v)$ and $\text{Prob}^*$

1: Get $\text{Prob}$ for control policy $\pi$
2: $\tau_{\text{new}} \leftarrow 0$
3: repeat
4: $\tau_{\text{old}} \leftarrow \tau_{\text{new}}$
5: for all $s \in S$ do
6: $\tau_{\text{new}}(s \mid v) \leftarrow \sum_{s' \in S} T(s, s') \cdot \text{Prob}(s') \cdot (1 + \tau_{\text{old}}(s' \mid v)) / \text{Prob}(s)$
7: end for
8: until $\max|\tau_{\text{new}} - \tau_{\text{old}}| < \varepsilon$
9: return $\tau_{\text{new}}, \text{Prob}$

time. Likewise, the probability of reaching a danger state $s_2$ from $s_0$ is 0.1 and that of self-transitioning is 0.1. Both $s_1$ and $s_2$ are absorbing states. When the agent reaches $s_1$ after 3 consecutive self-transitions, then we can say that it achieved its goal after 3 time steps. However, when the agent reaches $s_2$ after some self-transitions, we cannot evaluate its completion time because the agent cannot escape from $s_2$. Since an average completion time has to consider all possible paths, we cannot calculate the average completion time for such a case where there is an absorbing fail state. Instead, we calculate a conditional average completion time such that we only consider the paths that were successful. With the proposed algorithm, the average conditional completion time is 1.111\ldots.

Equation 6.4 can be re-written in a different form where the completion time in a state depends on those in other states. Suppose the number of time steps is modelled with the sum of transition reward between successor states $r(s, s') = 1$ until an absorbing state is met (i.e., $r(s, s') = 0$ where $s'$ is an absorbing state). We use notation $r_t$ for reward gain at time $t$. We have the following derivation for $\tau(s \mid v)$:
\[ \tau(s \mid v) = E_\pi[\# \text{ of time steps} \mid s, v] \]
\[ = E_\pi[\sum_{k=0}^{\infty} r_{t+k+1} \mid s, v] \]
\[ = E_\pi[1 + \sum_{k=0}^{\infty} r_{t+k+2} \mid s, v] \] \hfill (6.5)
\[ = \sum_{s' \in S} P(s' \mid s, v) \cdot (1 + E_\pi[\sum_{k=0}^{\infty} r_{t+k+2} \mid s', v]) \]
\[ = \sum_{s' \in S} P(s' \mid s, v) \cdot (1 + \tau(s' \mid v)). \]

With Bayes’ theorem, the function can be re-written as the following:

\[ P(s' \mid s, v) = \frac{P(s, v \mid s') \cdot P(s')}{P(s, v)} \]
\[ = \frac{P(v \mid s') \cdot P(s \mid s') \cdot P(s')}{P(s, v)} \]
\[ = \frac{P(v \mid s') \cdot P(s') \cdot P(s' \mid s) \cdot P(s)}{P(s, v) \cdot P(s')} \]
\[ = \frac{P(v \mid s') \cdot P(s' \mid s) \cdot P(s)}{P(v \mid s)} \]
\[ = \frac{Prob(s') \cdot T(s, s')}{Prob(s)} \]
\[ = \frac{P(v \mid s)}{Prob(s}). \] \hfill (6.6)

Substituting Equation 6.6 into Equation 6.5 gives:

\[ \tau(s \mid v) = \sum_{s' \in S} \frac{Prob'(s') \cdot T(s, s') \cdot (1 + \tau(s' \mid v))}{Prob'(s)}. \] \hfill (6.7)

This equation is identical to Equation 6.4.
6.3 Synthesis of Optimal Control Policy

The objective of synthesis is to generate an optimal control policy $\pi^*$ with respect to the expected number of time steps, such that the probability of satisfying the mission specification is greater than a threshold $\alpha$:

$$
\pi^* = \arg\min_\pi \left[ \tau(s \mid v) \right]
$$

$$
s.t. \text{Prob}(s) > \alpha, \forall s \in S^a,
$$

where $\alpha$ is a probability threshold and $S^a$ is a set of non-absorbing states ($S^a = \{ s \in S \mid \exists s' \in S\setminus s, T(s, s') > 0 \}$). For example, the set of non-absorbing states for an ‘until’ operator (e.g., $\Phi_1 \mathcal{U} \Phi_2$) is $\text{Sat}(\neg\Phi_1 \land \neg\Phi_2)$ and the states for a ‘next’ operator (e.g., $X\Phi$) is $\text{Sat}(\neg\Phi)$.

We denote a lower bound of $\alpha$ as $\alpha_l$ and an upper bound as $\alpha_u$. A probability threshold $\alpha$ is guaranteed to have an optimal solution if and only if it is lower than the upper bound. The upper bound shows the maximum possible success probability threshold that each state $s \in S^a$ satisfies which can be computed with Equation 6.1. In other words, $\pi_u$ with $\alpha = \alpha_u$ is the most conservative path possible:

$$
\pi_u = \arg\max_\pi \text{Prob}(s)
$$

$$
\alpha_u = \min_{s \in S^a} \left[ \text{Prob}^{\pi_u}(s) \right].
$$

The lower bound $\alpha_l$ is found by solving Equation 6.8 with $\alpha = 0$:

$$
\pi_l = \arg\min_\pi \left[ \tau(s \mid v) \right] \left[ \text{Prob}(s) > 0, \forall s \in S^a \right]
$$

$$
\alpha_l = \min_{s \in S} \left[ \text{Prob}^{\pi_l}(s) \right].
$$

The control policy with the fastest completion time generated is guaranteed to perform with probability of satisfaction greater than $\alpha_l$. In other words, the expected completion time in each state cannot be less than that with $\alpha_l$.

The synthesis algorithm is shown in Algorithm 6.2. In this algorithm, we slightly
Algorithm 6.2: Synthesis algorithm for optimal control policy $\pi^*$ (dynamic programming)

**Input:** $\alpha$

**Output:** $\tau(s \mid \nu)$ and $\text{Prob}(s), \forall s \in S$

1: $\pi^*(s) \leftarrow \arg \max_s [\text{Prob}^\pi(s)], \forall s \in S$
2: Get $\text{Prob}(s)$ and $\tau^{\text{new}}$ from $\pi^*$
3: repeat
4: $\tau^{\text{old}} \leftarrow \tau^{\text{new}}$
5: for all $s \in S$ do
6: $\mathcal{A}' \leftarrow \{a \in \mathcal{A}(s) \mid \text{Prob}^a(s) > \alpha, \forall s \in S\}$
7: $\pi'(s) \leftarrow \{a \in \mathcal{A}'(s) \mid \arg \min_a \in \mathcal{A}(s) [\tau^a(s)]\}$
8: Get $\tau^{\text{new}}(s)$ and $\text{Prob}(s)$ from $\pi^*$
9: end for
10: until $\max[|\tau^{\text{new}} - \tau^{\text{old}}|]$ is small enough
11: return $\pi^*, \tau^{\text{new}}, \text{Prob}$

abuse the notations for $\text{Prob}^a(s)$ and $\tau^a(s)$ such that $\text{Prob}^a(s)$ and $\tau^a(s)$ are computed from $\pi'$ where $\pi'(s') = \pi(s'), \forall s' \neq s$ and $\pi'(s) = a$. It is straightforward to see that the algorithm produces a correct policy $\pi^*$ if and only if it exists, since the control policy is selected from the set of control policies subject to $\text{Prob} > \alpha$ and the number of elements in the set is finite.

### 6.3.1 Synthesis Example

In Figure 6.4, we demonstrate the synthesis result with the example in Figure 6.1. From Equations 6.9 and 6.10, the upper and lower bounds are computed to be 0.8 and 0.61. These values mean that there is no solution for $\alpha > 0.8$, and that $\alpha = 0.61$ is the probability of success of the fastest policy. For the given hard safety constraints of 0.5, 0.6, 0.7 and 0.799, the minimum probabilities of success are 0.6101, 0.6101, 0.7039 and 0.8, and the expected completion times given success are 7.61, 7.61, 11.50 and 29.99 steps respectively.

The control policies with safety constraints of 0.5 and 0.6 are identical since they are lower than the lower bound $\alpha_l = 0.61$. Hence selecting the control policy $\alpha = \alpha_l$
Figure 6.4 – Synthesis results with different safety constraints and the corresponding minimum probability of success guarantees that the fastest completion time is achieved with a probabilistic satisfaction guarantee. As the threshold increases, the control action in each state becomes more conservative. With the proposed method, the intuitive quantitative measure provides strong evidence of why the control policy is selected.
6.4 Summary

In this chapter, we have addressed bi-criteria objectives in regards to safety constraints. We have proved the analytical relationship between safety and completion time for a given PCTL formula. Then we presented model-checking and synthesis algorithms based on the proof. In model-checking, we determine if the robot model with a control policy exhibits a desired conditional completion time given a set of probabilities of satisfying a mission specification. Similarly, in synthesis, we find a control policy that minimises the conditional completion time given a safety threshold. The proof and the algorithms are novel for solving a bi-criteria objective using formal methods.

This chapter concludes the theoretical contributions of this thesis. The following chapter presents simulated examples of the algorithms presented in Chapters 4, 5 and 6.
Task Planning with Safety Constraints
Chapter 7

Applications of Task Planning with Constraints

This chapter presents simulated examples that support the theoretical contributions in Chapters 4, 5 and 6 in the domain of task planning with constraints. Sections 7.1 and 7.2 present examples for resource threshold constraints and fuel constraints in wind. Section 7.3 presents simulation examples with safety constraints. Section 7.4 summarises the chapter.

7.1 Resource Threshold Constraints

In this section we present results from two extended examples in the domain of planning for autonomous aircraft based on the algorithms proposed in Chapter 4. These examples demonstrate the use of resource threshold-PCTL (RT-PCTL) for practical applications, where constraints on resources are formally specified. In particular, the expressive power of the extended logic is presented and the RT-PCTL framework is compared with an existing method to show the uniqueness and importance of the framework. We also present an example with a large environment to demonstrate that the framework can be used for a relatively large system. In the first example,
we consider three scenarios for an autonomous thermal glider. The second example considers contingency planning for an autonomous solar-powered aircraft.

Suppose there is an autonomous thermal glider in a hexagonal grid-based environment that gains or loses altitude based on instantaneous thermal wind energy. The glider has precise \textit{a priori} knowledge of the time-invariant thermal energy distribution as shown in Figure 7.1a. The environment is discretised into a 10x10 hexagonal grid with six possible glider orientation values $d \in D$ where $|D| = 6$. A location-direction pair is denoted $(s, d)$. Each state is labelled with $\mathbf{S}$, $\mathbf{D}$, $\bar{\mathbf{S}}$ or $\mathbf{G}$ that denote \textit{safe}, \textit{danger}, \textit{semi-safe} and \textit{goal}. The task is formulated based on the labels. The glider fails the mission when: 1) it hits the boundary of the environment, 2) it enters a forbidden state, or 3) its altitude goes out of resource bounds $h = [0, 30], \forall s \in S$. The glider satisfies the mission when it completes its task without violating any constraints.

Figure 7.1b shows the dynamics of the glider in this scenario. There are three possible actions defined relative to current orientation. The altitude changes with the thermal energy in the environment and the action taken. The two actions that correspond to 60-degree turns lead to an additional relative altitude reduction of 0.1m, whereas maintaining the previous direction does not incur any relative loss of altitude. We assume that there exist unpredicted wind currents where the current on the xy-plane is much smaller than the aerial speed of the glider (i.e., the glider is not pushed too far or moving backward). Therefore transitions are stochastic. We also assume that the glider moves in its intended direction with probability 0.8 and moves 60 degrees to either side of the intended direction with probability 0.1. The transition uncertainty overestimates the probability of not following the intended path for demonstration purposes.

### 7.1.1 Reach the Goal and Avoid Danger

We consider an initial scenario where the glider is ‘to reach the goal state within 50 time steps while avoiding any danger states, maintaining minimum altitude of zero metres’. Note that the states with failure conditions are labelled as \textit{danger}. The
7.1 Resource Threshold Constraints

Figure 7.1 – Glider dynamics in a hexagonal grid. In (a), the shaded contours represent thermal energy. Brighter regions have upward airflow whereas darker regions have downward airflow. Numbers in the contour plot indicated state labels used in later examples. Numbers in the shaded scale indicate airflow velocity. The glider transition model is shown in (b).

RT-PCTL path formula for the specification is \([-DU \leq 50G]\). The corresponding state formula would be \(\mathcal{P}_{\max}^{x_0}[\neg DU \leq 50G]\) where \(x_0\) is the initial altitude of the glider. Note that the resource bound is \(h = [0, 30]\) for all states.

For the purpose of emphasising how RT-PCTL naturally works with continuous values, a piecewise-constant control policy is generated using a heuristic that is divided into risky and conservative actions based on the agent’s altitude. The risky action gives the most direct path to the goal position without considering the energy distribution along the path, whereas the conservative action gives the greatest expected return of instantaneous altitude increase. The risky action is taken when the accumulated resource exceeds the amount of resource required to take the most probable path to the goal state, otherwise the conservative action is taken. Note that such a heuris-
Figure 7.2 – Most probable path of a glider using the heuristic control policy in Equation 7.1, launched from minimum altitude at \((s_{68}, d_1)\) to reach the goal without entering danger states while maintaining altitude between 0 and 30 m.

tic does not necessarily produce an optimal policy; we synthesise and demonstrate the optimal policy later in this section.

The heuristic policy is defined as:

\[
\pi(s, x) = \begin{cases} 
\{ a \in A(s) \mid \text{‘Most direct route’} \} & \text{if } x > -\alpha \\
\{ a \in A(s) \mid \max_{d \in D} \sum_{s' \in S} P^{a}_{ss'} \cdot r_{s'} \} & \text{else}
\end{cases},
\]

(7.1)

where \(A(s)\) is the set of actions available at state \(s\) and \(\alpha\) is the sum of changes in altitude along the most direct path to \(G\) from \(s\) that avoids forbidden states.

The most probable path of the glider from \((s_{68}, d_1)\) starting with the initial altitude of 0 m is shown in Figure 7.2. Figure 7.3 shows the probability of mission success evaluated with discrete altitudes in comparison to PPFs. Probabilities were calculated using Algorithm 4.1. The PPFs in Figure 7.3 represent success probability over all possible paths from the given state while following the given piecewise-constant control policy, not just for the path in Figure 7.2. Although the control policy is deterministic, the transition model is probabilistic. Hence, there exists more than one path given a control policy over a mission horizon.

Discrete approximation evaluates success probability at discrete altitude values, in
7.1 Resource Threshold Constraints

(a) Probabilities of mission success at state \((s_{68}, d_1)\) within 50 time-steps

(b) Magnification around altitude 9.6m

**Figure 7.3** – Success probability with respect to entering altitude at state \((s_{68}, d_1)\) for the exact PPF solution compared with the discrete resource representation (a). The graph is showed magnified around altitude 9.6m in (b), where a sudden drop in success probability is revealed. The discrete cases fail to capture this pattern.

Contrast to the PPF with a piecewise-constant function. This difference is important because success probability can change abruptly with altitude. In Figure 7.3, the PPF value has a dip centred at 9.7m corresponding to a decision boundary in the policy. Above this range, the glider has sufficient energy to fly directly to the goal. Below this range, the policy directs the glider along a more energy-conservative path. Within this range, the policy takes the direct route but has high probability of failure. This is a good illustration of the benefit of the PPF representation because although the glider has reasonable direct control over lateral position, it has less direct control over altitude. The PPF captures safety-critical altitude conditions exactly, but discrete approximation in general does not.

To further illustrate this point, we chose three instances of entering altitudes (9.4m, 9.6m and 9.8m) for examination. For each altitude, the most probable paths are shown in Figure 7.4. Around 9.6m, the most probable path fails with the exact solution, whereas the path with discrete approximation gives a successful result. The sudden decrease in the probability that we observe in Figure 7.3 is evident in the exact solution whereas the discrete solution does not capture this decrease.

To yield the same degree of accuracy and to capture safety-critical behaviour with
Applications of Task Planning with Constraints

Figure 7.4 – Most probable paths at several starting altitudes (9.4m, 9.6m and 9.8m), with probability of mission success as indicated. (a-c) Resource values are discretised at resolution 0.5m. (d-f) Resource values are represented continuously using PPFs. The policy has a decision boundary at starting altitude 9.6m that is not captured in the discrete case. The incomplete path (i.e., path that failed to reach the destination) in (e) shows that the policy is likely to fail at this decision boundary, whereas (b) shows that the success probability is incorrectly evaluated in the discrete case. Such incorrect evaluation can result in catastrophic failure of the platform in practice.

discretisation, the altitude resolution would have to be infinitely small. Although there is a tendency that smaller resolution gives results closer to the result from the PPF, it is difficult to know in advance which resolution would be sufficient. For example, the minimum and maximum differences in two consecutive altitudes in the PPF are approximately 0.0677 and 1.0149 respectively. The resolution of 0.1 is too small for the minimum difference of approximately 0.0677 and the resolution of 0.01 is not able to represent continuous real values. The PPF approach can guarantee the absence of such safety-critical issues exactly.

Verification using PPFs is useful for robotics applications. Suppose we want to choose
a launch location by identifying the set of states that guarantees that the probability of holding the path formula $\neg \mathcal{D}U \leq 50 \mathcal{G}$ over a set of atomic propositions starting from the ground within 50 time steps is greater than 0.3. This set is denoted as

$$\text{Sat}\left(\mathcal{P}_{h > 0.3}^{\leq 50} \right) = \{(s_{17}, d_{2}), \cdots, (s_{47}, d_{6})\}, \quad (7.2)$$

where the set consists of 6 states satisfying the path formula.

We demonstrate the synthesis of an optimal control policy for this scenario in Fig-
Figure 7.6 – Most probable path using the optimal policy for the scenario in Section 7.1.2, launched from minimum altitude at \((s_{68}, d_1)\).

The policy is synthesised using Algorithm 4.3. Figure 7.5a and Figure 7.5b show the most probable path and PPF for the optimal control policy starting from \((s_{68}, d_1)\). The optimal control policy states that the glider should turn left when the altitude is below 7.09 m and should go straight when above. The result matches with our intuition that the glider would need to harvest more energy before approaching the goal when the altitude is low. Figure 7.5c shows an example of a state with three decision boundaries.

7.1.2 A Complex High-Level Mission Specification

In this scenario, the glider has a more complex high-level mission specification as shown in Figure 7.6. The mission is ‘to reach the goal position within 50 time steps such that the glider always travels along safe states or semi-safe states if the probability of reaching non-danger states in the next immediate time step is greater than 0.8’. The path formula for the specification is \(\left( S \lor (\bar{S} \land \mathcal{P}^{h}_{0.8}(X \neg D)) \right) \mathcal{U}^{\leq 50} \mathcal{G} \). The most probable path is shown in Figure 7.6 for the optimal policy.

We can choose a suitable launch configuration by examining the verification result. The set of states satisfying the mission with probability greater than 0.5 launched
from minimum altitude for the optimal control policies $\pi_1$ and $\pi_2$ is given by

$$\text{Sat} \left( P_{>0.5}^{h} \left( (S \lor (\bar{S} \land P_{>0.8}^{h} [X \land \neg D]) ) \right) U^{<50} G \right)_{\pi_1} = \{(s_{63},d_1), \ldots, (s_{75},d_2)\}, \quad (7.3)$$

where there are 16 states satisfying the path formula under the conditions. The satisfying states have two operation modes: 1) executing $\pi_1$ until reaching $G$ while satisfying this inner formula, and 2) executing $\pi_2$ to avoid an immediate visit to $D$. All the operation modes are guaranteed to satisfy the probability requirements (i.e., $>0.5$ and $0.8$).

### 7.1.3 A Sequencing Mission Specification

In this mission, the glider is to visit two locations, labelled $V_1$ and $V_2$. The specification in Equation 7.4 states that ‘the probability of reaching $V_1$ and being able to reach $V_2$ with probability greater than 0.3 from any altitude is greater than 0.8 from the minimum altitude’.

$$\Phi_3 = P_{\max}^{0.3} \left[ \mathcal{F}(V_1 \land P_{>0.3}^{s_{23}} \mathcal{F}(V_2)) \right]_{\pi_1} \quad (7.4)$$

Note that the control policy $\pi_1$ is for $\mathcal{F}V_1$ and the policy $\pi_2$ is for $\mathcal{F}V_2$, and $V_1 \in \mathcal{L}(s_{54},d_*)$ and $V_2 \in \mathcal{L}(s_{23},d_*)$. The most probable path using the optimal policy launched from $(s_{77},d_2)$ is shown in Figure 7.7. Again, the glider has two modes of operation: executing $\pi_1$ or $\pi_2$.

### 7.1.4 Altitude Regulation in Large Environment with Realistic Thermal Model

In this example, we demonstrate the specification of complex altitude regulations and the synthesis of an optimal controller complying with the regulations. We demonstrate the case with a thermal glider example where the glider has to approach a region of interest with sufficiently low altitude in order to take aerial pictures, where
Applications of Task Planning with Constraints

Figure 7.7 – Most probable path for the scenario in Section 7.1.3 using an optimal policy. The glider is launched from minimum altitude at \((s_{77}, d_2)\) to reach \((s_{54}, d_s)\), from which it flies to \((s_{23}, d_s)\).

the altitude for aerial photograph should be below 15 metres. The glider has strict regulations in the airspace that the regions between the start and the goal should be flown above 20 metres above the ground. Furthermore, the glider is not to fly through household regions. In this scenario, we use a larger environment with a realistic thermal model. The RT-PCTL formula over a set of atomic propositions for the mission is expressed as:

\[
\Phi_3 = P^6_{\max} \neg \text{house} \cup \text{photo},
\]

where \(h\) is the set of altitude ranges for each state \(s \in S\), \(\text{house}\) and \(\text{photo}\) denote the labels for house regions and regions for picture taking respectively.

In this scenario, we discretise the environment into a \(50 \times 50\) grid. Since there are 6-directions for each position, there exist 15000 states (i.e., \(50 \times 50 \times 6\)). The goal state is \(s_{714}\); hence \(h_{714} = [0, 15]\), since \(h_s\) denotes for the altitude range at state \(s\). For the middle regions with safety regulations, the UAV is required to fly between the altitudes of \(20\) m and \(30\) m (i.e., \(h_s = [20, 30]\)) while the rest is to fly between \(0\) m and \(30\) m (i.e., \(h_s = [0, 30]\)). The wind model used is Gedeon’s thermal model \([10]\), which is an extension of a Gaussian model. It models the circular vertical movement of airflow around the source of upward or downward thermal flow. Hence, the source of upward flow is surrounded by downward airflow, and vice versa. The equation for
Figure 7.8 – Thermal map modelled with Gedeon’s thermal model presented in Equation 7.7 [10]. The sources of thermals are surrounded by opposing flow. The UAV starting from the bottom-right corner is to reach the goal location at the top-left corner, while complying with the altitude regulation: fly above safety altitude in the middle regions, fly below certain altitude when approaching the goal, and not to fly over household regions shown in black blocks. The optimal trajectory of the UAV with respect to the probability of satisfaction complying with the regulations is shown as the grey line.

Gedeon’s thermal model is:

\[ w_z(r) = W \cdot \exp\left(-\frac{r}{R}\right)^2 \cdot \left(1 - \left(\frac{r}{R}\right)^2\right), \tag{7.6} \]

where \( W \) is the magnitude of the peak thermal, \( R \) is the radius of the source thermal, and \( r \) is the distance from the centre of the thermal origin. The equation can be re-written for the xy-plane as:

\[ w_z(x) = \sum_{i=0}^{N_w} W_i \cdot \exp\left(-\left(\frac{|X_i - x|}{R_i}\right)^2\right) \cdot \left(1 - \left(\frac{|X_i - x|}{R_i}\right)^2\right), \tag{7.7} \]
Figure 7.9 – The trajectory of the UAV is shown in 3D. The UAV starts from the bottom-right corner and stays around the upward thermals to gain enough altitude before moving to the middle region. The UAV then intentionally flies through downward thermals before approaching the goal with sufficiently low altitude.

where $\mathbf{x}$ is the position vector, $N_w$ is the number of thermal sources and $X_i$ is the location and $R_i$ is the radius of the $i$-th thermal source. The top-view optimal trajectory is shown in Figure 7.8.

Figure 7.9 shows that the optimal controller executed from the bottom-right corner gains altitude before entering the middle regions with safety regulations. Then, before approaching the goal region, the UAV intentionally travels through the regions with downward thermal to lose altitude. Complex regulations in the airspace are naturally expressed in a single framework, and the corresponding optimal controller is synthesised from the natural expression.

### 7.1.5 Surveillance and Contingency Planning with a Solar-Powered Aircraft with Emergency Flight Plan

Suppose we have a solar-powered autonomous aircraft that stores energy in a system of batteries. The robot gains more energy when flying in clear sky than when flying
**7.1 Resource Threshold Constraints**

- **(a)** Most probable long-term trajectory between $g_1$ and $g_2$
- **(b)** Most probable path starting from $g_2$ with initial battery level 30 percent
- **(c)** Most probable path continuing back to $g_2$
- **(d)** Most probable path continuing to $g_1$

**Figure 7.10** – Most probable path using an optimal policy for a solar-powered surveillance aircraft launched with 30 percent battery level from $g_2$ without the consideration of emergency flight plans at the synthesis level. Surveillance locations are shown in (a). Several snapshots of the mission are shown in (b-d).

Under clouds, when the battery is full, no further energy can be stored. There is an emergency battery pack, initially fully charged but with low capacity, that overrides the primary one in the event of a system failure.

The mission is to travel repeatedly between two locations and to be able to land safely in one of several emergency landing zones at any time with the secondary battery pack. The resource of interest is the battery level percentage. The mission fails when the battery level is zero, and the battery level cannot exceed 100 percent. The battery charges in sunny regions and discharges in dark regions. The resource bounds are $h = [0, 1)$ for all states except the target locations $g_1 \in \mathcal{L}(s_{43}, d_*)$.
Applications of Task Planning with Constraints

Figure 7.11 – Most probable path analogous to Figure 7.10, but with emergency landing locations as shown in (a)

and \( g_2 \in L(s_{78}, d_s) \), which have \( h = [0.3, 1) \) for safety purposes. The synthesis is to maximise the probability of successfully travelling to each target location within 30 time steps each. Control policy \( \pi_1 \) is for reaching \( g_1 \) and policy \( \pi_2 \) is for reaching \( g_2 \). Note that the synthesis only provides optimality for each control policy and its mission specification independently.

The synthesis specification \textit{without} considering the emergency flight plan is shown in Equation 7.8 and the most probable paths are shown in Figure 7.10. Note that the \( z \)-axis of the figure represents the battery level in this example. The long-term trajectory tends to spend more time in sunny areas than the dark areas to keep the battery level high enough. The glider has two modes of operation; it can choose to
execute $\pi_1$ to reach $g_1$ and execute $\pi_2$ to reach $g_2$.

\[
\Phi_4 = \mathcal{P}_{x;[0.3,1]}^{\pi_1} \left[ F^{\leq 30} g_1 \right] \land \mathcal{P}_{x;[0.3,1]}^{\pi_2} \left[ F^{\leq 30} g_2 \right] \tag{7.8}
\]

The surveillance mission with an emergency flight plan is specified in Equation 7.9 and its paths are shown in Figure 7.11. The goal of the synthesis is to reach emergency landing zones within a certain time using the emergency battery when the aircraft has encountered a system failure during its surveillance mission. The secondary battery has capacity equal to 15 percent of the primary battery and the probability of safely reaching the emergency zones within 10 time steps should be greater than 90 percent. The glider in this scenario has two modes of normal operations ($\pi_1$ and $\pi_2$) and an
emergency mode which is activated in the event of emergency ($\pi_e$). If a state is satisfied with the composite formula $\Phi_5$ for the given conditions (i.e., battery capacity), the glider can always execute the contingency plan during its normal operation if the resource meets the condition. We also assume that the switching to a contingency plan occurs fast that there is no delay.

$$\Phi_5 = \mathcal{P}^{x_{[0.3,1]}}_{\max} \left[ \mathcal{P}^{0.15:[0,1]}_{>0.9} \left[ F_{\leq 10} e \right]^{\pi_e} \mathcal{U}_{\leq 30} g_1 \right]^{\pi_1} \land \mathcal{P}^{x_{[0.3,1]}}_{\max} \left[ \mathcal{P}^{0.15:[0,1]}_{>0.9} \left[ F_{\leq 10} e \right]^{\pi_e} \mathcal{U}_{\leq 30} g_2 \right]^{\pi_2} \right)$$  \hspace{1cm} (7.9)

Unlike the long-term trajectory without contingency planning, the trajectory in Figure 7.11a passes closer to the darker areas where the emergency landing zones are located. The synthesis naturally balances between maintaining the battery level and the safety of reaching emergency zones when maximising the probability of satisfaction of the mission.

Figure 7.12 shows the PPFs of the aircraft moving from $g_2$ to $g_1$ with and without the emergency plan considered at the synthesis level. The PPF with the emergency plan is computed analytically whereas that without the plan is simulated by randomly introducing failure and observing how well the aircraft reaches the contingency destination as soon as the failure occurs ($F_{\leq 10} e$). The probability of satisfying the surveillance mission as well as the emergency procedure is much greater when the control policy is synthesised with the emergency plan. The plan without the consideration of contingency plans may be more efficient in normal operation, but the figure shows that such a plan would fail miserably in the event of an emergency. Note that the PPFs provide a quantitative performance guarantee on satisfying both the normal and contingency missions.
7.2 Fuel Constraints with Wind

In this section, we present examples with a petrol-powered UAV flying at fixed air-speed and constant altitude based on the algorithms presented in Chapter 5. We demonstrate how a high-level planner and a low-level controller are coupled in a hybrid system and how the system satisfies the high-level task in the presence of wind. More importantly, we demonstrate the feasibility of online synthesis. We also demonstrate the use of different algorithms for low-level controller.

The size of the environment is $2000m \times 2000m$ and the wind field is interpolated using Gaussian process regression without uncertainty. The environment is shown in Figure 7.13 where 10 wind observation points are demonstrated with bold arrows. The environment is discretised into a $10 \times 10$ grid. The airspeed of the UAV is $10m/s^{-1}$, the set of control inputs is $U = \{-4, -2, 0, 2, 4\}$ deg $s^{-1}$ and the UAV-specific constant $C_a$ is 0.001.

![Figure 7.13](image)

**Figure 7.13** – Wind vectors drawn on the environment sized 2000m by 2000m. The vector field is interpolated with Gaussian process regression from 10 observation locations.
Figure 7.14 – Comparing two different algorithms in the same problem environment. The goal of the UAV is avoid the danger regions while approaching the goal region from the runway region. (a) uses GBA to minimise fuel consumption with 15 discrete steps and (b) takes a path with the minimum number of sequence with 13 discrete steps. The amount of fuel left at mission completion is 78.249\%(a) and 78.216\%(b).

7.2.1 Reach Goal while Avoiding Danger with Direction Constraint

We consider an initial scenario where the UAV is ‘to avoid danger regions until reaching goal region where the goal region has to be approached from runway region’. With no landing procedure (i.e., $\alpha = 0$), the mission specification is written in LTL as

$$\phi_1 = \neg (\text{goal} \lor \text{danger}) U (\text{runway} \land \Box \text{goal}).$$  \hspace{1cm} (7.10)

The environment is labelled with symbolic propositions on the discrete states appropriately and the synthesis algorithm is executed starting from $[120m, 320m, 30\text{ deg}]^T$ in Figure 7.14. For the purpose of comparison, we demonstrate two trajectories with different algorithms: one with GBA with minimum fuel consumption is shown in
Figure 7.14a and the other that takes the sequence with the minimum number of discrete states is shown in Figure 7.14b. The numbers of discrete steps to accomplish the mission are 15 and 13 steps for GBA and the other respectively, however the proportions of fuel left are 77.30% and 72.56%. The difference in average head wind along the two trajectories is 0.5\,ms^{-1}. The efficiency benefit can be visually observed since the trajectory with GBA follows the wind flow to minimise the effective air distance whereas the other algorithm often goes against the wind.

The approximate flight distance in this example is 2600m, which is relatively short compared to more practical flight distances. For large aircraft operating over a long distance, a small increase in fuel efficiency results in substantial savings. For instance, the fuel capacity of a Boeing 737-300 is 19131\,kg [37]. A large amount of fuel would be saved even with a small percentage increase in efficiency.

Also, it is important to note that the difference in the average head wind in this example (approximately 0.5\,ms^{-1}) is very small compared to the UAV’s airspeed (10\,ms^{-1}). In practice, the wind fluctuates over space and time significantly [1]. Larger gains in fuel efficiency are possible where the average head wind along a fuel-efficient trajectory is much lower than that of the trajectory with shortest ground distance.

### 7.2.2 Surveillance Mission

In this scenario, we demonstrate a surveillance mission where a number of locations of interest must be visited infinitely often and a landing procedure begins when the fuel goes below 20%. The LTL formula over a set of atomic propositions is written as

\[
\phi_2 = \Box((\neg e_{20\%} \Rightarrow (\bigwedge_i N_g \Diamond \text{goal}_i)) \land (e_{20\%} \Rightarrow \Diamond \text{land}))
\]  

(7.11)

where the goal regions are at \(s_{33}, s_{72}\) and \(s_{86}\), and the landing base is at \(s_{29}\). Figure 7.15 shows the result of synthesis for this mission.

In Table 7.1, we show the average clock time to perform synthesis with different
Figure 7.15 – A continuous execution of the surveillance mission encoded in Eqn. 7.11. The UAV visits three regions and begins to reach the landing zone when the fuel goes below 20%. The sequence of discrete states between regions is optimal w.r.t. the fuel consumption based on the approximated wind vector. The continuous trajectory follows the sequence by selecting the best control action at each discrete state.

numbers of discrete states using a standard laptop computer. The environment is divided into grids from $10 \times 10$ up to $100 \times 100$ while keeping the cell size the same ($200 \times 200m$). With a reasonably large number of discrete states such as listed in Table 7.1, synthesis can be performed in a plan-as-you-go manner. Note that we have gained a significant efficiency at the cost of losing completeness. In this way, new plans are synthesised one after the other during execution. The UAV is only required to wait for the initial synthesis, and as long as the total synthesis time shorter than the flight time, the UAV does not wait for the completion of synthesis when transiting to a new Büchi state. Updated wind estimates could also be incorporated in the process.

### 7.2.3 Large Scale Scenario

In this scenario, we demonstrate the algorithm with a large scale environment and discuss the feasibility of using the method in plan-as-you-go manner. The xy-size of the environment shown in Figure 7.13 is multiplied by 10 while preserving the same
Table 7.1 – Task-level synthesis time for different numbers of discrete states is shown for the problem in Section 7.2.2. The size of the environment is enlarged while preserving the cell size (200 x 200m). Discrete synthesis and continuous synthesis refer to synthesising a state sequence for a single Büchi transition and a continuous trajectory for the sequence. Average flight time refers to the average time taken to complete one Büchi horizon.

<table>
<thead>
<tr>
<th>Number of States</th>
<th>Average Synthesis Time (s)</th>
<th>Average Flight Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discrete</td>
<td>Continuous</td>
</tr>
<tr>
<td>100</td>
<td>0.151</td>
<td>28.806</td>
</tr>
<tr>
<td>400</td>
<td>0.941</td>
<td>58.065</td>
</tr>
<tr>
<td>2500</td>
<td>14.137</td>
<td>161.942</td>
</tr>
<tr>
<td>10000</td>
<td>101.257</td>
<td>404.761</td>
</tr>
<tr>
<td>40000</td>
<td>777.445</td>
<td>1419.364</td>
</tr>
</tbody>
</table>

Section 7.2.2 shows that the hybrid system approach with value iteration may not work for an online synthesis, since the partial synthesis time (i.e., synthesis of value iteration) is too long with respect to the flight time (i.e., actual flight time with the controller with partial synthesis). Figure 7.16 shows the accumulated travel and synthesis time with respect to the number of iterations with 10000 states. The bold black line illustrates the elapsed travel time at an iteration, the dotted line illustrates the accumulated synthesis time where re-planning occurs when required, and the grey line illustrates the accumulated synthesis time where the controller is synthesised in a plan-as-you-go manner. An iteration refers to solving for a continuous trajectory in a single discrete state of the system. If the goal state is reached, the time taken for partial synthesis is included in the accumulation for the iteration. For example, at this 30-th iteration, the accumulated travel time elapsed at the iteration is approximately 650s, and the accumulated time taken to synthesis at the iteration is about 300s. Note that if the travel time is less than the synthesis time, then we cannot run the algorithm in a plan-as-you-go manner.

Figure 7.17 shows the timelines for partial synthesis and execution of the two scheduling methods: plan-when-needed and plan-as-you-go. When the UAV starts, it has to wait for a certain amount of time for the synthesis of its first plan. For plan-when-
Applications of Task Planning with Constraints

Figure 7.16 – Accumulated elapsed travel and synthesis time with respect to iterations, with 10000 discrete states. Elapsed travel time represents the accumulated total travel duration at $i$-th iteration, and synthesis time represents the accumulated total synthesis time at $i$-th iteration. The grey line represents the synthesis time using plan-as-you-go manner, and the dotted line represents the synthesis time in which re-planning occurs when needed. Jumps in synthesis time represent the partial synthesis time (approx. 109 seconds). After the 8-th iteration, the elapsed travel time always exceeds the synthesis time with plan-as-you-go manner; allowing for an online execution.

Figure 7.17 – Timeline for partial synthesis and execution. Two ways of scheduling synthesis are presented: plan when required in Figure 7.17a and plan-as-you-go in Figure 7.17b. Partial synthesis includes the synthesis for the sequence of states to follow and the generation of the continuous trajectory.

(a) Timeline of partial synthesis and execution. A new controller is synthesised when reached the end of current Büchi horizon. The UAV has to wait during the synthesis.

(b) Timeline of partial synthesis and execution. A new controller is synthesised in plan-as-you-go manner. No time is wasted between the executions. The longer the synthesis, the more time wasted.

needed, a new controller is synthesised when the end of current horizon is reached. As a consequence, the UAV has to wait until synthesis is done. If the waiting period
is too long, the UAV may reach the regions that violate the specification, since it has no plan to execute. Therefore, the plan-when-needed method is dangerous for a system with a significant number of discrete states. In contrast, for this plan-as-you-go method, a new controller is synthesised straight after the previous synthesis. Therefore, given that the synthesis time is shorter than the execution time, the hybrid system approach with GBA can be used for relatively large systems for online execution.

### 7.2.4 GBA with Rapidly-Exploring Random Tree (RRT)

We demonstrated the use of GBA with value iteration, where the environment is discretised into a set of states in a grid. We then searched for the discrete sequence minimising the fuel consumption in wind, while satisfying the given specification. We have shown in Section 7.2.2 and 7.2.3 that the inherent bottleneck for an online synthesis is with the number of discrete states. Furthermore, the synthesised trajectories using a grid do not reflect how robots would move in practice (e.g., robot moves rectilinearly).

In this section, we present the use of GBA with other motion planning algorithms. Since GBA is not tied to any specific navigation algorithm, we could easily replace the low-level controller part of the hybrid controller with other algorithms. In particular, we demonstrate a preliminary result with RRT. Note that the result does not consider the dynamics of robots for simplicity, and no branch would be made in undesired regions. Figures 7.18 and 7.19 show the synthesised trajectories for the scenario in Section 7.2.2 where the objective is to visit the goal locations repeatedly. In Figure 7.18, the rapidly-exploring random tree (RRT) planner stops when a newly created node is in a goal location, and the trajectory leading to the node is stored. In the next Büchi horizon, the RRT planner runs in the same way. In Figure 7.19, the RRT planner creates at least $10^4$ nodes and finds the trajectory with the minimum fuel consumption for a given Büchi horizon. Intuitively, the trajectory in the latter RRT planner generally moves to avoid head wind.
We demonstrate the time analysis of partial synthesis in Table 7.2 for three RRT modes: ‘fast’ for stopping as soon as reaching the goal, ‘$10^4$’ and ‘$10^5$’ for the corresponding minimum number of nodes to create. It is clear that the computation time for partial synthesis is significantly reduced compared to the value iteration approach in Section 7.2.2.
Table 7.2 – Time analysis for partial synthesis with RRT. There are three modes: ‘fast’ for stopping when the goal is reached, ‘$10^4$’ and ‘$10^5$’ for the corresponding minimum number of nodes to create. For each mode, 1000 samples are taken. All times are shown in seconds.

<table>
<thead>
<tr>
<th></th>
<th>Fast</th>
<th>$10^4$</th>
<th>$10^5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.2232</td>
<td>2.9179</td>
<td>131.9108</td>
</tr>
<tr>
<td>Max.</td>
<td>1.2776</td>
<td>4.0739</td>
<td>142.2374</td>
</tr>
<tr>
<td>Upper Quartile</td>
<td>0.2193</td>
<td>2.9017</td>
<td>132.5981</td>
</tr>
<tr>
<td>Median</td>
<td>0.1559</td>
<td>2.8718</td>
<td>131.3309</td>
</tr>
<tr>
<td>Lower Quartile</td>
<td>0.0925</td>
<td>2.8420</td>
<td>130.0636</td>
</tr>
<tr>
<td>Min.</td>
<td>0.0454</td>
<td>2.7617</td>
<td>124.6683</td>
</tr>
</tbody>
</table>
Applications of Task Planning with Constraints

Figure 7.19 – RRT-variant of the scenario in Section 7.2.2, where the number of nodes must be greater than $10^4$. The algorithm seeks for the trajectory with the minimum fuel consumption. The trajectory generally avoids head winds.

7.3 Safety Constraints

This section presents outdoor simulation examples of the algorithms presented in Chapter 6. We demonstrate how the synthesis algorithm is used in practice with two such examples. In each example, we tune the level of required safety to show
the changes in the completion time. Suppose there is an autonomous ground vehicle in an outdoor environment with a number of geometrical constraints. The vehicle has precise \textit{a priori} knowledge of static features in the environment as shown in Figure 7.20. The environment is discretised into a $10 \times 10$ grid where each state is labelled with $S$, $W$, $C$, $D$ and $G$ that denote \textit{safe}, \textit{water}, \textit{mud}, \textit{slope} and \textit{goal} respectively. The labels represent \textit{‘no constraint’}, \textit{‘unable to operate due to water’}, \textit{‘slippery ground due to mud’}, \textit{‘danger due to steep slope’} and \textit{‘destination’} respectively. The \textit{slope} label is given to a state with a slope angle greater than 50 deg. The probabilistic computation tree logic (PCTL) mission specification given to the vehicle is written based on the labels.

The vehicle has four possible actions $A$ in all states: \textit{up}, \textit{down}, \textit{left} and \textit{right} with respect to the absolute geometry. On dry ground, the vehicle moves in its intended direction with probability 0.8 and moves $\pm 90$ deg to the intended direction with probability 0.1. When the vehicle is on muddy ground, the transition uncertainty decreases to 33.3\% for three directions. The vehicle is bounced back to its starting position when it hits the boundary. This transition function is simple, but could be replaced by sophisticated mobility prediction methods.

![Figure 7.20 – 10x10 simulation environment with steep slope (red), water (blue), mud (brown) and goal (green) states. Safe states do not have any colour.](image-url)
7.3.1 Avoid Slope and Water until Reaching the Goal State

We consider an initial scenario when the vehicle is ‘to reach the goal state while avoiding any slope and water’. The formula for the specification over a set of atomic propositions is \([-D \land \neg W \U G]\). From Equation 6.9 and 6.10, we have \(\alpha_u = 0.5946\) and \(\alpha_l = 0.0136\) respectively.

Figure 7.21 demonstrates the control policy synthesis results for hard probability thresholds of 0.1, 0.2, 0.5 and 0.59 respectively. The minimum probability of success increases from 0.1065, 0.2168, 0.5004, to 0.5924, which all satisfy the hard safety
7.3 Safety Constraints

constraints specified. As the required level of safety increases (i.e., from 10%, 20%, 50% to 59%), the maximum expected completion time also increases from 21.927, 22.750, 26.442 to 81.238. The synthesis has successfully returned the fastest control policy while all states holds true for the hard safety constraint.

7.3.2 Avoid Slope, Water and Mud Adjacent to Water until Reaching the Goal State

Suppose there has been a report from the field operator that the muddy ground is likely to become swamp if it is next to water. Thus the robot should not visit any states labelled mud if it is adjacent to a state labelled water. The mission is expressed as $[\neg D \land \neg W \land \neg (C \land EX W)] U G$. Note that the state formula $EX \Phi$ means ‘there exists at least one path in which the next state satisfies $\Phi$’. The upper and lower bounds are $\alpha_u = 0.7796$ and $\alpha_l = 0.0054$ respectively, and the optimal policy is shown in Figure 7.22.

From $\alpha = 0.2$ to $\alpha = 0.77$, the minimum probability of success increases from 0.2601
to 0.7741 while the maximum expected completion time given success increases from 22.3811 to 144.4481.

### 7.4 Summary

This chapter has demonstrated the results of the algorithms proposed in Chapters 4, 5 and 6. We have demonstrated the use of RT-PCTL for expressing complex altitude constraints in autonomous thermal gliders and solar-powered aircraft, and shown the efficiency of GBA using examples of petrol-powered aircraft. We also have shown how safety and completion time are quantitatively measured and used in practice for synthesis.

This chapter provides encouraging results towards the feasibility of applying formal methods in realistic environments. Due to lack of expressivity in constraints, the inherent limitations of formal methods have restrained the application areas in robotics to toy problems, but we have presented algorithms that overcome the limitations in practical applications. For example, we have shown that altitude regulations (set by government organisations) can be expressed formally for autonomous thermal gliders. We also have demonstrated a simulation example of a petrol-powered aircraft that uses real models of aircraft dynamics and wind interpolation. With the example of an autonomous ground vehicle, we have shown how to consider different properties in rough environments, such as slopes, mud and water, to make a safe decision. These results show the potential to extend formal methods for outdoor task planning with constraints, and inspire future research in the field.
Chapter 8

Conclusion

In this section, we present a detailed summary of the contributions of the thesis. The section concludes with a discussion of possible future improvements to the algorithms shown, and outlines directions for future work.

8.1 Thesis Summary

This thesis has identified and addressed two problems within the overall vision of task planning for autonomous outdoor robots: problems with resource and safety constraints. First, the thesis proposed an extension to probabilistic computation tree logic (PCTL) called resource threshold-PCTL (RT-PCTL) that naturally specifies a mission requirements with complex constraints on resource bounds. With RT-PCTL, the value of the accumulated resource is part of the formal specification. Secondly, the thesis proposed a task-planning method with realistic control dynamics and physical assumptions. It presented an online algorithm called greedy Büchi algorithm (GBA) that reduces the problem size to avoid the scalability problem for fuel and wind constraints. The algorithm synthesises a hybrid system that reacts to changes in the fuel level. Lastly, the thesis proposed an algorithm for optimal synthesis of a controller with respect to completion time given minimum safety constraints. The algorithm naturally balances between completion time and safety. The algorithms presented
are formally analysed to show their theoretical properties, and are validated with simulated examples using realistic models of environments.

The thesis has described existing work in robotic task planning, and addressed the limitations in applying the work for outdoor robots. It has identified two core problems for outdoor robots and has successfully built the theoretical foundations for further research in multi-constrained task planning where a number of different outdoor constraints are considered in a single framework with hard measures of guaranteed performance.

8.2 Contributions

8.2.1 Resource Threshold Constraints

In Chapter 4, we have presented an extension to PCTL for systems with continuous-valued resource threshold constraints and stochastic transitions. We introduced the piecewise-constant control policy and presented algorithms for model-checking a given policy against a formal specification and performance guarantee, and for automatically generating an optimal piecewise-constant control policy. We validated our theoretical results through simulated examples of autonomous aircraft in multiple scenarios including contingency planning.

The examples demonstrate the significance of our results. We showed examples with complex task specifications that cannot be expressed in other forms of PCTL, and provide a level of confidence in the aircraft’s ability to complete its mission without knowing in advance the exact path the aircraft will follow through symbolic high-level states. Model-checking in our method provides a performance guarantee that applies to a piecewise-constant control policy where continuous-valued energy resources are represented exactly.
8.2 Contributions

8.2.2 Fuel Constraints in Wind

In Chapter 5, we have presented an efficient synthesis algorithm for complex UAV tasks involving constraints on the operational state of the robot under realistic physical assumptions. We illustrated the behaviour of this algorithm through two examples where the UAV performs navigation and surveillance tasks in a static continuous wind field with fuel constraints. Our simulation results indicate that synthesis is fast enough to allow for replanning during long-duration tasks where wind estimates may evolve over time.

8.2.3 Safety Constraints

In Chapter 6, we used formal methods to address the inherent trade-off between risk and reward in stochastic motion planning problems. We presented novel model checking and synthesis algorithms for PCTL-specified tasks that do not simply maximise success probability, but instead minimise completion time within a given success threshold. Our algorithms provide quantitative measures that avoid the problem of manually choosing a weighted cost function and allow for the natural formulation of complex task specifications in Markov decision processes (MDPs).

8.2.4 Applications

In Chapter 7, we have demonstrated the use of the theoretical contributions in realistic scenarios. We have shown that RT-PCTL is capable of formally specifying resource constraints. Specifications that were not possible to be expressed are specified using formal language and provably-correct controllers are synthesised automatically in a push-button manner. We also have shown that GBA is efficient enough for an online execution with realistic models of the environment and platform. We have evaluated the importance with synthesis time analysis, and demonstrated that the proposed framework using GBA can be extended very easily with other types of motion planning
algorithms. Lastly, we showed the importance of quantitative balancing of safety and completion time with two simulated examples.

8.3 Future Work

There are a number of natural extensions to the approaches and algorithms presented in the thesis. Such extensions include the combinations of multiple methods presented in a unified framework, operations in dynamic and uncertain environments, the use of more complex models, guaranteed satisfaction for constrained problems, and algorithmic improvements.

8.3.1 Unified Frameworks

- **Resource Threshold and Safety Constraints:** The model-checking and synthesis algorithm for RT-PCTL presented in Chap. 4 is based on a Markov decision process (MDP). Since the analytical relationship between safety and completion time proved in Chap. 6 is also based on the MDP formulation, the RT-PCTL framework can be extended naturally to include safety constraints, where the extended framework balances between the probability of success and conditional completion time, with the consideration of resource threshold constraints. This natural extension could be an important milestone towards the overall vision of task planning for outdoor robots. Outdoor robots are often constrained to more than one variable. The unified framework aims to analytically solve the task planning problem with two most important constraints for autonomous outdoor robots.

The unified framework would be useful for the synthesis of controllers for resource-constrained systems where the level of guaranteed safety is quantitatively specified and tuned for faster completion of the task. Suppose we have the scenario from Section 7.1.4. The safety factor could fit into the mission of taking aerial pictures, so that the glider sacrifices a portion of safety for urgent completion
of a mission. The example specification in English could be ‘while guaranteeing success probability of 90%, approach the forest with altitude below 15 metres. Middle regions should be passed with altitude above 20 metres.’

• **GBA with Other Motion Planning Algorithms**: In Section 7.2.4, preliminary results of using GBA with rapidly-exploring random tree (RRT) were presented, where the value iteration-based low-level controller is replaced with an RRT planner. Likewise, other types of sampling-based motion planning algorithms could replace the motion planner for algorithmic benefits such as guaranteed performance and anytime synthesis. For example, we could replace the low-level controller with an anytime motion planner to find a valid trajectory at any time, and RRT* to converge to an optimal trajectory asymptotically. It is important to note that GBA has great potential in its flexibility to incorporate any motion planning algorithms. The overall hybrid system can benefit from the algorithmic properties of the motion planning algorithms.

### 8.3.2 Dynamic and Uncertain Environments

• **RT-PCTL in Dynamic and Uncertain Environments**: RT-PCTL represents an important step towards the grand goal of complex mission specifications for stochastic systems, but the environment considered is still strictly deterministic. It is important to extend RT-PCTL for dynamic environments where state labels change over time. With this extension, robots with resource constraints can be used in applications such as search and track tasks.

The resource in RT-PCTL is modelled as a deterministic real value. In practice, the resource value is often represented with a probability distribution. An interesting future work would be to replace the deterministic resources with those represented with probability density functions, allowing for synthesis of a safer controller in more uncertain environments.

• **GBA in Dynamic Environments**: The hybrid system approach with GBA re-plans the sequence when the new horizon starts. In the re-planning phase,
the low-level motion planner can be synthesised based on the current environment settings. Therefore the changed properties in the environment can be considered naturally in the re-planning phase. For example, changes in wind or movements of obstacles can be considered. Since synthesis with a changed environment is not different to that with an unchanged environment, the computation time for synthesis would remain the same. With existing task planning approaches, adapting to changes in the environment requires synthesis of the overall controller which is not tractable for online synthesis. This extension opens the door to task planning in dynamic environments, and can be very useful for outdoor robots.

- **GBA and MDP with Stochastic Transitions**: In Chapter 5, the value iteration algorithm is used for low-level control. Since the algorithm is based on the MDP formulation, it can easily be extended to the case of transition uncertainty. The extension could be used to represent a stochastic wind field, where wind vectors vary within a certain range which could be modelled with a stochastic transition model. In the formulation, the states causing Büchi transitions are considered as the goal states and the objective of the MDP is to minimise expected fuel consumption for reaching one of the goals. The formulation with stochastic transitions can be naturally extended to include safety constraints. For example, the objective of the MDP formulation could be to minimise the expected fuel consumption while preserving safety above 80%.

The extension naturally provides the notion of stochasticity to linear temporal logic (LTL) which is capable of expressing tasks where the events occur in linear time. There exists a number of methods for providing stochasticity to LTL but this extension is more useful, since the extension is based on GBA which is efficient compared to the existing methods.

- **Safety Constraints with Extended MDPs**: The algorithms for safety-constrained planning are based on MDPs, but can be extended to more powerful variations such as mixed-observability MDPs (MOMDPs) and partially-observable MDPs (POMDPs). Practical implementation of temporal logic meth-
ods, including ours, requires further work in developing methods for state reduction and approximation, for evaluating problems with large state spaces, and for handling complex transition models such as those provided by statistical mobility prediction techniques. Our work on safety constraints has a great potential, since there are a large number of motion planning algorithms and frameworks that are based on the MDP formulation. This work can be fitted naturally into the algorithms and frameworks for further expressivity in safety measured quantitatively.

8.3.3 Complex Models

**Fuel Consumption Models:** The form of the Breguet range equation in Section 5.2 assumes constant altitude, temperature, and UAV velocity. However, the equation could easily be replaced with other forms of this equation that treat these parameters as variables. For instance, a more complex fuel model considering altitude can be used for commercial aviation where the rules in airspace regarding altitude should be expressed and executed in a fuel-efficient way. More sophisticated models of UAV dynamics, such as point-mass models, could also be introduced in future work. As already discussed in Section 8.3.1, the framework using GBA is flexible enough to work with any motion planner and model of low-level control. Therefore, the framework is not only limited to use Breguet range equation but also other more complex equations (e.g., fuel models for unmanned underwater vehicles).

8.3.4 Guaranteed Satisfaction

- **Qualitative Measure of Satisfaction for LTL:** The output of synthesis using formal methods guarantees the satisfaction of the given specification and environment settings. The synthesised controller is conservative for the worst-case. However, there are a large number of variables that constrain outdoor robots. Therefore the synthesis for the worst-case may not be useful in practice. Instead, it is important for synthesis to provide a quantitative measure
of satisfaction, rather than the absolute satisfaction. The work presented in Chapter 6 is a pioneering result in this direction, and can be further extended for outdoor applications.

- **GBA with Guaranteed Landing Procedure**: In Section 7.2.2, the reactive task of the UAV is to execute a landing procedure when the fuel level is below a certain threshold. However, the execution does not guarantee that the aircraft lands with fuel remaining, since the algorithm currently does not guarantee the satisfaction of the overall mission. A fragment of GBA could be formed to guarantee the contingency part of the mission. The extension would guarantee the safety of the UAV implicitly.

### 8.3.5 Algorithmic Improvements

**Completeness of GBA**: The proof for the overall completeness of GBA is not presented. Instead, we provide a proof of completeness for partial synthesis. There is a constraint in GBA that the visited Büchi state should not be re-visited before entering the accepting state, to avoid staying in a loop. However, there may be no possible transition from one Büchi state to another depending on the environment settings. Further research on algorithmic improvement for completeness of GBA is required. A possible extension is to check the regions in the environment against the Büchi automaton for reachability to accepting states. Updating the algorithm for overall completeness is important, since the outdoor applications of interest are usually safety-critical problems where the tasks have to be satisfied.
Bibliography


