Motion Planning and Stochastic Control with Experimental Validation on a Planetary Rover

Rowan McAllister, BE (Hons 1) BSc

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

THE UNIVERSITY OF
SYDNEY

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

August 2012

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.

Rowan McAllister, BE (Hons 1) BSc

31 August 2012

Abstract

Planetary rovers are required to safely navigate across unstructured and hazardous terrain with increasing levels of autonomy. Autonomy is necessary to react to impending danger because remote control has been proved challenging and dangerous for planetary rovers due to the large communication delays. Safety is especially a concern in space applications where a robot is unaccompanied throughout its entire mission. Unstructured terrain poses several types of hazards to a robot such as getting stuck, toppling or scraping against rocks. In a dense collection of localised hazards, platform safety is very sensitive to deviations from an intended path. To maintain the safety of the platform during navigation, planetary rovers must consider control uncertainty during motion planning. Thus, not only does the system need to make predictions of action outcomes, it also needs to estimate the *accuracy* of these predictions. The aim of this research is to provide planetary rovers with the ability to plan motions to goal regions that optimise the safety of the platform by including information about the accuracy of its controls. Modelling such control uncertainty is difficult due to the complex interaction between the platform and its environment. In this thesis, we propose an approach to learn the outcome of control actions from experience, represented statistically using a Gaussian Process regression model. This model is incorporated explicitly in the planning process using dynamic programming to construct a control policy for navigation to a goal region. Motion planning strategies are considered that take into account different types of uncertainties, including uncertainty in distance, heading and yaw of the platform, across various motion primitives. The approach is implemented on a holonomic rover with six wheels and a Rocker-bogie frame and tested on a Mars analogue terrain that includes flat ground, small rocks, and non-traversable rocks. Planning is computed over a terrain map built using an on-board RGB-D camera. We report the results of 200 simulated and 95 experimental trials that validate the approach. These results demonstrate that explicitly incorporating knowledge of control uncertainty into the motion planning process increases platform safety by decreasing the likelihood of the rover getting stuck and reducing the cost accumulated over the executed path. Accounting for heading uncertainty resulted in the most significant increase in platform safety.

Acknowledgements

I wish to sincerely thank both my supervisors Robert Fitch and Thierry Peynot for their constant guidance and support. This work would not have been possible without their continued assistance, and I have learned so much under their supervision.

A sincere thanks also to my work colleagues Ali, Esa, Ken and Angela whom helped build, maintain and debug the rover test platform. Being part of this team, and interfacing this research with other aspects of the rover gave me a great sense of satisfaction, a great source of learning and a lot of fun.

I would also like to acknowledge the Pathways to Space program of the Powerhouse museum in Sydney, for their use of the 'Marscape' which allowed for testing in a realistic Martian environment, and maintained interest in the project as a public exhibition. Also Salah Sukkarieh for heading the rover project.

Lastly I would like to thank my family for their consistent love and support.

"I have approximate answers and possible beliefs and different degrees of certainty about different things, but I am not absolutely sure about anything"
- Richard Feynman

To Mum, Dad, Marion and Emily

Contents

List of Figures

List of Tables

Nomenclature

a	action
A	action set or action space
s	state
S	state space
Δs	change in state
δs	error in change of state
δs_{head}	error in change of state by heading
δs_{dist}	error in change of state by distance
δs_{yaw}	error in change of state by yaw
$p(s'\|s,a)$	control uncertainty model
$P(s'\|s,a)$	transition probability function
$R(s',s,a)$	reward function
α	chassis configuration angles
θ	pitch
ϕ	roll
ψ	yaw
Θ	hyperparameters of the GP
$\boldsymbol{\lambda}$	terrain profile features / characteristics

DP	dynamic programming
GP	Gaussian process
MDP	Markov decision process

Chapter 1

Introduction

1.1 Objectives

Autonomous planetary rovers have the potential for persistent operation, reacting in real time to dangerous situations without the need for frequent manual intervention [37]. Operating rovers via remote control, by contrast, is limited by high communication latency. To enable persistent autonomous exploration, rovers must be capable of robust and reliable motion planning that considers uncertainty.

In unstructured environments, motion planning algorithms must consider various forms of uncertainty. One significant source of uncertainty in outdoor terrain is *control uncertainty*. Robots such as planetary rovers are designed for mobility in challenging environments, and understanding the associated control uncertainty for the purpose of motion planning is difficult due to the complexity of this type of environment. It is critical to consider control uncertainty in motion planning, particularly in environments that expose the robot to the risk of serious mechanical damage. We are interested in this problem in the context of planetary rovers [46]. An example of a planetary rover is shown in Fig. 1.1. This thesis addresses navigation of autonomous ground vehicles traversing unstructured and potentially dangerous terrain, optimising platform safety with respect to control uncertainty.

Figure 1.1 – Planetary rover "Mawson" used for experimental validation, shown in the Mars yard at the Powerhouse Museum in Sydney, Australia. All experiments were performed in this environment.

The goal of classical geometric motion planning is to plan a sequence of motions, from start to goal, that minimise time or distance travelled while avoiding obstacles [35]. The conceptual distinction between free space and obstacles for planetary rovers on unstructured terrain, however, is not clear. One definition of *obstacle* in the context of unstructured terrain is a region that is impossible to traverse or would cause mechanical damage to the robot. Such obstacles are not directly observable, though a continuous metric of risk to the platform's safety can instead be derived using a kinematics model to predict a relative likelihood of instability. This situation cannot be modelled by simple distance thresholds surrounding obstacles because risk varies across free space as a continuous metric.

The result of a control input is never certain due to imperfect actuation and imperfect knowledge about robot-terrain interaction. Thus a robot will not follow an intended path exactly, which can be disastrous in an environment cluttered with high risk regions. A knowledge of control uncertainty is necessary to avoid control commands where, due to control uncertainty, the robot would have a high likelihood of deviating into a high risk region. As deterministic planning algorithms do not account for deviations from the computed path, they expose a robot to considerable risk.

Accurately predicting executed behaviour in response to a given control input is difficult for planetary rovers due to complex terramechanics [20]. For previously unobserved terrain, prior models of terrain properties may not be available. It is therefore important to model control uncertainty with a method that can be feasibly executed online during operation of the robot, and to validate such a model experimentally.

To reach this goal, the objectives of this thesis are to:

- Develop a motion planning framework that considers control uncertainty to increase safety of an autonomous platform.
- Provide a method to estimate control uncertainty of the platform. Realistic estimates of control uncertainty in new situations are only possible by observing the relation between control uncertainty and the local environment in previous motions. Thus, control uncertainty will be learned from previous traversals of terrain.
- Validate that the methods proposed increase the safety and the reliability of a real platform when traversing a high-fidelity Mars-analogue environment.

1.2 Motion Planning and Control Uncertainty

The aim of this motion planner is to compute control inputs that guide the robot safely towards a goal region given a representation of the surrounding environment. A motion policy[1] is used to consider the expected long-term consequences of each potential deviation. A policy is required, rather than a path, since potential deviations of the robot cannot be anticipated due to the uncertainty of control input outcomes. The motion planner predicts the outcomes of control inputs from specified positions together with a measure of uncertainty on each prediction using a terrain-robot dynamics model. This model is trained on various terrain, and then used for predictions given an assessment of the immediate environment at a position.

[1] A motion *policy* is a many-to-one mapping of each robot position in an environment to the control input that is to be executed from that position.

Our approach builds a statistical model of control uncertainty directly from observed behaviour, represented as a Gaussian process (GP). We consider uncertainty in the heading of the platform and in distance travelled. We use this GP model to build a stochastic transition function for use in motion planning. The planning goal is to compute a policy that allows the robot to reach a given goal location while maintaining the safety of the platform. We assume that a map of the environment is available, represented as a digital elevation map. Platform safety is represented by a cost function over this terrain map, which is constructed *a priori* using on-board sensors. We compute the policy using dynamic programming (DP), where the resolution of discretised geometric states is equal to that provided in the elevation map.

This is a general approach to motion planning which considers uncertainty in control in the planning phase to compute safe paths over unstructured terrain. A number of research groups have proposed frameworks for autonomous ground vehicles with similar aims. However, few of these account for control uncertainty in the planning phase, nor address appropriate means of training a model of control uncertainty for realistic predictions of action outcomes. Existing motion planning research under uncertainty has not yet been extended to cases of unstructured terrain outdoors. This gap between realistic perception and good motion planning under explicit uncertainty has not been tackled before on unstructured terrain, although much attention has been given to simulated environments [2, 7, 10, 19, 30, 31, 42].

In this thesis, we present the details of our approach and its implementation for the planetary rover shown in Fig 1.1, which was introduced originally in [39]. The environment consists of flat terrain, traversable rocks, and non-traversable rocks. GP models are learned for rock traversal, mapping environment features to a distribution of resulting rover configurations (in state space) for two types of control actions. The cost map is constructed from data collected by an RGB-D camera on-board the rover. We report the results of 200 simulated trials and 95 experimental trials that evaluate the rover's ability to traverse flat terrain and small rocks while avoiding non-traversable rocks. We compare the performance of the rover in executing policies constructed with and without control uncertainty. Our results show empirically that planning

with control uncertainty improves the rover's ability to navigate while avoiding non-traversable areas. These results demonstrate the value of planning under uncertainty for planetary rovers using a real platform in a realistic environment.

1.3 Contributions

The main contributions of this thesis are an approach to motion planning that considers control uncertainty to improve the safety of an autonomous platform, and validation of this approach on *real* unstructured terrain, supporting its suitability for complex environments in the real world.

The specific contributions are:

- a stochastic transition model based on Gaussian processes,
- an instantiation of the model using an experimental Mars rover platform traversing single rocks,
- a motion planning algorithm based on dynamic programming to maximise platform safety,
- a simulation of the motion planner using learned data,
- hardware experiments that demonstrate the full system.

1.4 Thesis Structure

Chapter 2 is a review of related work. First, motion planning algorithms are reviewed in the context of a planetary rover system. Multiple sources of uncertainty are then considered, and implications of these sources of uncertainty on motion planning are discussed. Finally, methods to incorporate control uncertainty into motion planning are discussed, as well as how models of uncertainty have been learned from experience.

Chapter 3 introduces key algorithms used in the development of our technical approach. The DP algorithm is first discussed, outlining how optimal control policies are computed, followed by an example. The GP model is then presented, and inference using GP is discussed.

Chapter 4 outlines the approach chosen in this thesis for motion planning under stochastic control. This approach is general and applicable to any ground robot. The choice of DP as a search algorithm is first discussed, including the model of control uncertainty as a stochastic transition function. Learning mobility prediction with the use of Gaussian processes is then discussed. Finally, a high-level description of the whole system is presented.

Chapter 5 discusses implementation details of the approach used on the particular robotic platform used for experimental validation. Details of the platform, how it is modelled and specification of the state space are given. The source of data chosen to train the GP is discussed.

In **Chapter 6** an experimental validation is presented. Results from motion planning in simulation and in real experiments on flat terrain and on rough terrain are given and discussed. Results show smoother behaviour and an increase in executed safety of the platform when planning considering control uncertainty.

Chapter 7 provides a conclusion and discusses future work.

Chapter 2

Related Work

A typical autonomous terrain traversal system is shown in Fig. 2.1 [4]. This system observes the environment using sensors, which pass data to the rover's software components (yellow) to decide the next action command or path for a controller to follow. In turn, the controller sends control commands to the motors and actuators on the rover platform. Sensor data is used to determine the rover's localisation and also to build a terrain model. The terrain model is a representation of the rover's environment. It is used to predict rover-terrain interaction throughout the environment and for building a cost map. A cost map is used to quantify the difficulty of traversability at locations throughout the known environment. This cost depends on vehicle knowledge - a knowledge of constraints about how the rover can move and operate in an environment. The motion planner uses the vehicle knowledge, a specification of the goal region to traverse to and input from the localisation and cost map to decide the next appropriate action command to send to the controller. The aim of the motion planner is to send a series of action commands, directing the rover to its goal location, in a way which optimises an objective function such as accumulated cost or safety of the platform.

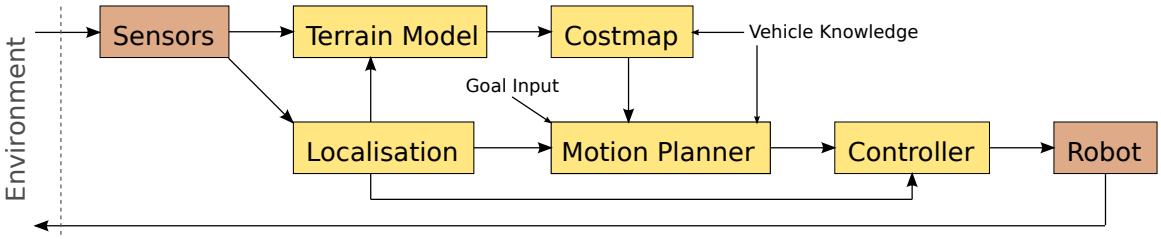


Figure 2.1 – A typical planetary rover system. Hardware components shown in red, software components shown in yellow.

2.1 Motion Planning

Planning motions of a robot through geometric space that avoid collisions with obstacles necessitates a knowledge of the robot's geometry. The robot's *configuration* (vector) is a specification of the values of each of its degrees of freedom, which determine the geometric space the robot occupies. Configuration space is the set of all configurations possible. This is often expressed as $C_{obstacle}$ for configuration states that result in a contact with an obstacle, and C_{free} otherwise. Using this abstraction to plan in configuration space rather than geometric space allows generalised motion planning algorithms to be used, which are independent of how the robot is constructed. Planning for point robots in known C-space has been studied extensively [33, 35]. The general motion planning problem of finding a shortest collision free path, known as the piano-mover's problem, is PSPACE-complete.

Sampling-based planners

Sampling-based algorithms are able to find feasible paths in high-dimensional spaces efficiently. Feasible paths are collision free but not necessarily shortest. Sampling-based algorithms are not complete: they cannot detect if a solution does not exist in finite time. However, if a solution does exists, the planner will eventually find it. This weaker type of completeness is known as probabilistic completeness [27].

Probabilistic Roadmap (PRM) [28] is a sampling-based method that consists of two steps: construct a roadmap and then query the roadmap. A roadmap is made up of

- vertices: states which exist in free space (robot not in contact with an obstacle),
- edges: the result of an action which would cause the robot to transition continuously through state space from one state to another without collisions.

The roadmap is constructed by sampling the configuration space many times (randomly, or according to a pattern), discarding samples which are not in C_{free}, then using a local planner to attempt to connect to nearby local configurations. The roadmap is queried using any graph search algorithm, and may terminate once a collision-free path exists between start and goal configurations (or possibly keep searching for lower cost paths). Because the roadmap can be queried for various start and goal configurations, PRM is known as a *multi-query* method.

Rapidly-exploring random trees (RRT) [34] is another, widely used sampling based planner [7, 42], suited to high-dimensional spaces. RRTs compute open loop paths by randomly selecting new configurations in configuration space, and use a local planner to compute a path to a close configuration on the existing tree to the random sample. Due to constraints of the robot, the path may not link the two configurations exactly, but nevertheless the tree grows. This has the property of efficient 'space filling': finding a path (albeit non-optimal paths) such that a path can be found between the root-configuration to most other configurations (or at least close by). The RRT algorithm is guaranteed to be non-optimal, however the recent development of the RRT* algorithm [24] has been proven to probabilistically converge towards an optimum path-solution with increased sampling. An RRT admits a single start and goal configuration, and thus is a *single-query* method.

Grid search algorithms

Grid search is a family of algorithms that plan over a discretisation of both configuration-space and action-space. Reachability trees, A* (or similarly D*) and dynamic programming are resolution-complete grid search algorithms. Resolution-completeness means that the solution is computed with respect to the given discretisation.

Reachability trees are arguably the simplest grid-search algorithm. Motion plans are computed by building and traversing a tree structure of configuration-nodes using Breadth First Search (BFS) which finds shortest paths with (and only with) unit edge weights. The root node is the robot's start configuration, and node expansion involves assessing the result of each motion primitive available to the robot. Each motion primitive results in a new configuration child-node. The process stops when a goal-configuration is reached. The path returned is the path of least possible motion primitives the robot requires in moving from start to goal.

A* and Dijkstra's algorithm find shortest paths with non-negative, non-unit edge weights and are thus more general than BFS. A* also uses a heuristic function to guide a search, making it much more efficient in large search domains. Dijkstra's algorithm is a special case of A* that uses a constant-zero (uninformative) heuristic function. As such, A* is more general and efficient than BFS and Dijkstra's algorithm, and a common choice for motion planning algorithms [17, 31]. D* is a similar but dynamic version of A*. A major drawback of reachability trees, Dijkstra's algorithm, A* and D*, however, is configuration-transitions are assumed to be deterministic. These algorithms do not consider consequences of uncertainty in transitions explicitly. In addition, these algorithms compute only a single path and not a complete motion policy which is more relevant to a robotic system which will inevitably deviate from an intended path.

Markov decision processes (MDPs) are commonly used to formulate problems in motion planning with uncertainty [35, 36], treating uncertainty explicitly and providing optimal solutions with respect to the model of uncertainty. Control uncertainty is represented as a stochastic transition function $P(s'|s,a)$, describing a probabilistic distribution of resultant states as a function of an original state and action selected. An optimal policy can be computed using $P(s'|s,a)$ with dynamic programming [2]. However, these techniques are most often evaluated in simulation only and there is a critical need for further validation using real robots.

The Hamilton-Jacobi-Bellman (HJB) equation [23] from optimal control theory is a continuous calculus of dynamic programming, solving the search explicitly with ordi-

nary differential equations (ODEs). Such a technique can be far more efficient than iterative computation, but unfortunately so far, a general technique for solving associated ODEs has not been found. Solutions have been to some simple systems whose dynamics are described by linear differential equations (e.g. mass-spring systems). This can be extended to Linear-Quadratic control problems, which additionally identify a cost to minimise which is a quadratic functions of state-deviation and control input.

2.2 Uncertainty

Uncertainty can describe a lack of knowledge or inability to detect precisely the state of an entity or the outcome of a process. From an agent's point of view, there are multiple possibilities of what the true state of an entity is, or what the outcome of a process will be. Quantification of uncertainty, with sufficient data, can be represented mathematically using probability density functions [1]. Different types of uncertainty have been identified [1, 41]:

1. *Ontological*: Irreducible or true randomness, e.g. quantum mechanics where ground truth itself is probabilistic,

2. *Epistemological*: A lack of full knowledge about the universe (possibly including the robot itself). This includes knowledge of the state of the universe, and knowledge of physical processes. A subtype is *systematic* uncertainty: something can be known in theory, but is not considered in practice. For instance, a model neglecting certain information about reality, in order to simplify, or limited by computational capacity, and as a result predicts differently to reality.

3. *Aleatoric*: Limits of the precision of knowledge or detection capability to verify that two control inputs are identical. For instance, repeated trials of a system that records successive control inputs as identical will actually result in different outcomes due to undetectable differences between successive control inputs.

This thesis is concerned with both Aleatoric and Epistemological uncertainty. The following subsections consider the different forms of uncertainty that exist in the rover system (Fig. 2.1) and the related work that mitigates risk and cost to a rover by incorporating knowledge of each uncertainty into the motion planner.

2.2.1 Localisation Uncertainty

Localisation uncertainty implies that a robot does not know exactly where it is in relation to the world around it. If a robot cannot be exactly sure of where it is, the risk of collisions with known obstacles is increased. When motion planning operates in a global frame of reference, this uncertainty and associated risk of collision will increase with time if the localisation system is proprioceptive. The severity (or variance) of this uncertainty is dependent on the localisation algorithm. Typically, environments with many distinct features allow for lower localisation uncertainty [12].

Considering localisation uncertainty in motion planning involves planning in belief space. Belief space is a probabilistic distribution over possible states. The partially-observable Markov decision process (POMDP) [22] is a common solution for planning in belief states according to the MDP model of the environment [21, 30]. Optimal actions in the POMDP framework are those which maximise the average expected utility of executing the action from each possible state the robot is in, weighted by the level of belief (probability) that the robot is in that state. POMDPs are complete and provide optimal motion policies, although do not scale as well as other methods including PRM-adapted approaches for searching belief space [44]. Other methods plan according to *anticipated* localisation error in possible future configurations. Given a path towards some future configuration, the localisation algorithm can be modelled according to known features it would see, and thus a prediction of localisation error could be computed. Such a method was proposed in [10] by including the predicted localisation error in the cost function.

2.2.2 Environmental Mapping Uncertainty

When an object is sensed, e.g. using a laser range finder, the distance and/or direction measured will vary slightly from the true value due to imperfect sensing. A similar case arises if a robot is given an imperfect map of its environment. As a consequence, a robot cannot be exactly sure of:

1. How far it is from sections of the terrain (terrain-robot relation),
2. The topology of terrain (terrain-terrain relation).

Examples can include point clouds where individual points have associated position uncertainty [49]. If uncertainty exists in the representation of the terrain, a robot planning motions over such terrain will be uncertain about its interaction with that terrain. The robot will therefore be uncertain about the incurred risk and cost at locations within the elevation map.

Methods to compute uncertainty in terrain include incorporating a sensor model [48]. In addition, estimation of unseen terrain structure with uncertainty is possible with Gaussian process regression [32]. This is particularly useful in unstructured terrain, of which observations usually contain shadows caused by occlusions. A motion planning method in [43] uses GP regression of unstructured terrain to compute elevation-uncertainty at unobserved locations. This involved selection of foothold locations during traversal of a legged robot to a goal region. By including the elevation-uncertainty of terrain into a cost function, a greater reliability of selecting safe footholds was achieved compared with an *Interpolated Elevation Grid* terrain model that 'fills in the gaps' with no uncertainty. Methods to deal with environmental mapping uncertainty in motion planning have involved frameworks of probabilistic obstacles, where an obstacle is known to exist, but the location of that obstacle is only *approximately* known [11, 16, 40]. These motion planners reason about the probability of each configuration being an obstacle. This was implemented using PRMs, querying least-cost paths where the cost of each configuration is a function of the belief that the configuration is in $C_{obstacle}$.

2.2.3 Control Uncertainty

Due to imperfect actuation, an action command that the controller receives from the motion planner will not be executed in an exactly predictable way. Thus, from a state the rover is in before executing an action, the resultant state is not known with certainty *a priori*. In regards to motion planning, this means a robot can never be certain about the exact trajectory it will actually execute.

Control uncertainty can be considered in motion planning using a stochastic transition function. This stochastic transition function expresses the probability of arriving at a resultant state s' from an original state s and action a selected by: $P(s'|s,a)$ which can be incorporated into motion planning explicitly. DP can use this model of control uncertainty to compute control commands which are optimised with respect to this model to avoid high risk regions [2]. Control uncertainty is the type of uncertainty this thesis addresses, and a discussion of how control uncertainty is incorporated into motion planning is given in Sec. 2.3.

2.3 Incorporating Control Uncertainty in Motion Planning

Incorporating control uncertainty in motion planning can be avoided using 'safety factors': a minimum distance a planned path needs to be from an obstacle, such that even if the robot slips or skids unexpectedly, it is unlikely to collide with an obstacle. Such an approach first involves the expansion of obstacles using Minkowski addition [50] to account for the shape of the robot [2], and a further obstacle expansions by the 'safety factor' amount in each direction. Such an approach can work well in indoor environments, where configuration space can often be classified as C_{free} or $C_{obstacle}$. Additionally, safety factors pose artificial limits on what would otherwise be C_{free}, such that in cluttered environments, it may remove the possibility that a solution can be found at all. If unstructured terrain is modelled in this fashion, it often represents a very cluttered environment, and thus such an approach systematically

discards many feasible paths. Conversely, in unstructured terrain we may wish to express difficulty of traversal by a continuous metric, otherwise such an approach is neither optimal nor robust with respect to a measure of difficulty, nor the uncertainty model of the platform.

The rest of this section presents previous research on motion planning with control uncertainty. Major considerations in such research are:

- choice of planning algorithm,
- how control uncertainty can be incorporated into the planning process,
- how that control uncertainty is modelled and/or learned.

One approach is to avoid regions the platform estimates would incur higher uncertainty in control. This avoids the problems of dealing with uncertainty by avoiding regions of higher uncertainty. This has been done by expressing uncertainty as a cost, where higher uncertainty incurs higher cost, and then planning a path that minimises this cost assuming deterministic control [17]. However, by 'avoiding the problem', such an approach can fail in terrain where most states incur considerable control uncertainty. In addition, such a cost function is slightly 'misguided': high uncertainty does not necessarily imply high risk of collision, nor vice versa. A fundamental requirement of motion planning is to avoid obstacles, and such a cost function does not optimise against probability of collisions.

Modelling uncertainty of control can be done implicitly by modelling a feedback controller along a complete pre-selected path, or explicitly during the development of the path itself [27]. Linear-Quadratic-Gaussian (LQG) controllers have been used to model potential deviations from a pre-computed path to consider the consequences from potential deviations [7, 10, 15, 19, 21, 42]. However, this treats motion planning and control uncertainty as decoupled: candidate paths are selected, and afterwards evaluated in light of a stochastic control model. Several iterations may result in some rejected paths until a suitable path is found. Whilst this methodology can provide paths that satisfy a minimum-threshold of safety, the path selected is not optimised

with respect to control uncertainty. Modelling control explicitly, by contrast, treats the problems as coupled. Once the robot has deviated from its path, the best option may be to take an *alternative* route. This can be accomplished by solving an optimal control *policy* which does not assume (nor force) a robot to commit to a pre-computed path, once deviations inevitably occur in the execution of its motion strategy.

Additionally, an implication of using LQG controllers to model uncertainty in the controls is that the model assumes the degree and nature of uncertainty in controls is homogeneous throughout all the environment. While this could be suitable in homogeneous environments, it is not a realistic assumption on unstructured terrain. Additionally, these methods were validated in simulation only, and not unstructured terrain. Learning mobility, and consequently control uncertainty, directly from experience of previous traversals over terrain is more realistic. Techniques to predict mobility of a robotic platform have been achieved using *terramechanics* and *near-to-far learning*. These methods have been used to estimate uncertainty in outcome of motion-primitives and to estimate slip. Slip is the difference between wheel angular velocity and linear velocity of the wheel centre divided by wheel radius, normalised by the maximum of both terms [5]. Any slip value other than zero is often undesirable.

Physics-based approaches that study terramechanics provide detailed mobility models by considering features such as soil cohesion and density [18]. These methods have been developed and tested for rovers on real terrain, not just simulation. An advantage of these methods is feature identification: motion is predicted based upon the values of a small and relevant set of features selected from a physical understanding of the rover-environment interaction (e.g. soil cohesion, soil density, wheel traction etc.). This can reduce the number of parameters that require calibration during a learning phase to predict rover motion. Statistical mobility prediction using terramechanics has been proposed that generates a Gaussian distribution over predicted future states on homogeneous terrain [20], and to predict slip [19]. These methods are well suited to locally flat and homogeneous terrain of a known slope. However, it is difficult to precisely model non-homogeneous terrain that includes different grades of sand and rocks of different sizes and shapes. In addition, it is extremely difficult to model how

rocks may *move* in reaction to the force exerted by a rover wheel.

Recent work has modelled mobility as a function of terrain in a self-supervised learning framework called 'near-to-far' learning [9, 29]. These approaches use observations when 'far away' to classify terrain types according to the associated proprioceptive mechanical properties encountered when *on* ('near') that same terrain. While this approach does not explicitly learn a model of control uncertainty, this learning framework could be suitable for that purpose, and could be used to learn continuously even when deployed on missions. Inference using regression, however, would have the advantage over classification used in this technique of a less abrupt change in behaviour near the decision boundaries of the classifier.

Research done by [3, 25] learned to estimate slip based on visual and geometric information of unstructured terrain not yet traversed. Work in [25] uses terrain slope and control inputs to estimate wheel slip using GPs. This method does not utilise the uncertainty in slip explicitly, rather it creates a mobility map of velocity limits as a function of estimated slip and uses deterministic A* to solve for least cost path. However, this framework for slip estimation could be extended to provide a model of uncertainty in resultant state and thus control uncertainty. The use of non-parametric learning methods such as GPs in the context of estimating mobility in unstructured terrain has the advantage over parametric models that it makes fewer assumptions about the environment and environment-robot interaction [26].

Integrating a realistic model of control uncertainty into the planning process requires a non-deterministic planning algorithm. Methods that either avoid regions of uncertainty, decouple control and planning, or use parametric models of control uncertainty suffer from lower expectation of robot safety because they make strong assumptions of the environment. Despite these drawbacks the aforementioned methods have some advantages which are summarised in Sec. 2.4, and used to inspire our approach presented in Chapter 3.

2.4 Conclusion

The literature shows that incorporating a model of uncertainty into motion planning increases the safety of a platform. Safety is additionally optimised, with respect to the uncertainty model, when the model is *explicitly* incorporated into motion planning. A realistic uncertainty model necessitates training. This is especially true for unstructured terrain where the degree of control uncertainty can vary widely, and is not homogeneous. Effective methods of self-supervised learning could be used to inference control uncertainty, using regression instead of classification. For unstructured terrain, we argue that learning without a complex terramechanics model, nor any parameterised model, is required. The use of GPs as a non-parametric learning algorithm is a suitable choice in the context of motion prediction over unstructured terrain because it makes fewer assumptions about the complex interaction between the environment and the robot than parametric models.

Chapter 3

Background

This chapter introduces two key algorithmic ideas used in this thesis: dynamic programming and Gaussian processes.

3.1 Dynamic Programming

Dynamic programming computes optimal control policies given a complete description of an environment. In particular, DP can be used to compute the solution to a problem formulated as an MDP. Using DP in this way assumes the environment is Markovian, localisation is accurate and control is either deterministic or stochastic.

3.1.1 Overview

An MDP is defined by a finite set of states S, a set of actions available $A(s)$ executable at each state $s \in S$, a transition probability function $P(s'|s,a)$ which outputs the probability that executing action a from state s will result in the agent being in state s', and a reward function $R(s',s,a)$ which outputs the 'reward' an agent would receive by executing action a from state s which results in state s' [47].

Dynamic programming operates by computing a value function $V(s)$ for each $s \in S$, which describes the 'desirability' of an agent being in each state s if the goal of the

agent is to transition to a goal state in S. Default values are given to all states initially, and various algorithms exist that compute the optimal value function $V^*(s)$. The optimal value function in the MDP framework satisfies the Bellman optimality equation:

$$V^*(s) = \max_a \Big\{ \sum_{s'} P(s'|s,a)(R(s',s,a) + \gamma V(s')) \Big\}, \tag{3.1.1}$$

where s is the state of an agent, a is an executable action at state s and $0 < \gamma \leq 1$ is the discount factor. γ determines how important future rewards are to an agent relative to immediate rewards. Given the Bellman optimality equation (Eq. 3.1.1), the optimal policy $\pi^*(s)$ is computed:

$$\pi^*(s) = arg\max_a \Big\{ \sum_{s'} P(s'|s,a)(R(s',s,a) + \gamma V(s')) \Big\}, \tag{3.1.2}$$

A more extensive description of the various algorithms in DP, and methods to increase their efficiency can be found in Sutton and Barto's book (1998) [47]. This thesis uses the value iteration method, detailed below, to construct an optimal motion planner.

3.1.2 Value Iteration

Multiple methods exist for computing the convergence of the value function at each state in S to satisfy Eq. 3.1.1. Value iteration is one such method.

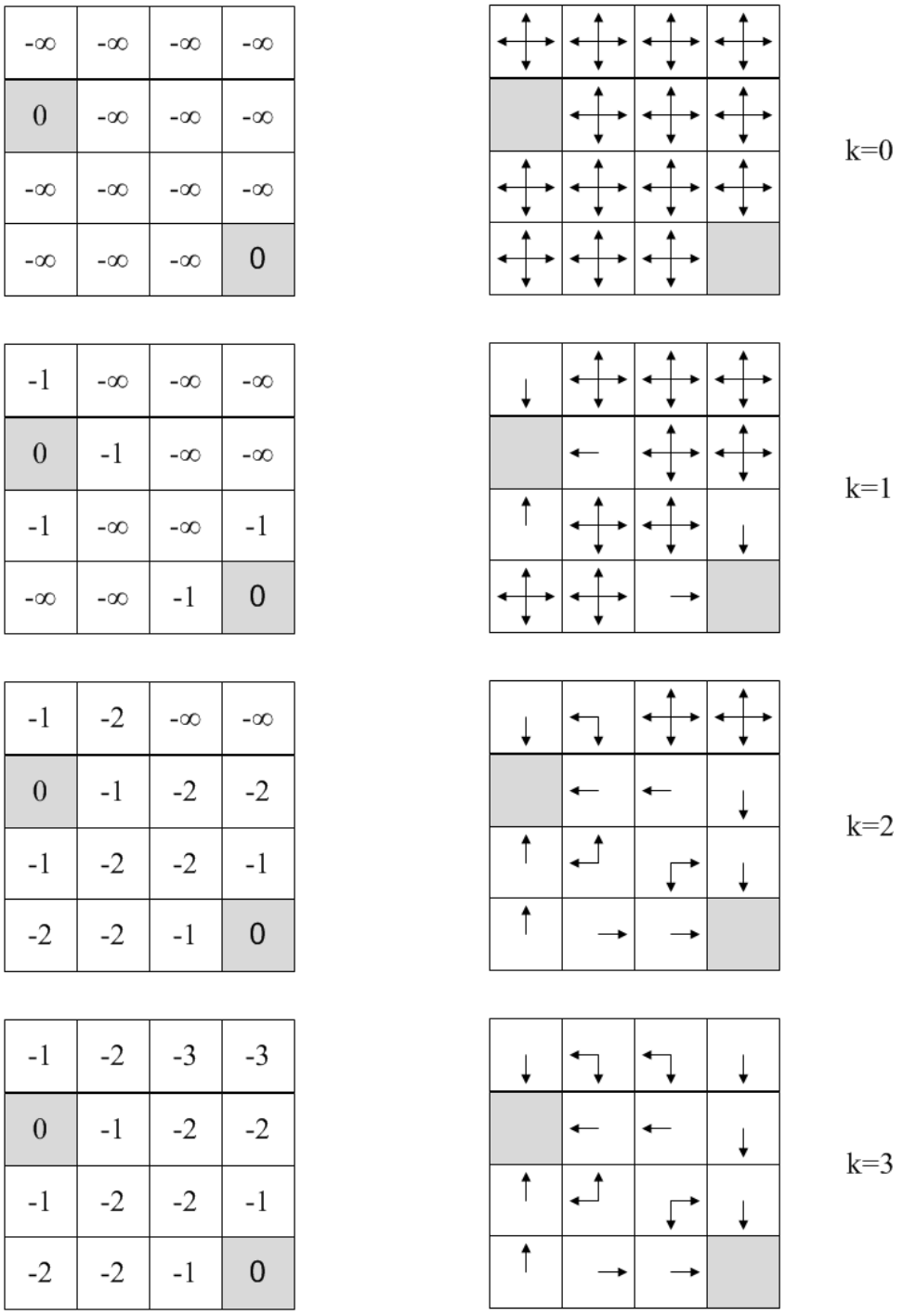
Pseudocode of value iteration is presented in algorithm 3.1. Value iteration begins by initialising the value of all states in S to $-\infty$, except for goal states which retains fixed values of zero. Goal states may comprise a single state, or multiple states, perhaps all state that lie within a *goal region*. Values of each state are updated iteratively, and once all state values have stabilised (condition on line 9), the iteration terminates, and the optimum policy is returned. The optimum policy is a greedy search of the highest value state (line 10). At each iteration (while loop: lines 2 - 9) the value of each state is updated according to the current-iteration or previous-iteration value of resultant states s' it can reach, by execution of each action $a \in A$ (line 6). This update (line 6), assesses the maximum value of state s according to each action $a \in A$

that can be executed at state s. For each action, the 'value of state s if action a is taken' is computed by a probabilistically-weighted reward and value of the resultant state. The maximum value is selected, and the value function is updated for state s (line 6).

Algorithm 3.1: Dynamic programming: Value evaluation, (edited) from Sutton and Barto [47]

1: Initialise $V(s) = -\infty \forall\ s \in S$
2: **while** $\Delta < \theta$ (a small positive number) **do**
3: $\quad \Delta \leftarrow 0$
4: $\quad$ **for** each $s \in S$ **do**
5: $\quad\quad v \leftarrow V(s)$
6: $\quad\quad V(s) \leftarrow max_a \sum_{s'} P(s'|s,a)[R(s',s,a) + \gamma V(s')]$
7: $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
8: $\quad$ **end for**
9: **end while**
10: Output: $\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s',s,a) + \gamma V(s')]$

An example of iterative value function convergence, using algorithm 3.1, is shown in Fig. 3.1 with corresponding policy improvement in Fig. 3.1(b). This shows four successive iterations (denoted by k), or *backups*, before convergence is achieved (when $k = 3$). In this example, the MDP environment is made of 16 states arranged in a 4×4 grid. Available actions from any state are to transition either left, right, up or down to the next neighbouring state. However, a certain action is unavailable at a particular state if it would cause the agent to transition out of bounds. Goals states are coloured grey, they begin with value zero, and are the only states exempt from value updates. Each iteration updates the value function of every state once, which the agent can use to navigate towards a goal using Eq. 3.1.2. Each action incurs a reward of '-1' ($R(s',s,a) = 1, \forall\ s \in S, \forall\ a \in A$ in this example) to execute an action. Beginning at the first iteration ($k = 0$), all non-goal state values are set to negative infinity. After the first backup ($k = 1$), the value of each state updates by the maximum value of 'reward + value-of-neighbour-state' it finds. The value of states



(a) Policy evaluation (b) Policy improvement

Figure 3.1 – Example optimum-policy evaluation and improvement by value iteration; grey cells indicate goal states (value 0) and an agent in any other state can transition one cell left, right, up or down per iteration.

adjacent to the goal states update to $-1+0=-1$, whilst states further away remain at negative infinity as their neighbours were still currently valued as negative infinity. Since values were updated during this backup, the algorithm has not necessarily yet converged, and thus it will iterate again. As the $k=2$ iteration, states adjacent to '-1' valued states update to value '-2' and so on. Finally the value function converges after the $k=3$ backup, and the optimal and greedy policy in Fig. 3.1(b) at iteration $k=3$ is returned. By following this optimal policy, an agent anywhere on the environment can find its way to one of the goal states with an optimum number of action executions (multiple arrows indicate equally good choices).

An alternative technique of value iteration is policy iteration. Policy iteration initialises a policy with random actions to perform from each state, and iteratively:

1. computes policy evaluation (a value function V^{π} for the current policy π). This step is itself an iterative method similar to algorithm 3.1 except for line 6, where the action is chosen according to the current policy π rather than the action which minimises the RHS expression,
2. uses V^{π} to improve the policy π.

Several iterations many be required before an optimal policy is returned. Policy iteration is, however, generally not as efficient as value iteration due to the nesting of the policy evaluation iteration within policy iteration [47]. Thus policy iteration is not used in this thesis.

3.1.3 Sweeping

Algorithm 3.1 is correct and complete and guaranteed to converge in polynomial time [38]. Efficiency for DP value function convergence is heuristically increased by focusing on states whose values are frequently changing. For instance, if the value of one state s' is updated, the value of each state s that can transition to it (predecessor states) may also need to be updated due to Eq. 3.1.1. Conversely, if no resultant

states s' transitionable from state s updated their value in the previous backup, there is no need for s to update. Thus, value propagation can still be achieved without updating every state's value during each backup, rather only the predecessor states whose resultant state's values did change on the previous iteration. This more selective process is referred to as *sweeping*. This is implemented using a queue shown in Algorithm 3.2.

Algorithm 3.2: Dynamic programming: Value evaluation by sweeping

1: $queue.push(\forall\ state \in GoalStates)$
2: **while** $queue \neq \emptyset$ **do**
3: $\quad s_{updated} = queue.pop()$
4: $\quad$ **if** $queue.contains(s_{updated})$ **then**
5: $\quad\quad continue$
6: $\quad$ **end if**
7: $\quad$ **for** $\forall\ a \in Actions$ **do**
8: $\quad\quad$ **for** $\forall\ s \in s_{updated}.getPossiblePredecessors(a)$ **do**
9: $\quad\quad\quad v =\leftarrow \sum_{s'} P(s'|s,a)[R(s',s,a) + \gamma V(s')]$
10: $\quad\quad\quad$ **if** $v > V(s)$ **then**
11: $\quad\quad\quad\quad V(s) \leftarrow v$
12: $\quad\quad\quad\quad queue.push(s)$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: **end while**
17: Output: $\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s',s,a) + \gamma V(s')]$

Prioritised Sweeping uses a priority queue in Algorithm 3.2. As the change in value in some updates will be greater than others, and a large change in value is more 'relevant' in value propagation than small updates, associated predecessor states can be updated in an order which is prioritised according to the magnitude of this change.

3.2 Gaussian Processes

This thesis makes use of Gaussian process regression for learning-based mobility prediction. GP theory is introduced in this section, though is covered in much greater

detail in [45]. As mentioned previously in Sec. 2.3, GPs have the advantage over parametric models that they makes fewer assumptions about the complex environment-robot interaction. In addition, GPs are used in this thesis as they are adept at providing predictions with uncertainty given sparsely populated, spatially correlated data. Thus, control uncertainty (the DP stochastic transition function) can be modelled directly with PDFs the GP predicts, relying on limited training data which may only form a small subset of the many possible terrain formations the rover could encounter.

3.2.1 Model

A GP is a statistical modelling tool used in supervised learning to perform regression and classification. GPs are a standard framework to learn a model of correlated data and to provide inference with uncertainty. Uncertainty distributions are modelled as Gaussian probability distributions.

GP regression is concerned with exploiting the relationship between training input data X and training output data Y to model the correlations between the output values given different inputs. As such, a new test input vector $\boldsymbol{x}_*$ (or 'covariates') can be used to predict a test output y_*. Many other regression methods fit artificially-selected functions to data (e.g. a cubic polynomial) to model a relationship between inputs and outputs, with a focus on training free-parameters within such a function, optimising for Root Mean Square error of the test datum. However, it is often the case that there exist multiple functions which describe such a relation well, or perhaps there is not enough test data to justify why one candidate function should be used and others discarded. GPs, by contrast, can be thought of as a probabilistic distribution over the functions that describe the relation $Y = f(X)$.

A GP is specified by a mean and covariance function. The covariance function describes how one observation is associated (or *correlated*) with another observation. Typical covariance functions would have high correlation of outputs between two test datum that are similar, and small correlation between two test datum that are very

different. Covariance functions have free-parameters to define, called hyperparameters, represented by Θ.

The observed outputs differ from the underlying function by additive noise w of mean zero and variance σ_n^2: $y = f(x) + w$, where $w \sim \mathcal{N}(0, \sigma_n^2)$. σ_n^2 represents the noise in the observation of the output. The covariance function between two inputs is given by: $cov(x_i, x_j) = k(x_i, x_j) + \sigma_n^2$. Here, the covariance function cov is composed of an underlying a covariance function between inputs.

A common choice of covariance function used to describe the spatial correlation between two input vectors is the squared exponential:

$$k(\boldsymbol{x}, \boldsymbol{x'}) = \sigma_f^2 \exp\Big(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x'})^\top \Lambda^{-2}(\boldsymbol{x} - \boldsymbol{x'})\Big) + \sigma_n^2 I, \tag{3.2.1}$$

where σ_f^2 is the input variance (noise expected in observing the input data) and Λ is a length scale matrix of diagonal elements that describe the smoothness of the data or how 'fast' the covariance decays as the distance between inputs increases. These parameters comprise the hyperparameters for the squared exponential covariance function: $\Theta = \{\sigma_f, \Lambda, \sigma_n\}$.

3.2.2 Inference

The predictive distribution is given by a Gaussian,

$$p(f_*|X, Y, \boldsymbol{x}_*) \sim \mathcal{N}(\mu_*, \Sigma_*), \tag{3.2.2}$$

with predictive mean

$$\mu_* = K(\boldsymbol{x}_*, X)[K(X, X) + \sigma_n^2 I]^{-1} Y, \tag{3.2.3}$$

and variance

$$\Sigma_* = K(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(\boldsymbol{x}_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, \boldsymbol{x}_*), \tag{3.2.4}$$

where X is the $n \times m$ matrix of all n training input vectors, each of dimension m, Y is the $n \times 1$ vector of all training output (scalar) values, and $\boldsymbol{x}_*$ is the test input vector. $K(X, \boldsymbol{x}_*)$ is the covariance matrix which stores the covariance of each training input value against the test input values.

Training a GP is to define (or *learn*) the hyperparameters of the covariance function. This is often referred to as model selection. A standard method to compute hyperparameters involves assessing the likelihood of the observations, given the inputs *and* hyperparameters. This is called the marginal likelihood, though the log of the marginal likelihood is often used:

$$log\ p(Y|X,\Theta) = -\frac{1}{2}Y^\top K_Y^{-1}Y - \frac{1}{2}log|K_Y| - \frac{n}{2}log\ 2\pi, \tag{3.2.5}$$

where $K_Y = K_f + o_n^2 I$.

A natural choice of hyperparameters are those which best describe why the outputs Y were observed. This is done by computing the likelihood of observing the training outputs given the hyperparameters. We wish to solve for the hyperparameters Θ by maximising the $log\ p(Y|X,\Theta)$ term in Eq. 3.2.5. by gradient decent. Unfortunately, this optimisation problem is often not convex[1], however local optima solutions are often adequate. The partial derivatives of Eq. 3.2.5 with respect to the hyperparameters used for gradient decent are:

$$\frac{\partial}{\partial\Theta_i}log\ p(Y|X,\Theta) = \frac{1}{2}tr\Big((\boldsymbol{\beta}\boldsymbol{\beta}^\top - K^{-1})\frac{\partial K}{\partial\Theta_i}\Big), \tag{3.2.6}$$

where $\boldsymbol{\beta} = K^{-1}Y$. Various methods used for adapting the step size and stopping criteria during decent exist. This thesis uses the implementation by [45] which uses the slope ratio and Wolfe-Powell stopping criterion.

As found in Sec. 2.2, uncertainty that exists in different subsystems of a rover's framework deteriorate the performance of a motion planner in a real environment.

[1] If a function is not convex, then local optima computed by gradient decent of such a function are not guaranteed to be the global optima of the function.

Compared to an idealised motion planner in a simulated environment, some of this deterioration is inevitable, but much can be mitigated by explicitly addressing how to plan motion *given* these sources of uncertainty. As GPs are adept at providing predictions with uncertainty given sparsely populated, spatially correlated data, they are useful tools for learning and modelling different sources of uncertainty given adequate training data. As DP can utilise such predictions to quantify state-transition probabilities (control uncertainty) explicitly in planning, a natural framework using DP and GPs together can be constructed to allow a rover to learn from and plan motions according to the uncertainty of its controls.

Chapter 4

Motion Planning and Learned Control Uncertainty

With the aim of reaching a given goal region *safely* while optimising the total cost of traversal over the *executed* trajectory, our approach is to take into account the stochasticity of the control of the robot at the planning stage. This requires modelling the control uncertainty, which we achieve by experience, using machine learning. This section first presents our model of control uncertainty (Sec. 4.1), followed by the presentation of the learning technique used to train our model of control uncertainty (Sec. 4.2). The motion planning algorithm, which uses the trained uncertainty model to compute a motion policy rather than a single path, is then discussed (Sec. 4.3). Finally, an overview of the complete system is given (Sec. 4.4). Later, Chapter 5 details the implementation details which are specific to our test platform.

4.1 Control Uncertainty

Control uncertainty is modelled as a PDF of relative transition between states $p(\Delta s|s,a)$. $p(\Delta s|s,a)$ can be expressed using $P(s'|s,a)$ where $\Delta s \equiv s' - s$:

$$P(s'|s,a) = \int p(\Delta s|s,a) f(s+\Delta s, s')\,\mathrm{d}\Delta s, \tag{4.1.1}$$

$$\text{where } f(s_1, s_2) = \begin{cases} 1 & \text{if the discretisation of } s_1 \text{ corresponds to } s_2 \\ 0 & \text{otherwise} \end{cases}$$

$p(\Delta s|s, a)$ comprises our model of stochastic actions and is tied to observed environmental features that vary across the terrain and is learned through experience. In addition, $p(\Delta s|s, a)$ is assumed to be a Gaussian for the purposes of training, discussed below. This assumption was found to be reasonable empirically (see Fig. 6.1 and 6.2)

4.2 Learning-based Mobility Prediction

The stochastic transition function $P(s'|s, a)$ in Eq. 3.1.2 is not known *a priori* and requires empirical calibration. This section describes how relative state transitions (specifically $p(\Delta s|s, a)$) are learned from experience.

The estimation of relative change in state is achieved using Gaussian process regression. As described in Sec. 3.2, GPs are effective in cases where the input data is sparsely populated and spatially correlated. This is the case for our data. An advantage of using regression opposed to classification to predict motion is two-fold. Firstly multiple models are not required, one for each terrain classification. Second, information otherwise discarded during a classification process is instead included in the learning of control uncertainty. This is especially useful when a robot considers mobility on terrains that lies close to a classifier's decision boundaries.

In unstructured terrains, Δs may strongly depend on factors such as the terrain profile along the executed path and also the action executed. A complete terrain profile, as a subset of the elevation map or sensed proprioceptively continuously during action execution, contains too much information such that it would overfit the learning algorithm if input directly. For example, the concerned subset of the elevation map comprises all elevation cells traversed by each wheel. In our implementation (see Chapter 5), we have a: 2.5cm state discretisation, 30cm crab actions, and 6 wheels. This means there are approximately 72 elevation cells the rover traverses during an

action primitive. In this case, the input dimensionality (72) exceeds the training set size for any particular action (Table. 6.1), therefore it is unsuitable for learning. Thus, a smaller number of single-valued *features* or *characteristics* (functions) of the terrain profile are extracted for learning. A trivial choice of features may be those that down-sample this subset, or represent a coarser discretisation of the terrain profile. However, since this choice of functions is in no way related to the particular environment-robot interaction, there is no guarantee this dimensionality reduction would retain the relevant information to effectively predict motion. Terrain profiles are represented by a vector of features:

$$\boldsymbol{\lambda}(s,a) = \{\lambda_1(s,a), ..., \lambda_{Dim(\boldsymbol{\lambda})}(s,a)\} \tag{4.2.1}$$

where each feature $\lambda_i(s,a)$ is a function of a state-action pair, as state-action pairs identify the expected terrain profile with reference to the elevation map. The aim is that the information content of $\boldsymbol{\lambda}(s,a)$ should be neither too low nor too high for effective learning and that no pair of functions (λ_i, λ_j) be too strongly correlated (to avoid unnecessary redundancy in the learning). Good choices for candidate λ functions to test are those which can distinguish between situations where the robot would be more prone to slipping or have trouble climbing over a rock, with say hard flat terrain where the rover might have very little stochasticity in control. Therefore, $p(\Delta s|s,a)$ is learned as a function of $\boldsymbol{\lambda}(s,a)$ and a:

$$p(\Delta s|s,a) = p(\Delta s|\boldsymbol{\lambda}(s,a),a). \tag{4.2.2}$$

We use the GP formulation from [45]. The input vector $\boldsymbol{x}$ is a function of the terrain features, shifted such that the input has zero mean, i.e., $\boldsymbol{x} = \boldsymbol{\lambda} - \bar{\boldsymbol{\lambda}}_{train}$, where $\bar{\boldsymbol{\lambda}}_{train}$ is the mean of each terrain feature in the training data. We define $\Delta s_{i,a}$ as the i^{th} single-valued component of the change of state Δs resulting from executing action a. We define a GP for each $\Delta s_{i,a}, i \in [\![1, N]\!], a \in A$, where $N = Dim(s)$. The output value $y_{i,a}$ of one of these GPs is defined as $y_{i,a} = \Delta s_{i,a} - \bar{\Delta s}_{i,a}$, where $\bar{\Delta s}_{i,a}$ is the mean value of Δs_i across all executions of action a in the training data. $y_{i,a} = f_{i,a} + w_{i,a}$

has a zero mean and variance σ_n^2 equal to the variance of the additive noise $w_{i,a}$, i.e., $p(y_{i,a}|f_{i,a}) = \mathcal{N}(0, \sigma_n^2)$.

The covariance function used to describe the spatial correlation between two input vectors is the squared exponential (Eq. 3.2.1). Using Eq. 3.2.2, the predictive distribution is:

$$p(\Delta s_{i,a}|\boldsymbol{\lambda}(s,a)) - \bar{\Delta s}_{i,a} = p(f_*|X, Y, \boldsymbol{x}_*) \sim \mathcal{N}(\mu_*, \Sigma_*), \tag{4.2.3}$$

with predictive mean μ_* and variance Σ_* are given by Eq. 3.2.3 and Eq. 3.2.4.

Thus, $p(\Delta s_{i,a})$ is to be computed for untraversed terrain profiles using Eq. 4.2.3, where the test input vector $\boldsymbol{x}_*$ has not been observed directly but rather estimated. Estimation employs a kinematics model of the robot over the series of states that would be traversed by executing action a from state s.

Finally, our planner considers the uncertainty in each component Δs_i separately by using the full distribution learnt from Δs_i and the expectation of the other components. Representing $\boldsymbol{\lambda}(s,a)$ as $\boldsymbol{\lambda}$, Eq. (4.2.2) is calculated as:

$$\begin{aligned} &p(\Delta s|\boldsymbol{\lambda}, a) = p(\{\Delta s_1, ..., \Delta s_i, ..., \Delta s_N\}|\boldsymbol{\lambda}, a) \approx \\ &\{\mathbb{E}(\Delta s_1|\boldsymbol{\lambda}, a), ..., p(\Delta s_i|\boldsymbol{\lambda}, a), ..., \mathbb{E}(\Delta s_N|\boldsymbol{\lambda}, a)\}. \end{aligned} \tag{4.2.4}$$

4.3 Motion Planning

We compute a motion policy for the robot using dynamic programming. A policy can be viewed as a representation of least cost paths from every state to the goal. In contrast to a computing a single path from a given state, a policy is a means to compute which control action to take at every state in the state space. The choice of DP includes computation of a motion policy to direct the robot towards a goal region, which is optimal with respect to the discretisation of the state-space, discrete set of (stochastic) primitive motions the robot has available and the reward function [35]. Grid search algorithms such as A* can also achieve this, however, as DP incorporates knowledge of control uncertainty explicitly during the development of

the path, the expected total cost of traversal is additionally optimised with respect to our uncertainty model (Sec. 4.1). Finally DP computes a motion policy rather than a single path, which is relevant to this context given that the robot will inevitably deviate form the intended path.

As part of the choice of DP we have assumed accurate localisation and stochastic control. In practice, localisation can be considered as accurate enough as long as its error is at most comparable to the state discretisation. DP is a feasible method in low dimensional state spaces; in our problem the state s can be defined using two lateral dimensions x and y and one dimension for orientation ψ, i.e., $s = \{x, y, \psi\}$.

The motion policy is computed using DP with sweeping (see Sec. 3.1.3) where the state s defines the robot state (discretised into uniform cells), each action a in A is a short motion primitive, and the discount factor $\gamma = 1$ to ensure expected time to goal does not influence the safety of the platform. The reward function $R(s'|s, a)$ is computed from a cost map which represents the difficulty of traversal at each state in the terrain. Predictions of control uncertainty $P(s'|s, a)$ are computed by using regression on a trained GP as described in Sec. 4.2.

This choice of MDP, outlined in Sec. 3.1, is not a risk sensitive MDP, thus our motion planner is not risk adverse nor risk prone per se, rather it optimises for the average value of resultant state plus traversal cost, weighted by the probably of arrival, in selecting subsequent action commands. As such, the motion planner will still avoid dangerous regions when there is a reasonable probability it will transition there, though it is not adverse to high levels of control uncertainty when far from danger.

4.4 System

Fig. 4.1 shows an outline of the system. Once an elevation map of the robot's local environment has been computed, a kinematics model is used to estimate both the terrain profile characteristics and cost of traversal ($R(s', s, a)$) at each state-action pair. Characteristics of the terrain profile, as observed by the Inertial Measurement

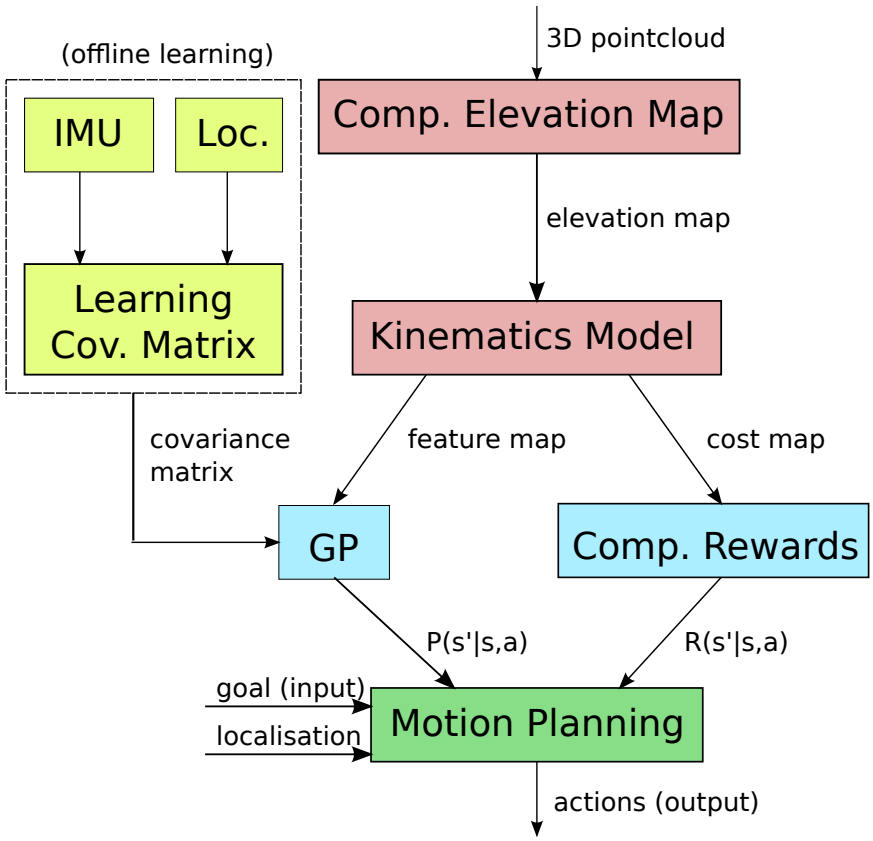


Figure 4.1 – System Outline. Colours indicate perception (red), offline learning (yellow), estimation (blue) and planning (green).

Unit (IMU) and the localisation system, are used as input data to train the GP offline. The GP can then be used to estimate the stochastic transition function $P(s'|s,a)$ on terrain similar to that encountered during training. Using both $P(s'|s,a)$ and $R(s',s,a)$, the motion planner computes the value function (Eq. (3.1.1)) over the observed state space to follow greedily as per the policy given in Eq. (3.1.2).

Chapter 5

Implementation

This section describes the implementation of the proposed approach on our experimental Mars rover shown in Fig. 5.1(a).

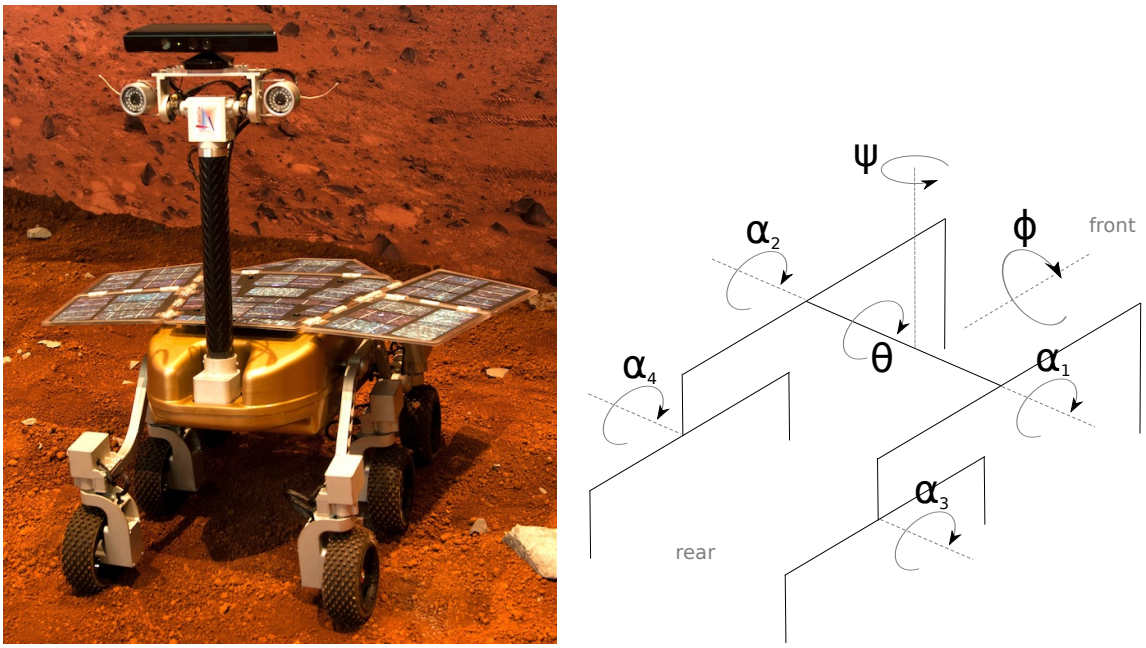
5.1 The Environment

All experiments described in (Chapter 6) were conducted in the Mars Yard environment shown earlier in Fig. 1.1, at the Powerhouse Museum in Sydney Australia. This environment is $117m^2$ in area and was designed to reproduce typical Mars terrain. It contains rocks of various sizes, small craters, and various grades of sand, dirt and gravel.

5.2 The Robot

"Mawson" is a six wheeled rover with individual steering servo motors on each wheel and a Rocker-bogie chassis. The platform is equipped with:

- an RGB-D camera (Microsoft Kinect) mounted on a mast, tilted down 14°, used for terrain modelling and localisation,



(a) Mawson (b) Orientation and Internal angles

Figure 5.1 – The Mawson Rover (a) and its chassis configuration (b).

- a 6-DOF IMU (Sparkfun Razor IMU, includes 3 accelerometers and 3 gyroscopes) used to measure the roll (ϕ) and pitch (θ) only of the robot,
- three potentiometers to observe the configuration of the chassis by measuring both bogie angles and the rocker differential (α_i in Fig. 5.1(b)).

The IMU does not require initialisation because only relative changes in pitch and roll during an action primitive are used for learning, described in Sec. 5.5.1. Absolute values of pitch and roll are not used directly. Similarly, the system is also robust against long-term drift in the values of pitch and roll.

For localisation and terrain modelling we use the RGB-D SLAM algorithm [13], implemented in the Robot Operating System (ROS) [14], which uses data from the RGB-D camera to perform simultaneous localisation and mapping (SLAM) online.

An elevation map is generated from the point clouds supplied by the RGB-D camera by distributing elevation points in a regular Cartesian grid. The grid resolution is $0.025m \times 0.025m$. This decision was based on a lower bound of the resolution,

according to the point cloud density the sensor provided, approximately $1 - 2cm$ between points at a range of $6m$ from the sensor. A soft upper bound is the contact area the wheel forms with the terrain, approximately $0.05m \times 0.05m$, to provide the kinematic model with the most accurate data possible to predict motions.

5.2.1 Localisation Accuracy

Test were conducted to determine accuracy of RGB-D SLAM localisation estimations in the Mars Yard environment. Various motion primitives (detailed in Sec. 5.4.2) were trialled including forward, backward and sideways translations, and rotations. Typical distance and yaw errors of RGB-D SLAM over 10m traversals are on average 9.7 cm and 0.069 radians [13]. Distance errors in the alternative InterSense localisation system used in one test (described in Sec. 6.4) was found empirically to be approximately 3cm. Neither localisation system used odometry information nor a system model, and thus any slip of the wheels did not contribute to localisation inaccuracy.

5.3 Kinematics Model

A kinematics model is used to estimate how the platform would interact with its environment. This section first describes an algorithm to estimate the configuration of the platform's Rocker-bogie chassis on an elevation map in Sec. 5.3.1. Estimation of terrain profile characteristics $\boldsymbol{\lambda}(s, a)$ using this algorithm is presented in Sec. 5.3.2. Finally, the difficulty of traversability of a state (cost function) in the elevation map is computed using the kinematics model, defined in Sec. 5.3.3.

5.3.1 Predicting attitude angles and chassis configuration

To predict the attitude angles $\{\phi, \theta\}$ and chassis configuration $\{\alpha_2 - \alpha_1, \alpha_3, \alpha_4\}$ (Fig. 5.1(b)) of the rover at given (x, y, yaw) positions on the elevation map, we use a method similar to [8] configured to Mawson's physical parameters. Although it

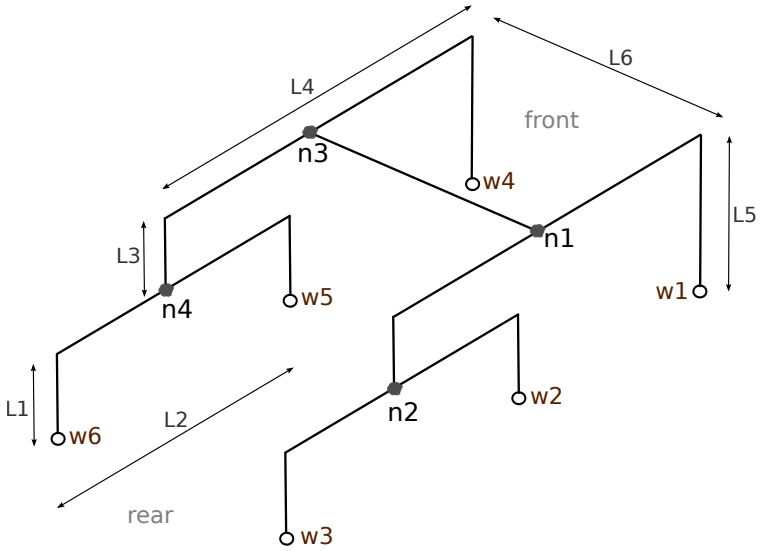


Figure 5.2 – Rover's Kinematic Structure: The rover is modelled with four nodes (pin joints) and six wheels.

does not take into account the dynamics of the platform, this simplified model is a sufficient approximation since the rover operates at low speeds.

The kinematic model of the rover is based on the pin-joint model in Fig. 5.2, which includes four pin-joint nodes that can freely rotate and six contact points (wheels) that rest on the surface of the elevation map. Wheel shape is not included in the model; the point of contact between a wheel and the elevation map is approximated as the base of the wheel. As such, local deformations of both the wheel and terrain caused by wheel/terrain interactions are not modelled. This approximation is justified in this approach as long as vertical displacements of wheels caused by terrain deformations are dominated by vertical displacements caused by a wheel climbing over rocks. In addition, variation in wheel height, compared with the measurements taken of wheel heights on flat terrain, are required to be small. Both of these have been the case in our training and testing. Another consequence of wheel size when traversing jagged edges is the ability to 'smooth out' vertical deviations of the wheel, which have thus been ignored. However, these errors experienced during testing were also small, plus the localised smoothing effects do not effect how the terrain characteristics are computed (Eq. 5.5.2 - 5.5.5). This is thus a simplistic model, but realistic enough for our purposes. An iterative process is used to increment or decrement robot attitude

angles $\{\phi, \theta\}$ and pin-joint rotations $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. During each iteration, wheel positions are calculated according to these values, and each wheel's height above the elevation map is used as an error term in the following iteration to adjust orientations and pin-joint rotations further. Convergence of $\{\phi, \theta, \alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ is achieved when each wheel's height above the elevation map is less than 1cm.

The algorithm is a single-loop function. It commences with each angle $\phi, \theta, \alpha_1, \alpha_2, \alpha_3, \alpha_4$ initialised to zero. Let h_i represent the elevation of the (simulated) wheel minus the elevation of the map at the same x-y location. The iterative process follows:

- The rover is 'lowered' onto the elevation map by the minimum h_i value, so that at least one wheel will be in contact with the elevation map, the other wheels will have non-negative h_i values:
  $[h_1, ..., h_6] \rightarrow [h_1, ..., h_6] - min(h_1, ..., h_6)$

- The pitch θ of the rover is then corrected (Fig. 5.3(a)) such that the average h_i value of the front wheels is equal to that of the rear wheels:
  $\theta \rightarrow \theta - asin\left(\frac{h_1+h_4}{(h_3+h_6)(L_4+L_2/2)}\right)$

- The roll ϕ of the rover is then corrected (Fig. 5.3(b)) such that the average h_i value of the right-side wheels is equal to that of the left-side wheels:
  $\phi \rightarrow \phi - asin\left(\frac{h_1+h_2+h_3}{(h_4+h_5+h_6)(L_6)}\right)$

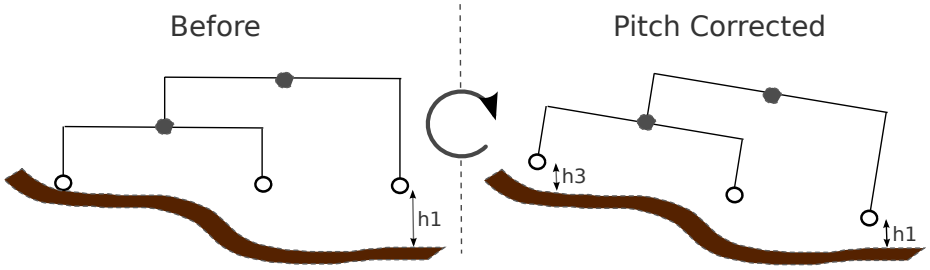
- The Rocker's right pin-joint is corrected such that the h_1 value of the front-right wheel is equal to the average of the right-bogie wheels:
  $\alpha_1 \rightarrow \alpha_1 + asin\left(\frac{2h_1}{(h_2+h_3)(L_4)}\right)$

- The Rocker's left pin-joint is similarly corrected:
  $\alpha_2 \rightarrow \alpha_2 + asin\left(\frac{2h_4}{(h_5+h_6)(L_4)}\right)$

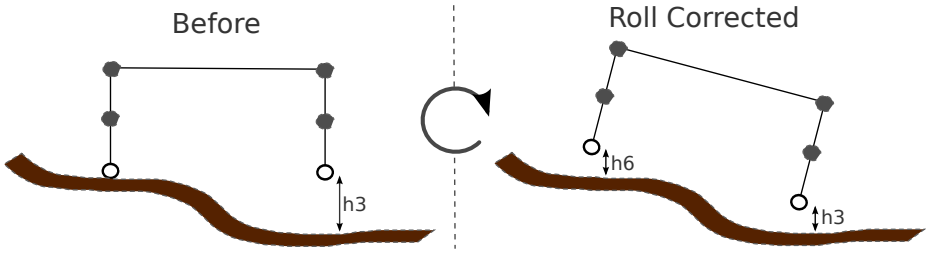
- The right-bogie pin-joint is corrected such that h_2 and h_3 of the right-bogie wheels are equal:
  $\alpha_3 \rightarrow \alpha_3 + asin\left(\frac{h_2}{h_3 L_2}\right)$

- The left-bogie pin-joint is similarly corrected:
  $\alpha_4 \rightarrow \alpha_4 + asin\left(\frac{h_5}{h_6 L_2}\right)$

This loop exits when the all h_i values are less than 1cm. An exception is if a $asin()$ function returns NaN which means not all six wheels would in contact with the elevation map at this (x, y, yaw) location. In this case the state is marked as an obstacle and the motion planner will not plan through it. Pseudocode of the kinematics model is presented in appendix A.2.



(a) Pitch correction: from the right-side of view of the rover, an iteration of the kinematics model algorithm corrects the rover's pitch.



(b) Roll correction: from the rear view of the rover, an iteration of the kinematics model algorithm corrects the rover's roll.

Figure 5.3 – Kinematics model algorithm

5.3.2 Feature Map

Given an elevation map and a method to predict attitude angles and chassis configuration angles, a feature map is computed. This maps states-action pairs to characteristics of the expected terrain profile encountered (Eq. 5.3.1). The terrain profile is represented by the evolution of rover configuration angles as estimated by the kinematics model. Thus, for any state-action pair, a corresponding value of $\boldsymbol{\lambda}(s, a)$ is

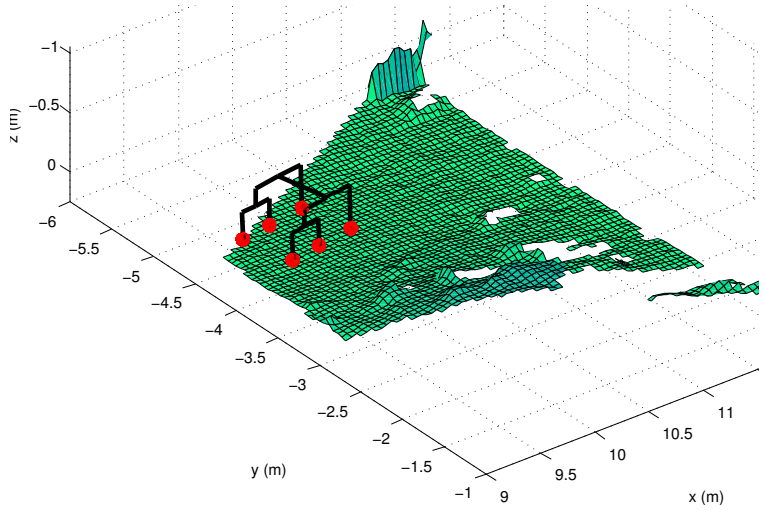


Figure 5.4 – Rover simulated on a terrain model using the kinematic model.

stored, used as test input data for the GP to predict mobility:

$$featureMap : \{S, A\} \rightarrow \boldsymbol{\lambda}(S, A) \subset \mathbb{R}^{Dim(\boldsymbol{\lambda})}. \tag{5.3.1}$$

$\boldsymbol{\lambda}(s, a)$ is estimated for each (s, a) pair by first assessing the expected terrain profile the rover would traverse. This is computed as a linear interpolation between the start state 's' and the the expected resultant state $E(s'|s, a)$. The kinematics model is used to estimate rover attitude angles and pin-joint rotations at states along this line, giving a sequence of expected configurations along the motion primitive. This sequence, representing the terrain profile, is used as input into $\boldsymbol{\lambda}$, discussed further in Sec. 5.5.2.

5.3.3 Cost Map

The cost function chosen to generate the cost map from the elevation map penalises large absolute values of roll, pitch, and configuration angles of the chassis at a given position $s = \{x, y, \psi\}$:

$$cost_{terrain}(s) = (cost_{\phi\theta}(s) + cost_{\alpha}(s))^2, \tag{5.3.2}$$

where:

$$cost_{\phi\theta}(s) = (\phi^2 + \theta^2), \tag{5.3.3}$$

$$cost_{\alpha}(s) = (\alpha_2 - \alpha_1)^2 + \alpha_3^2 + \alpha_4^2. \tag{5.3.4}$$

The $(\alpha_2 - \alpha_1)$ term is used as only the differential of the rocker can be measured. Since the configuration of the robot at a given position on the elevation map depends on its orientation, a 2D (x, y) cost map needs to be generated for each discretised orientation. The result is a 3-dimensional (x, y, ψ) cost map. The cost function was derived empirically by noting the difficulty of the platform in climbing different sizes of rocks.

5.4 Planning

This section describes how the motion planning algorithm of Sec. 4.3 is implemented. This includes how the state space, action space and reward function have been defined, specific to our test platform. Full pseudocode is available in appendix A.1.

5.4.1 State Space

As mentioned in Sec. 4.3, the rover's state s is defined as its position and orientation:

$$s \triangleq \{x, y, \psi\} \in \mathbb{R}^3, \tag{5.4.1}$$

where x and y are orthogonal lateral dimensions in meters, and ψ represents yaw in radians. This definition specifies all other orientations and internal angles ϕ, θ, α at each state using the kinematics model. State resolution was required to be smaller than the uncertainty bounds of resultant positions of actions in order for uncertainty to be considered by the DP[1]. A discretisation of $0.025m \times 0.025m \times \frac{\pi}{32} rad$ is sufficient in which actions could result in one of multiple distinct states. The discretisation was determined empirically using the standard deviation of errors of heading and yaw from actions listed in Table 6.1. This equates to approximately two to five possible resultant states considered for each state-action pair.

The point position $\{x, y\}$ refers to approximately the rover's centre of rotation. The centre of rotation lies 13mm from the rover's line of symmetry, to the right when rotating clockwise, to the left when rotation anti-clockwise, due to the front wheel's motors being more powerful and a forward centre of gravity. Thus, the rover would 'drag' the platform slightly when rotating. The robot's point position is defined as the midpoint between these two centres of rotation.

[1] If the state discretisation dominates the range of resultant positions such that within two standard deviations all possible resultant positions computed lie within the same state discretisation, then the planning is effectively deterministic. In this case information about the uncertainty of a motion primitive is lost.

5.4.2 Action Set

We define two action types for the rover: *crabbing* and *rotation*. Crabbing corresponds to executing a straight line translation in the xy-plane by a given distance and heading, with no change in ψ, and constant linear velocity ($0.11m/s$). The rover is able to crab in any direction. Rotation is a spin-on-the-spot motion primitive at constant angular velocity ($0.24rad/s$). It only changes ψ by a given magnitude. In total, the action set A is composed of 2 rotation and 8 crabbing motion primitives:

$$\begin{aligned} A \triangleq \; & \big\{\text{rotate}\,(\pi/4)\,,\ \text{rotate}\,(-\pi/4)\,, \\ & \text{crab}\,(0.3m, n\pi/4)\ \forall n \in [\![-3, 4]\!]\big\}. \end{aligned} \tag{5.4.2}$$

Restricting the actions to this set was the result of a trade-off between rover dexterity and algorithm complexity. A discrete set is used, rather than continuous, as the motion planning task in the MDP environment is not convex and cannot be solved by gradient descent. The set was chosen empirically with sufficient actions to achieve a suitable level of path diversity. However, the number of actions was limited to 10 to bound the training set size required (independent training is required for each action) and also limit the number of actions assessed at each state during planning. A single speed was chosen to limit what needed to be learned.

5.4.3 Reward Function

The reward $R(s', s, a)$ of an action is defined as the negative of the average cost of states that lie on a linear interpolation between a start state s and a resultant state s':

$$\begin{aligned} R(s', s, a) = \quad & -\xi - \frac{1}{K}\sum_{i=0}^{K} Cost\Big(s_x + \frac{i}{K}(s'_x - s_x), \\ & s_y + \frac{i}{K}(s'_y - s_y), s_\psi + \frac{i}{K}(s'_\psi - s_\psi)\Big), \end{aligned} \tag{5.4.3}$$

where $\xi = 0.003$ is a small penalty used to deter excessive motions on flat terrain, and K is the sampling resolution.

5.5 Learning and GP

To achieve mobility prediction from training data, we used the GP implementation from [45]. The approach was first tested on flat terrain which consisted of flat sand and large rock obstacles to avoid. This was intended as a preliminary trial of the approach before extending training and testing to 'rough terrain'. Rough terrain tests used rocks of different sizes, a subset of which were small enough to not be classed as obstacles by the kinematics model, and could be traversed.

5.5.1 Training

Training data were obtained through experimental runs of the rover executing each action a from A multiple times, while logging discretised sets of $\{\alpha, \phi, \theta, time, s, a\}$ continuously. The training data collected comprise the rover's attitude angles, provided by the on-board IMU, and the deviation from the expected motion when executing a given action, measured using the localisation system. Due to the left-right symmetry of the platform, training was only required on 6 of the 10 actions from A (Fig. 6.1 shows which set of training data can be combined between symmetric actions).

Due to slow localisation updates, Δs could only be measured at the end of each action primitive. However, multiple values of $\{\phi, \theta, \alpha\}$ were recorded during the execution of an action primitive, resulting in a vector of these values $\{\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\alpha}\}$, representing the terrain profile.

Flat terrain: In flat-terrain traversal, variations of $\{\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\alpha}\}$ was negligible, therefore, motion errors were learnt with respect to action only. As both the input and

output were uni-dimensional, basic mean and variances of motion errors are computed for each action, rather than using a GP.

Rough terrain: Terrain profiles were encoded more compactly by extracting features of the evolution of $\{\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\alpha}\}$ to limit training data dimensionality, and thus avoid overfitting. A combination of terrain profile features were tested with GP regression using cross validation. The features (vector of functions $\boldsymbol{\lambda}$) which produced the lowest Root Mean Square error between the GP mean estimations and test datum output was chosen:

$$\boldsymbol{\lambda} \triangleq \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}, \tag{5.5.1}$$

where

$$\lambda_1(\boldsymbol{\phi}, \boldsymbol{\theta}) = \ max(\phi_j - \phi_i), \ \forall i, j : i < j, \tag{5.5.2}$$

$$\lambda_2(\boldsymbol{\phi}, \boldsymbol{\theta}) = \ min(\phi_j - \phi_i), \ \forall i, j : i < j, \tag{5.5.3}$$

$$\lambda_3(\boldsymbol{\phi}, \boldsymbol{\theta}) = \ max(\theta_j - \theta_i), \ \forall i, j : i < j, \tag{5.5.4}$$

$$\lambda_4(\boldsymbol{\phi}, \boldsymbol{\theta}) = \ min(\theta_j - \theta_i), \ \forall i, j : i < j, \tag{5.5.5}$$

i.e., largest increase of ϕ, largest decrease of ϕ, largest increase of θ and largest decrease of θ during an action primitive. $\boldsymbol{\alpha}$ did not improve predictive results of the GP, and thus was not included.

5.5.2 Prediction

When planning over previously untraversed states, $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are estimated using the kinematics model from a state-action pair: $\hat{\boldsymbol{\phi}}(s, a)$ and $\hat{\boldsymbol{\theta}}(s, a)$. The $\boldsymbol{\lambda}$ functions are then applied to these estimates, which are input to the GP. Thus, the expression

$$p(\Delta s_i | \boldsymbol{\lambda}(s, a), a) = p(\Delta s_i | \boldsymbol{\lambda}(\hat{\boldsymbol{\phi}}(s, a), \hat{\boldsymbol{\theta}}(s, a), a))$$

is substituted in Eq. (4.2.4).

$\hat{\boldsymbol{\phi}}(s,a)$ and $\hat{\boldsymbol{\theta}}(s,a)$ are computed using the expected sequence of states $\boldsymbol{s}$ traversed by (s,a) as the linear interpolation of states between the start state 's' and the expected resultant state $E(s'|s,a)$. For each interval state s_i in $\boldsymbol{s}$, the kinematics model computes an associated ϕ_i and θ_i value in $\hat{\boldsymbol{\phi}}$ and $\hat{\boldsymbol{\theta}}$.

5.5.3 Outputs

The components Δs_i of Δs (see Sec. 4.2) were defined according to radial coordinates, as per the control space of crabbing and rotations, opposed to the Cartesian representation of the state space $\{x, y, \phi\}$ itself. The Δs_i components we consider are heading and distance travelled for crabbing actions

$$\Delta s_1 \quad = \Delta s_{head} \quad = atan2(\Delta y, \Delta x) \tag{5.5.6}$$

$$\Delta s_2 \quad = \Delta s_{dist} \quad = \sqrt{(\Delta x)^2 + (\Delta y)^2}, \tag{5.5.7}$$

and yaw for rotation actions

$$\Delta s_3 = \Delta s_{yaw} = \Delta \psi. \tag{5.5.8}$$

Therefore, Δs is represented by the tuple

$$\Delta s \triangleq \{\Delta s_{head}, \Delta s_{dist}, \Delta s_{yaw}\}. \tag{5.5.9}$$

5.6 Discussion

Uncertainty of the resultant state has several causes including imperfect actuation, undetectable properties of observed terrain, and an inability to perfectly model terrain-robot interaction. Slippage on sandy terrain, terrain deformability, differing power requirements to climb different rocks are factors which are not modelled explicitly, but correlate with the terrain profile features learned, and thus implicitly captured in the training data. Training and testing was restricted to non-sloping terrains to

limit the complexity of the scenarios. Whilst this does not encompass all scenarios expected on a Mars terrain, sloping terrains are left to future work.

In the rest of the thesis, we define control errors (δs) similarly to slip [3] as the difference between observed change in state (Δs) and "ideal" change in state ($\tilde{\Delta} s$) (i.e., if the controller followed the action-command perfectly); i.e., $\delta s = \Delta s - \tilde{\Delta} s$, a tuple analogous to Eq. (5.5.9).

Chapter 6

Experimental Results

We evaluated our approach both in simulation and through experiments with a planetary rover robot. Our experimental methodology consisted of two phases. First, we learned statistics of control error through empirical trials, described in Sec. 6.1. Then, we performed navigation experiments using these learned data to build the motion planner's stochastic transition function. Sec. 6.2 describes experiments in simulation, and Sec. 6.3 and 6.4 describe experiments using the robot.

6.1 Training on Flat and Rough Terrain

Training was conducted for two cases: flat-terrain traversal and rough-terrain traversal. For the flat terrain case, control errors were learned by executing multiple runs for each action. Rough-terrain training additionally involved traversal of various rocks (one at a time).

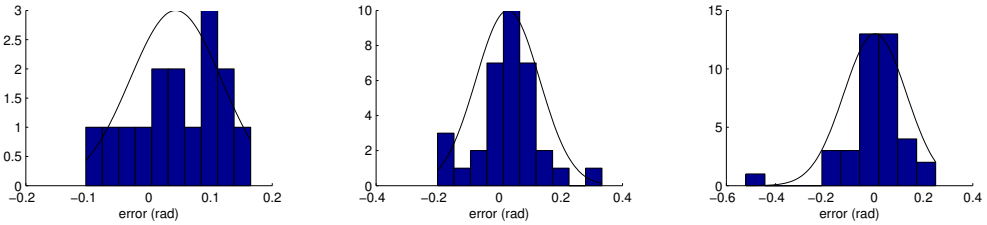
Statistics on these error terms of the training data obtained (before GP estimation) for δs_{head} and δs_{yaw} are shown in Table 6.1. In particular, this table indicates the mean and standard deviation (std.) of the error for each action.

Table 6.1 – Mobility Prediction by action, GP features not included

Action:	crab 0π	crab $\pm\pi/4$	crab $\pm\pi/2$	crab $\pm3\pi/4$	crab π	rotate $\pm\pi/4$
Error:	δs_{head}	δs_{head}	δs_{head}	δs_{head}	δs_{head}	δs_{yaw}
Flat Terrain						
mean (rad)	0.043	0.028	0.004	0.006	0.058	-0.117
std. (rad)	0.074	0.103	0.127	0.091	0.088	0.140
# samples	15	34	39	34	12	32
Rough Terrain - *marginalised by Action*						
mean (rad)	0.044	0.010	0.060	0.037	0.037	-0.063
std. (rad)	0.081	0.115	0.158	0.117	0.089	0.119
# samples	43	58	58	48	35	54

6.1.1 Flat Terrain Training

Training data included 166 motion primitive executions (approximately 28 motions per symmetric action), recording IMU values and localisation data. The heading errors (δs_{head}) and yaw errors (δs_{yaw}) learned for each of the six symmetric actions are shown in Fig. 6.1. Mean and variance were computed directly for flat terrain learning rather than using GPs because the input data (the action executed) was one dimensional. Although the terrain was mostly flat, the variance in motion primitive error is significant, which validates the need to take uncertainty into account in the planning. We found that the distributions could reasonably be approximated by a Gaussian.



(a) Crab(0.3m, 0π) (b) Crab(0.3m, $\pm\frac{\pi}{4}$) (c) Crab(0.3m, $\pm\frac{\pi}{2}$)

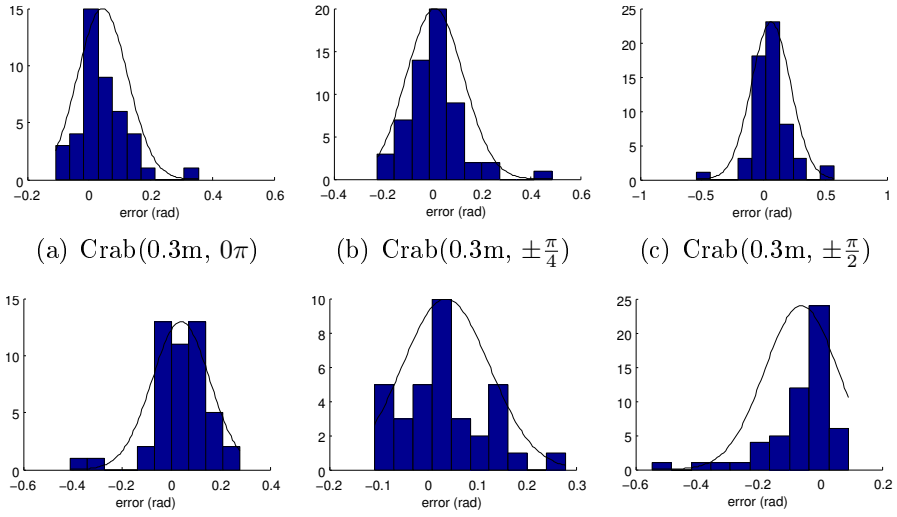
(d) Crab(0.3m, $\pm\frac{3\pi}{4}$) (e) Crab(0.3m, π) (f) Rotate($\pm\frac{\pi}{4}$)

Figure 6.1 – Mobility prediction by action on flat terrain. Histograms present heading errors recording during crabbing, and yaw errors recorded during rotations.

6.1.2 Rough Terrain Training

During rough-terrain traversal, various rocks were traversed by each wheel of the rover. Note that in some cases some rocks shifted under the weight of the rover, slightly sinking into the sand or rolling over. These types of situations, which are ex-

tremely difficult to predict by modelling, were therefore captured in our learning data. As expected, Table 6.1 and Fig. 6.2 shows the control errors were more significant in rough terrain data.



(a) Crab(0.3m, 0π) (b) Crab(0.3m, $\pm\frac{\pi}{4}$) (c) Crab(0.3m, $\pm\frac{\pi}{2}$)

(d) Crab(0.3m, $\pm\frac{3\pi}{4}$) (e) Crab(0.3m, π) (f) Rotate($\pm\frac{\pi}{4}$)

Figure 6.2 – Mobility prediction by action on rough terrain. Histograms present heading errors recording during crabbing, and yaw errors recorded during rotations.

Table 6.2 shows the GP hyperparameters obtained. A visual example of the terrain characteristics used in GP training (Eq. 5.5.2-5.5.5) traversing up and over a small rock is shown in Fig. 6.3.

Training data included 296 motion primitive executions (approximately 50 motions per symmetric action). The time to computing terrain features of all 296 motion primitives and train all 18 GPs (Eq. 4.2.3) was 40 seconds. This was done offline on an Intel Core 2 Duo 3.0GHz CPU with 4GB memory.

6.2 Simulation of Flat Terrain Traversal

We simulated the robot traversing flat terrain. A simulation was conducted to test the planner more times than was feasible on the real terrain. Control uncertainty was simulated using the learned data described in Sec. 6.1. The robot's environment

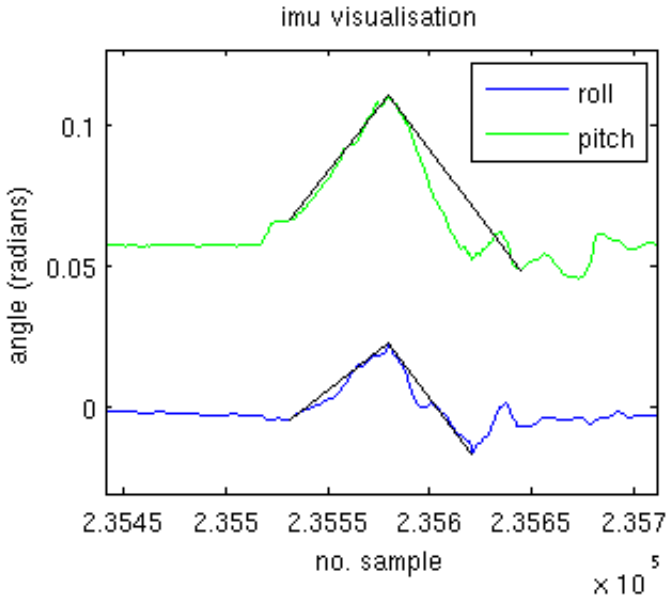
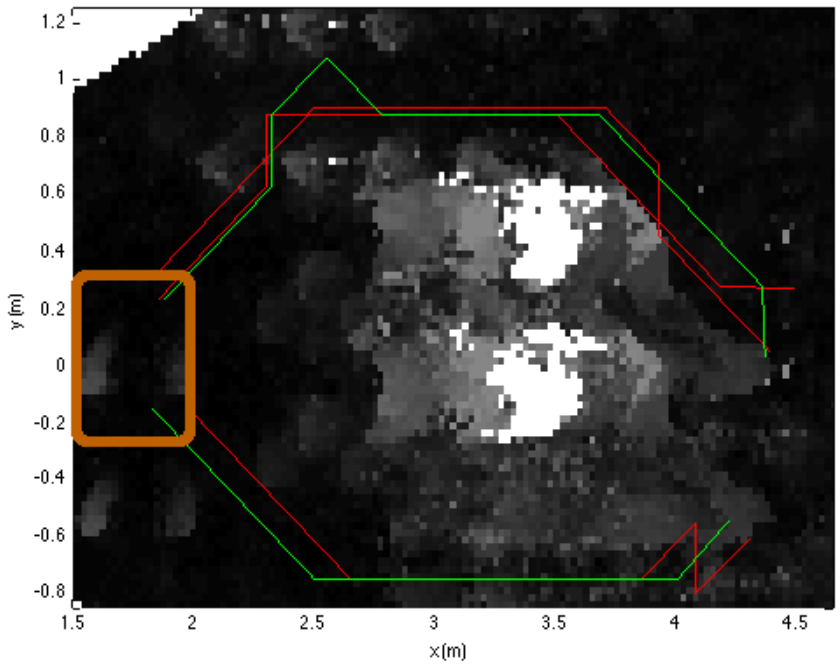


Figure 6.3 – Example of terrain profile features. A set of features are computed for an action primitive during training. Green: evolution of rover pitch during an action primitive. Blue: evolution of rover roll. Black: lines indicating the maximum increase and decrease of both roll and pitch during execution of the action primitive, according to Eq. 5.5.2-5.5.5.



(a) Simulated trajectories

(b) The real terrain

Figure 6.4 – (a) shows a few samples of simulated trajectories navigated around a cluster of rocks (shown in (b)). The cost on the map is shown as levels of grey, with white indicating the highest cost, for a single orientation value: the rover facing left. The brown rectangle on the left indicates the common goal region. Red trajectories were computed without considering uncertainty, while green trajectories considered uncertainty.

Table 6.2 – GP hyperparameters trained from traversals in rough terrain

Action	Error	Λ_{11}^{-2}	Λ_{22}^{-2}	Λ_{33}^{-2}	Λ_{44}^{-2}	σ_f	σ_n
crab 0π	δs_{head}	0.010	0.103	0.223	0.015	0.070	0.032
crab $\pm\frac{\pi}{4}$	δs_{head}	0.246	0.198	0.230	0.019	0.000	0.073
crab $\pm\frac{\pi}{2}$	δs_{head}	0.004	0.016	0.071	0.022	0.049	0.044
crab $\pm\frac{3\pi}{4}$	δs_{head}	0.157	0.064	0.056	0.665	0.127	0.054
crab π	δs_{head}	0.029	0.057	0.198	0.066	0.000	0.043
crab 0π	δs_{dist}	0.098	0.143	0.034	0.232	0.000	0.028
crab $\pm\frac{\pi}{4}$	δs_{dist}	0.002	0.066	0.395	0.949	0.000	0.039
crab $\pm\frac{\pi}{2}$	δs_{dist}	1.555	1.664	0.855	0.021	0.032	0.035
crab $\pm\frac{3\pi}{4}$	δs_{dist}	0.215	0.001	0.051	0.188	0.000	0.034
crab π	δs_{dist}	1.981	0.590	0.099	0.025	0.087	0.039
rotate $\pm\frac{\pi}{4}$	δs_{yaw}	0.076	0.443	0.002	0.143	0.001	0.024

was simulated using a point cloud acquired by the robot's RGB-D camera. This environment is a roughly flat area with a cluster of rocks, shown in Fig. 6.4(b). Trials consisted of placing the robot randomly around the cluster of rocks and directed to a unique goal region opposite the rock cluster. We used a cost function where any rock on the terrain appears as an obstacle.

Planning without uncertainty (where the expectation of the change in state $\mathbb{E}(P(\Delta s|\boldsymbol{\lambda}(s,a),a))$ learned is used instead of the full distribution) and planning considering uncertainty were each tested 100 times. When planning considering uncertainty, the flat-terrain learning data was used, where Δs_{head} was considered during crab actions, and Δs_{yaw} was considered during rotate actions. Resultant trajectories were assessed in light of the known Δs_{head} and Δs_{yaw} distributions to determine the probability of colliding with a rock as well as the expectation of accumulated cost for each trajectory. Results obtained for both methods can be compared using the statistics on all *executed* trajectories in Table 6.3. These statistics represent: the averages of total cost accumulated from start position to goal $cost_{total}$, the probability of hitting an obstacle summed over the entire trajectory $P_{collision}$ and the minimum distance 'Min. Dist.' to an obstacle over the trajectory.

$cost_{total}$ is the sum of rewards (Eq. 5.4.3) between the sequence of states each recorded

at the completion of an action, from state state s_0 to final state s_G, where G represents the number of actions executed to reach the goal:

$$cost_{total} = \sum_{i=0}^{G-1} R(s_{i+1}, s_i, a_i), \tag{6.2.1}$$

where a_i is the action executed from state s_i. $P_{collision}$ is computed using the learned uncertainty in heading from Table 6.1:

$$P_{collision} = 1 - \prod_{i=0}^{G-1} \sum_{s'} P(s', s_i, a_i).isreal(R(s', s_i, a_i)), \tag{6.2.2}$$

where the $isreal()$ function returns false if any state sampled by the reward function is invalid (an obstacle), true otherwise.

Table 6.3 – Simulated Planning around a Cluster of Rocks

No Uncertainty	$cost_{total}$	$P_{collision}$	Min. Dist. (m)
mean	1.132	0.028	0.154
std.	0.605	0.101	0.131
max.	6.879	0.68	0.530
min.	0.727	0	0.015
Head. Uncertainty	$cost_{total}$	$P_{collision}$	Min. Dist. (m)
mean	1.080	0.005	0.161
std.	0.138	0.035	0.113
max.	1.449	0.34	0.505
min.	0.733	0.0	0.025

Fig. 6.4(a) shows a few examples of trajectories executed. Holes present in the cost map are considered obstacles: they are either states of "impossible configuration" where the kinematics model estimates the rover would not have all six wheels on the terrain at that position, or occlusions due to shadow areas behind rocks where the camera could not observe. By convention these obstacles are not considered as valid states by the motion planner and it does not consider actions that may result in an invalid state.

Results in Table 6.3 highlight two of the major consequences of planning without uncertainty: a platform will be more likely to collide with an obstacle and will, on average, accumulate more cost in traversing to the goal region. In fact, when planning without uncertainty, the projected accumulated cost will always be underestimated when the robot deviates at least once from the lowest cost path it computes, which is frequently the case with imperfect control. Thus planning without uncertainty cannot provide any guarantee of total cost accumulated to reach a goal region, which is important when a decision needs to be made regarding if a goal region is worth visiting up to a certain cost of traversing there. The safety issue of rock collisions occurs when the planner follows paths very close to an obstacle without considering consequences of slight deviations from its path.

6.3 Experiments of Flat Terrain Traversal

We conducted a series of experiments using the physical robot traversing flat terrain. The robot navigated from a starting position to a goal region whilst avoiding large rocks (Fig. 6.5 shows the rover in the goal position). The terrain was flat sand and gravel, however due to the imperfect control on the loose terrain, the control uncertainties were significant, as shown earlier in the training data in Sec. 6.1. We used the same cost function as for the simulation (Eq. 5.3.2).

Trajectories were planned and executed on the robot 10 times for planning with uncertainty (including heading during crabbing actions and yaw during rotation actions). For comparison, 10 trajectories were also planned and executed without accounting for uncertainty. In both cases, two different starting points were used, while the goal region was the same.

Fig. 6.6 shows a few examples of the actual trajectories executed by the rover. Occlusions shown in the cost map in are due to "shadows" of rocks when they were observed by the robot.

The results of planning with uncertainty of heading during crabbing actions and yaw

Figure 6.5 – Flat traversal experiment, planning around a collection of rocks too large to climb over safely.

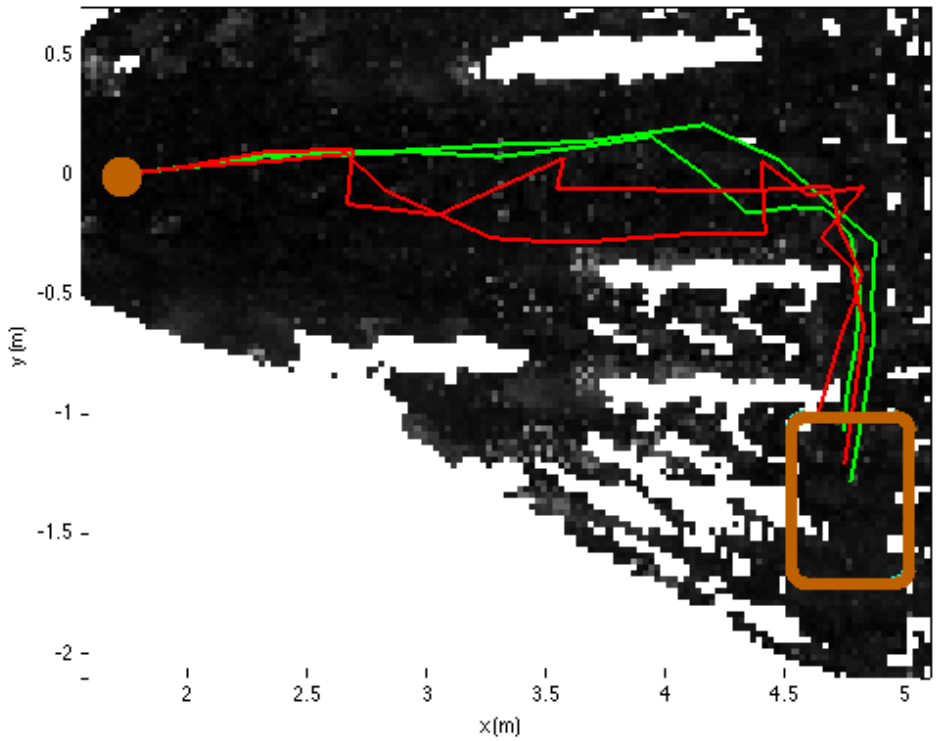


Figure 6.6 – Example of trajectories taken to avoid several rocks on otherwise flat terrain. Red: without uncertainty considered. Green: with uncertainty in heading. The starting position is indicated by the brown circle on the left and the goal region is shown as the brown rectangle at the bottom right. Planning with uncertainty generates paths that are less sensitive to control error.

Table 6.4 – Flat traversal: probability assessment

No Uncertainty			$cost_{total}$	$P_{collision}$	Min. Dist. (m)
# Trials:	10	Mean:	1.295	0.380	0.095
# Collisions:	2	Std.:	0.385	0.492	0.084
Uncertainty			$cost_{total}$	$P_{collision}$	Min. Dist. (m)
# Trials:	10	Mean:	1.177	0.016	0.165
# Collisions:	1	Std.:	0.262	0.005	0.088

during rotation actions are compared statistically against results of planning without uncertainty in Table 6.4. This table shows statistics of the accumulated cost along the executed paths. Results indicate that the robot would, on average, plan wider berths around rocks with uncertainty considered and execute safer paths in practice. A point of interest is the number of collisions with a rock (considered in this test as untraversable): 1 collision still occurred when using planning with uncertainty, against 2 collisions for the version without uncertainty, However, a subsequent probability assessment of each trajectory (Eq. 6.2.2) showed that in this case, Mawson was quite "unlucky" to collide with the rock as it had computed only a 16% chance of collision. This assessment also revealed the rover was quite "lucky" it did not collide with more than 2 rocks when uncertainty was not considered, with an average collision probability of 38% per trajectory.

6.4 Experiments on Unstructured Terrain: Traversing Rocks

We also conducted a series of experiments where the robot traverses rough terrain. Trained GP models were used to predict control uncertainty as described in Sec. 6.1. As in the training phase, the experiments were conducted in unstructured and rough terrain because of the presence of rocks that the robot sometimes has to travel across, but they were limited to areas with negligible terrain slopes.

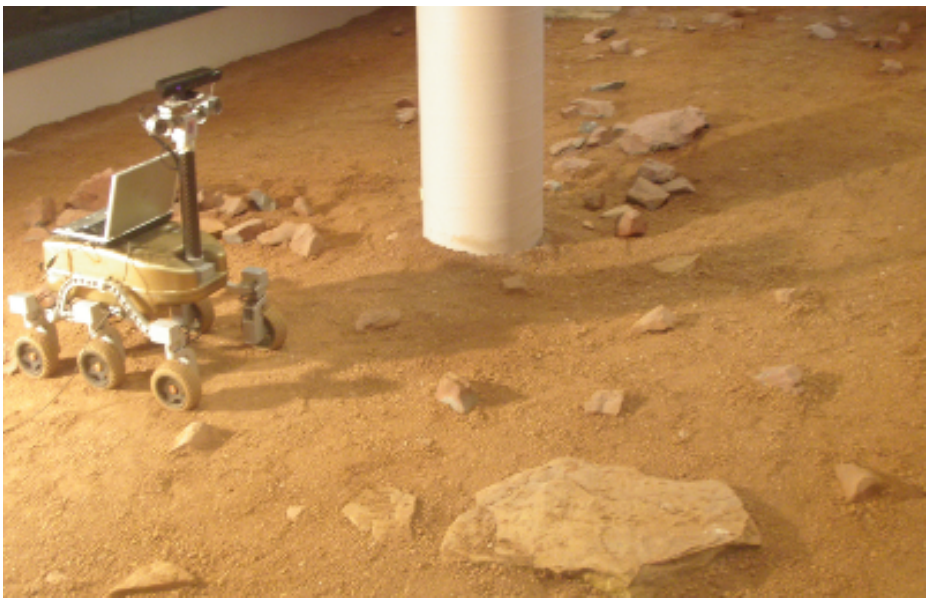
The learned rough-terrain models were limited to traversal of one rock at a time, and thus two rock fields (Fig. 6.7) were set up accordingly (labelled 'rock field A' and 'rock field B'). However, the layout of rocks was dense enough to cause the robot to traverse multiple rocks while navigating to the goal region.

In addition to Δs_{yaw} during rotation actions, two types of uncertainty were considered (separately) with crab actions: Δs_{head} and Δs_{dist}. Fig. 6.8 and Fig. 6.9 show example policies computed by our planning algorithm for each rock field.

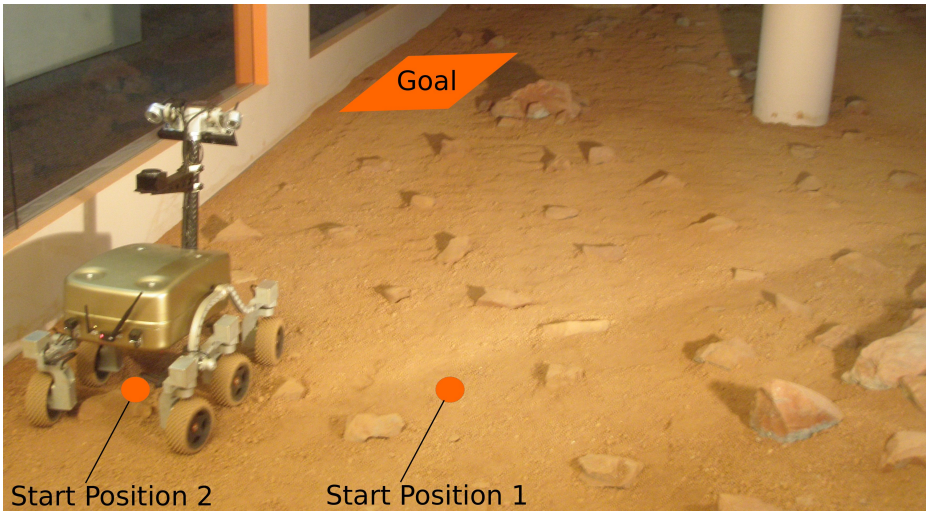
Testing conducted on rock field B used an upgraded localisation system installed in the Mars Yard. This was the InterSense IS-1200 inertial-optical tracking system [51]. This system uses a camera on the rover to compute rover position and orientation with respect to fiduciary markers placed in the Mars Yard. Accuracy of the system was superior to the previous system mentioned Sec. 5.2.1, with the additional benefit that, as a global positioning system, error in localisation did not increase with increased distance travelled by the rover. This new localisation system necessitated rough terrain learning to be conducted again, as localisation inaccuracies are indirectly captured in the learning data. We took this opportunity to additionally increase torque applied to the rover's wheels for greater ability to climb larger rocks in rock field B. Results of re-learning the control error in rough terrain are shown in Table 6.5, marginalised by action. This learning was used for rock field B only. This table additionally compares control error found during the learning phase with those recorded during the planning/test phase [1] traversing rock field B. The observed distributions reasonably match the predicted distributions.

Resultant trajectories in Fig. 6.10 show that whilst each planning method attempted to navigate between rocks, the method of planning without uncertainty (red) would tend to "zig zag" more severely in attempts to attain the least possible cost path, finely navigating every rock. Methods considering uncertainty (green, cyan), by contrast, tended to "hold the course" more. This is because the "zig zag" action sequences can actually result in more rock traversals in total, due to a greater distance travelled in

[1] Due to a bug in logging, values for two particular actions during planning were not recorded, marked by a 'n/a'.



(a) Rock field A. The goal region to the right (out of the picture).



(b) Rock field B.

Figure 6.7 – Rock traversal experiment setup: Mawson is shown at its starting position. It must traverse over a "rock field" to reach a goal region.

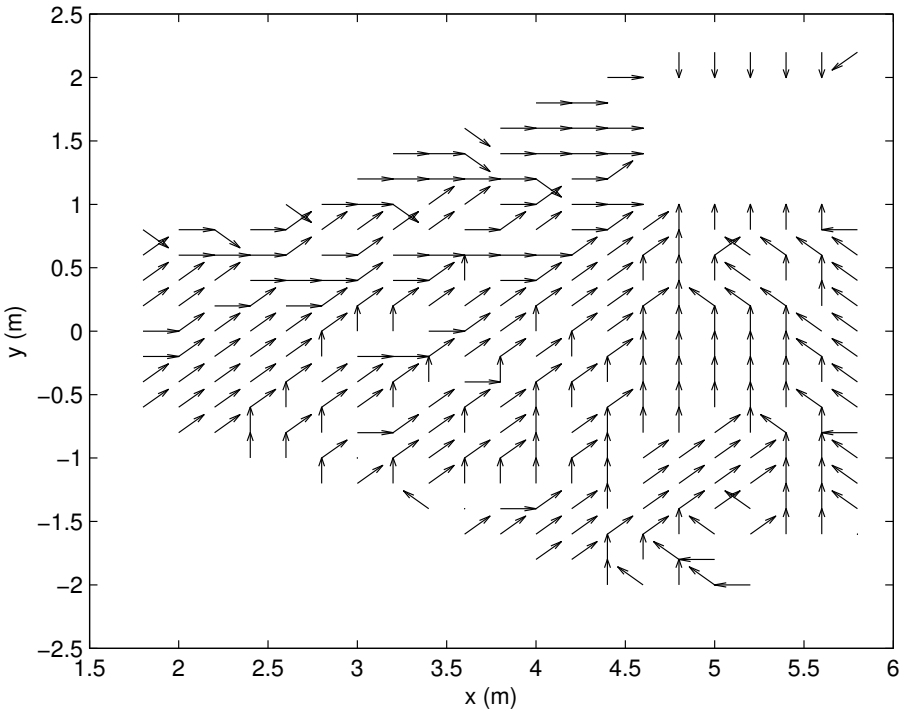
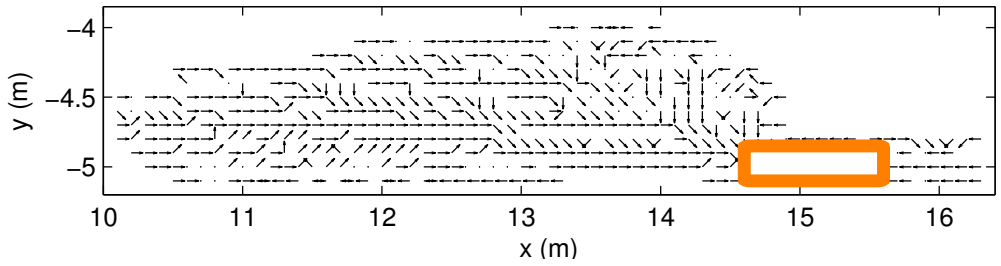
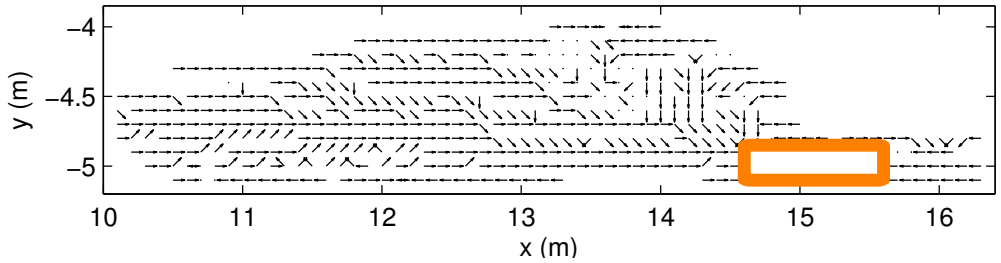


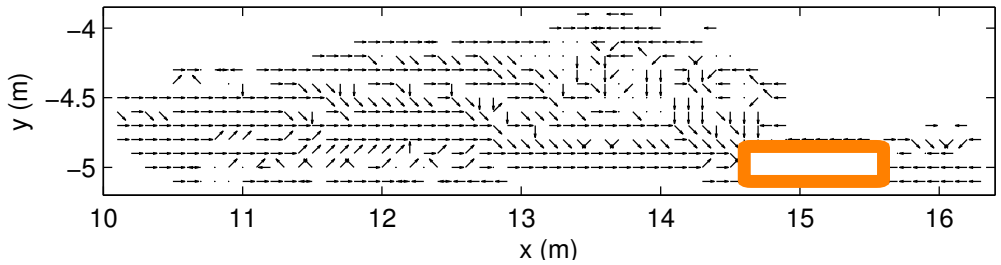
Figure 6.8 – Example policy obtained for rough terrain experiment with Δs_{head} considered, projected onto the $x - y$ plane. The goal is the empty square in the upper right corner of the figure. Arrows indicate the preferred crab actions at each state. Dots indicate rotations.



(a) Policy without considering uncertainty.

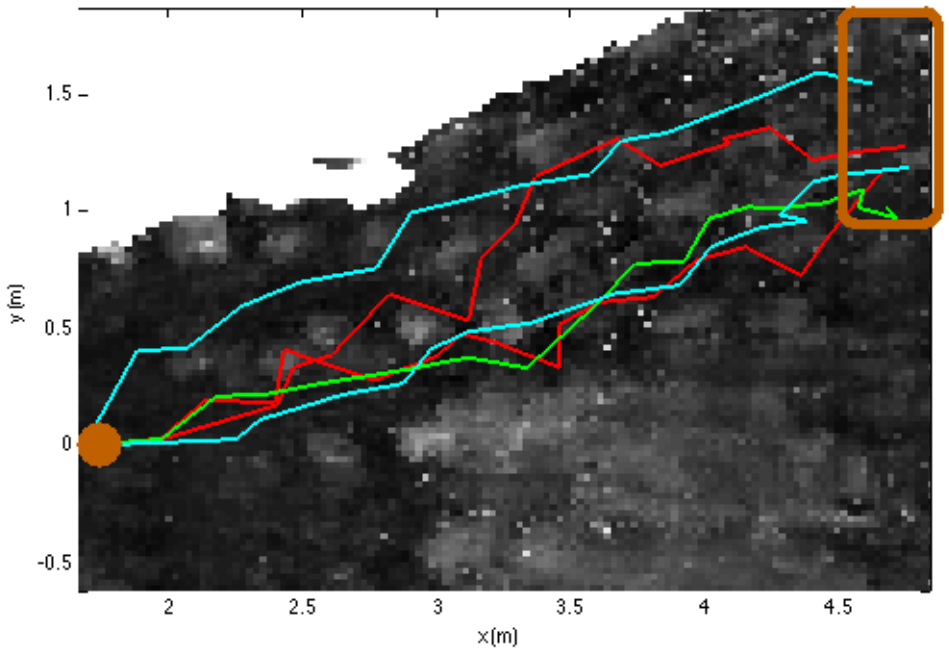


(b) Policy considering distance uncertainty.

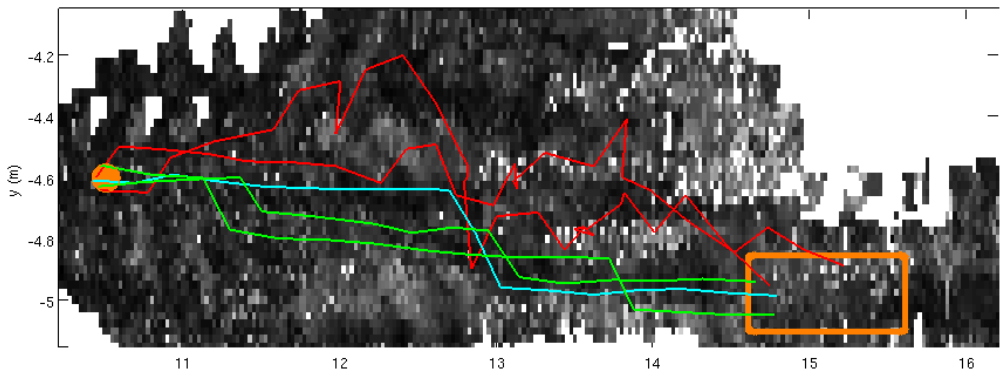


(c) Policy considering heading uncertainty.

Figure 6.9 – Motion policies over rough terrain (rock field B). Shown the for zero yaw (rover facing positive x direction), down-sampled 16 fold. The goal is the orange square in the bottom right corner.



(a) Rock field A.



(b) Rock field B.

Figure 6.10 – Examples of trajectories during the rock traversal experiments, comparing planning when considering different uncertainty sources. Shown are one or two example trajectories from each of the following motion planning methods trialled: Red: without uncertainty considered. Green: with uncertainty in heading. Cyan: with uncertainty in distance. Cost map shown in grayscale, from dark to light as cost increases. The starting position is indicated by the brown disk on the left and the goal region is shown as the brown rectangle on the right. The planner generally attempts to navigate between most of the rocks which are high cost.

Table 6.5 – Second learning results for rough terrain traversal, used for rock field B. This also compares control errors encountered during learning and testing.

Action:	crab 0π	crab $\pm\pi/4$	crab $\pm\pi/2$	crab $\pm3\pi/4$	crab π	rotate $\pm\pi/4$
Error:	δs_{head}	δs_{head}	δs_{head}	δs_{head}	δs_{head}	δs_{yaw}
Rough Terrain Learning - *marginalised by Action*						
mean (rad)	0.030	-0.013	-0.021	-0.023	0.032	-0.151
std. (rad)	0.078	0.104	0.080	0.092	0.063	0.181
# samples	21	47	42	34	17	32
Rough Terrain Planning - *marginalised by Action*						
mean (rad)	0.017	0.009	0.009	n/a	-0.011	-0.103
std. (rad)	0.139	0.129	0.101	n/a	0.061	0.27
# samples	757	251	63	29	21	152

the rock field.

Results shown in Table 6.6 indicate the policies chosen by the planning with uncertainty in this case were favourable, with less accumulated cost. In these tests, considering heading uncertainty was most significant to overall cost. This was followed by considering distance uncertainty, and not considering uncertainty in control performed the poorest. In this table, "stuck states" refers to situations where the rover dug one of its wheels in next to a rock during the test and could not initially mount the rock as the result. This occurred 20% - 30% of the time when no uncertainty was considered. When considering uncertainty in distance travelled, stuck states still occurred. However, planning with heading uncertainty again made more impact, with the least amount of stuck states. During repeated motion planning trials on rock field B, the accuracy of the elevation map computed deteriorated as the rover altered the terrain slightly each time, including exact locations of each rock. This resulted in a slight deterioration of each motion planning method. For fairness of comparison, the motion planning method was changed every 5 trails.

This trend in the data supports the claim that our approach to consider the control uncertainty can significantly improve the safety of the platform at the execution of planned trajectories.

Table 6.6 – Planning with traversal over rock field A and rock field B

Uncertainty considered	#Trials	#Stuck States	Mean Total Cost	Std. Total Cost
Rock Field A				
none	5	1	1.46	0.050
Δs_{dist},Δs_{yaw}	5	1	1.31	0.083
Δs_{head},Δs_{yaw}	5	0	1.19	0.061
Rock Field B, start position 1				
none	10	3	1.99	0.526
Δs_{dist},Δs_{yaw}	10	1	1.80	0.673
Δs_{head},Δs_{yaw}	10	0	1.54	0.354
Rock Field B, start position 2				
none	10	3	2.04	0.735
Δs_{dist},Δs_{yaw}	10	4	2.05	1.060
Δs_{head},Δs_{yaw}	10	3	1.96	0.743

Note that in these experiments we considered two sources of uncertainty independently. Considering them in combination should then have an even stronger impact on the platform safety. This will be left for future work.

Chapter 7

Conclusion and Future Work

This chapter summarises conclusions of this thesis in Sec. 7.1. Future work is discussed in Sec. 7.2.

7.1 Conclusions

Since the motion of any real mobile robot is stochastic to some degree, considering control uncertainty at the planning stage enables us to significantly enhance the safety of the platform (e.g. mitigating chances of collisions) and lower the cost accumulated on average during the execution of planned trajectories in real environments. These claims were demonstrated in this thesis using a planetary rover.

A model of uncertainty was built using learned data and Gaussian processes to predict motions over flat and unstructured terrain in a Mars-analogue environment. The trained model was then exploited to plan policies using dynamic programming, and to execute paths following the planned policies. Experimental validation was achieved both in simulation and in real experiments, in a variety of situations, taking into account uncertainties in heading and distance travelled. Our experiments compared the executed trajectories generated by planning with uncertainty with those generated by planning without considering uncertainty (deterministic control). These results show the improvement in safety and accumulated cost when accounting for uncertainty.

7.2 Future Work

Important areas of future work include the consideration of more complex terrain with larger slopes and denser collections of small rocks. Rough terrain traversals in this thesis were limited to traversing one rock at a time, however, complex terrain can necessitate learning for multiple wheels traversing different rocks simultaneously. It is also important to study the ability of our approach to learn and predict the deviations of control actions due to loose rocks that shift during traversal.

Online GP learning allows observations collected during navigation to be included in the training data. Benefits include the ability of the rover to *adapt* to new types of terrain as they are encountered. This is particularly relevant to planetary rovers, as the exact robot-terrain interaction found in a Mars environment will not be known *a priori*.

Real-time dynamic programming [6] is a further area of investigation, providing real-time updates of DP transition functions as they are learned. This allows learning computed online to update a motion policy in real-time, rather than taking effect on subsequent motion policies when a new goal region is selected.

Bibliography

[1] H. Agarwal, J. E. Renaud, E. L. Preston, and D. Padmanabhan. Uncertainty quantification using evidence theory in multidisciplinary design optimization. *Reliability Engineering and System Safety*, 85(1–3):281–294, 2004.

[2] R. Alterovitz, T. Simeon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty. In *Robotics: Science and Systems*, 2007.

[3] A. Angelova, L. Matthies, D. Helmick, and P. Perona. Learning and prediction of slip from visual information. *Journal of Field Robotics*, 24:205–231, 2007.

[4] S. Balakirsky and A. Lacaze. World modeling and behavior generation for autonomous ground vehicle. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1201–1206, 2000.

[5] R. Balakrishna and A. Ghosal. Modeling of slip for wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, pages 126–132, feb 1995.

[6] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. 1993.

[7] J. Berb, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. In *Robotics: Science and Systems*, 2010.

[8] D. Bonnafous, S. Lacroix, and T. Simeon. Motion generation for a rover on rough terrains. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[9] C. A. Brooks and K. Iagnemma. Self-supervised terrain classification for planetary surface exploration rovers. *Journal of Field Robotics, Special Issue on Space Robotics, Part I*, 29(3):445–468, 2012.

[10] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation*, 2011.

[11] B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *IEEE International Conference on Robotics and Automation*, pages 3313–3318, april 2007.

[12] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110, june 2006.

[13] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *IEEE International Conference on Robotics and Automation*, 2012.

[14] M. Quigley et al. Ros: an open-source robot operating system. In *Open-Source Software Workshop, IEEE International Conference on Robotics and Automation*, 2009.

[15] M. Greytak and F. Hover. Analytic error variance predictions for planar vehicles. In *IEEE International Conference on Robotics and Automation*, pages 471–476, may 2009.

[16] L. J. Guibas, D. Hsu, H. Kurniawati, and E. Rehman. Bounded uncertainty roadmaps for path planning. In Gregory Chirikjian, Howie Choset, Marco Morales, and Todd Murphey, editors, *Algorithmic Foundation of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 199–215. Springer Berlin / Heidelberg, 2009.

[17] D. Helmick, A. Angelova, and L. Matthies. Terrain Adaptive Navigation for planetary rovers. *Journal of Field Robotics*, 26(4):391–410, 2009.

[18] K. Iagnemma, H. Shibly, A. Rzepniewski, and S. Dubowsky. Planning and control algorithms for enhanced rough-terrain rover mobility. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[19] G. Ishigami, K. Nagatani, and K. Yoshida. Path planning for planetary exploration rovers and its evaluation based on wheel slip dynamics. In *IEEE International Conference on Robotics and Automation*, 2007.

[20] G. Ishigami, G. Kewlani, and K. Iagnemma. Statistical Mobility Prediction for Planetary Surface Exploration Rovers in Uncertain Terrain. In *IEEE International Conference on Robotics and Automation*, 2010.

[21] R. Platt Jr., R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

[22] L. P. Kaelbling, M. L. Littman, and A. R. Cassandr. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2): 99–134, 1998.

[23] H. J. Kappen. *American Institute of Physics Conference Series (AIP), Cooperative Behavior in Neural Systems*, feb .

[24] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*, 2010.

[25] S. Karumanchi, T. Allen, T. Bailey, and S. Scheding. Non-parametric learning to aid path planning over slopes. In *Robotics: Science and Systems*, 2009.

[26] S. Karumanchi, T. Allen, T. Bailey, and S. Scheding. Non-parametric learning to aid path planning over slopes. *The International Journal of Robotics Research*, 29(8):997–1018, 2010.

[27] L. Kavraki and S. LaValle. *Handbook of Robotics*. Springer, 2008.

[28] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, aug 1996.

[29] A. Krebs, C. Pradalier, and R. Siegwart. Adaptive rover behavior based on online empirical evaluation: Rover-terrain interaction and near-to-far learning. *Journal of Field Robotics*, 27(2):158–180, 2010.

[30] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. In *Robotics: Science and Systems*, 2011.

[31] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *The International Journal of Robotics Research*, 21(10-11):917–942, 2002.

[32] T. Lang, C. Plagemann, and W. Burgard. Adaptive non-stationary kernel regression for terrain modelling. In *Robotics: Science and Systems*, 2007.

[33] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[34] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. In *Technical Report*, oct 1998.

[35] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

[36] S. M. LaValle and S. A. Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *The International Journal of Robotics Research*, 17(1):19–42, 1998.

[37] P.C. Leger, A. Trebi-Ollennu, J.R. Wright, S.A. Maxwell, R.G. Bonitz, J.J. Biesiadecki, F.R. Hartman, B.K. Cooper, E.T. Baumgartner, and M.W. Maimone. Mars exploration rover surface operations: driving spirit at gusev crater. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1815–1822, Oct 2005.

[38] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402, 1995.

[39] R. McAllister, T. Peynot, R. Fitch, and S. Sukkarieh. Motion planning and stochastic control with experimental validation on a planetary rover. In *(to appear in) IEEE/RSJ International Conference on Intelligent Robots and Systems (accepted 2nd July 2012)*, 2012.

[40] P.E. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *IEEE International Conference on Robotics and Automation*, pages 1261–1267, may 2006.

[41] W. L. Oberkampf and J. C. Helton. Mathematical representation of uncertainty. In *American Institute of Aeronautics and Astronautics*, 2001.

[42] S. Patil, J. Berg, and R. Alterovitz. Motion planning under uncertainty in highly deformable environments. In *Robotics: Science and Systems*, 2011.

[43] C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard. Learning predictive terrain models for legged robot locomotion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3545–3552, sept. 2008.

[44] S. Prentice and N. Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 2009.

[45] C. E. Rasmusen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[46] P. S. Schenker, T. L. Huntsberger, P. Pirjanian, E. T. Baumgartner, and E. Tunstel. Planetary rover developments supporting mars exploration, sample return and future human-robotic colonization. *Autonomous Robots*, 14:103–126, 2003.

[47] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[48] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents. MIT Press, 2005.

[49] J. P. Underwood, A. Hill, T. Peynot, and S. J. Scheding. Error modeling and calibration of exteroceptive sensors for accurate mapping applications. *Journal of Field Robotics*, 27:2–20, 2010.

[50] G. Varadhan and D. Manocha. Accurate minkowski sum approximation of polyhedral models. *Graphical Models*, 68(4):343–355, 2006.

[51] D. Wormell, E. Foxlin, and P. Katzman. Advanced inertial-optical tracking system for wide area mixed and augmented reality systems. In *Proc. 10th International Immersive Projection Technologies Workshop/13th Eurographics Workshop on Virtual Environments*, 2007.

Appendix A

A.1 Pseudocode for Motion Planning Algorithm

This section lists pseudocode for the motion planning algorithm implemented on the test platform, to plan a motion policy over an elevation map to a goal region.

// Definitions, Data Structures

$GoalStates \doteq \{\forall\, state \in States : GoalRegion.xMin \leq state.x \leq GoalRegion.xMax,$

$GoalRegion.yMin \leq state.y \leq GoalRegion.yMax\} \in State^n$

$Transition \doteq \{newState, probability, reward\} \in \{State, \mathbb{R}, \mathbb{R}\}$

Require: $CostMap : State \to \mathbb{R}$

Require: $ElevationMap : State \to \mathbb{R}$

Require: $GoalRegion \doteq \{xMin, xMax, yMin, yMax\} \in \mathbb{R}^4$

Require: $KinematicModel : State, ElevationMap \to \phi \in \mathbb{R}, \theta \in \mathbb{R}$

Require: $P(relativeStateChange|roll, pitch, action)\ \forall\ \phi \in \Phi_{discreteValues}$, $\theta \in \Theta_{discreteValues}, action \in Actions$

// 1. build states

$buildWorld(costMap)$ // build a {x,y,yaw} matrix world of States

// 2. link states

```
startState =LOCALISE()
queue.push(startState)
while queue ≠ ∅ do
  state = queue.pop()
  for ∀ action ∈ Actions do
    st =STOCHASTICTRANSITION(state, action, elevationMap)
    if st can go out of bounds then
      continue // do not consider
    end if
    state.forwardTransitions.push(st) // parent records child
    for ∀ child ∈ st.children do
      child.backwardTransitions.push(st) // child records parent
      queue.push()
    end for
  end for
end while

// 3. dp - build poilcy
queueDP.push(∀ state ∈ GoalStates)
while queueDP ≠ ∅ do
  state = queueDP.pop()
  for ∀ st ∈ state.backwardTransitions do
    valueOffer =BELLMANVALUE(st.parentState, st.action)
    if valueOffer > st.parentState.value then
      st.parentState.value = valueOffer
      queueDP.push(st.parentState)
    end if
  end for
end while

// 4. execute policy
```

while $currentState \notin GoalStates$ **do**
  for $\forall\ st \in currentState.forwardTransitions$ **do**
    if possible that st can go of bounds **then**
      continue // do not consider
    end if
    $value =$BELLMANVALUE$(currentState, st.action)$
    if *value* is highest seen so far for *currentState* **then**
      $optimalAction = st.action$
    end if
  end for
  $controls.$EXECUTE$(optimalAction)$
  $currentState =$LOCALISE$()$
end while

// Function Definitions

STOCHASTICTRANSITION(state, action, elevationMap):
$\phi = kinematicModel.$ESTIMATEPHI$(state, elevationMap)$
$\theta = kinematicModel.$ESTIMATETHETA$(state, elevationMap)$
$P(newStates) = state + P(relativeStateChange|\phi, \theta, action)$
$transitions = \emptyset$
for $\forall\ newState \in P(newStates)$ **do**
  $reward =$ average cost of states in between *state* and *newState*
  $transitions.push(\{newState, P(newState), reward\})$
end for
return *transitions*

BELLMANVALUE(state, action):
for $\forall\ t \in state.forwardTransitions[action.ID].transitions$ **do**
  $expectedReward+ = t.probability \times (t.newState.value - t.reward)$

end for

return *expectedReward*

A.2 Pseudocode for Kinematics Model Algorithm

This section lists pseudocode for the kinematics model algorithm used to model the Rocker-bogie test platform on an elevation map. Some variable definitions are found in Fig. 5.2.

Require: $elevationMap : State \rightarrow \mathbb{R}$

Require: *nodeArrangements* (vector of all pin joint nodes (inc. wheel) locations to robot's point position when on flat ground)

Require: *States* (input state to query the robot kinematics)

pose is a set of robot's position, orientation angles, chassis configuration angles

$[h1, h2, h3, h4, h5, h6] = getWheelElevations(wheelPositions, elevationMap)$

while $|max(elevationOfEachWheel)| >$ a small positive number **do**

// drop simulated robot

$pose.height- = min(h1, h2, h3, h4, h5, h6)$

$reconfigureNodes(pose)$

// correct the pitch

$pose.pitch- = asin((avg(h1, h4)/avg(h3, h6))/robot.length)$

$reconfigureNodes(pose)$

// correct the roll

$pose.roll- = asin((avg(h1, h2, h3)/avg(h4, h5, h6))/robot.width)$

$reconfigureNodes(pose)$

// correct the rocker-right pin-joint

$pose.alpha1+ = asin((h1/avg(h2, h3))/L_4)$

// correct the rocker-left pin-joint

$pose.alpha2+ = asin((h4/avg(h5, h6))/L_4)$

// correct the right-bogie pin-joint

$pose.alpha3+ = asin((h2/h3)/L_2)$

```
    // correct the left-bogie pin-joint
    pose.alpha3+ = asin((h5/h6)/L_2)
    if Any value in pose is NaN then
      mark state as obstacle
      break
    end if
  end while
  return pose


  // Function Definitions
  RECONFIGURENODES(pose):
  // Function which updates node positions based on robot pose
  // rotation for vehicle orientation
  rotationMatrix = getRotationMatrix(pose.orientationAngles)
  for ∀ node ∈ nodes do
    node = rotate(rotationMatrix, pose.position, pose.position+nodeArrangement(node))
  end for
  // rotation for joints in frame
  // add root nodes that never change their position with repect to pose.position
  queue.add(nodes.roots)
  while queue ≠ ∅ do
    node = queue.pop()
    if node.isWheel() then
      continue
    end if
    // find location of child nodes
    for ∀ childNode ∈ node.children do
      childNode = rotate(node.pinJointAngle, node, node+(nodeArrangement(childNode)−
      nodeArrangement(node)))
      queue.add(childNode)
```

end for

end while

$wheelElevations = getElevationOfEachWheel(nodes.wheelPositions, elevationMap)$

return $wheelElevations, nodes$