Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2021

Learning-based Motion Planning and Control of a UGV With Unknown and Changing Dynamics

Åke Johansson & Joel Wikner

LINKÖPING
UNIVERSITY

Master of Science Thesis in Electrical Engineering

Learning-based Motion Planning and Control of a UGV With Unknown and Changing Dynamics

Åke Johansson & Joel Wikner

LiTH-ISY-EX--21/5399--SE

Supervisor: **Kristoffer Bergman**
Saab Dynamics AB
Anja Hellander
ISY, Linköpings universitet

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Research about unmanned ground vehicles (UGVs) has received an increased amount of attention in recent years, partly due to the many applications of UGVs in areas where it is inconvenient or impossible to have human operators, such as in mines or urban search and rescue. Two closely linked problems that arise when developing such vehicles are motion planning and control of the UGV.

This thesis explores these subjects for a UGV with an unknown, and possibly time-variant, dynamical model. A framework is developed that includes three components: a machine learning algorithm to estimate the unknown dynamical model of the UGV, a motion planner that plans a feasible path for the vehicle and a controller making the UGV follow the planned path.

The motion planner used in the framework is a lattice-based planner based on input sampling. It uses a dynamical model of the UGV together with motion primitives, defined as a sequence of states and control signals, which are concatenated online in order to plan a feasible path between states. Furthermore, the controller that makes the vehicle follow this path is a model predictive control (MPC) controller, capable of taking the time-varying dynamics of the UGV into account as well as imposing constraints on the states and control signals. Since the dynamical model is unknown, the machine learning algorithm Bayesian linear regression (BLR) is used to continuously estimate the model parameters online during a run. The parameter estimates are then used by the MPC controller and the motion planner in order to improve the performance of the UGV.

The performance of the proposed motion planning and control framework is evaluated by conducting a series of experiments in a simulation study. Two different simulation environments, containing obstacles, are used in the framework to simulate the UGV, where the performance measures considered are the deviation from the planned path, the average velocity of the UGV and the time to plan the path. The simulations are either performed with a time-invariant model, or a model where the parameters change during the run.

The results show that the performance is improved when combining the motion planner and the MPC controller with the estimated model parameters from the BLR algorithm. With an improved model, the vehicle is capable of maintaining a higher average velocity, meaning that the plan can be executed faster. Furthermore, it can also track the path more precisely compared to when using a less accurate model, which is crucial in an environment with many obstacles. Finally, the use of the BLR algorithm to continuously estimate the model parameters allows the vehicle to adapt to changes in its model. This makes it possible for the UGV to stay operational in cases of, e.g., actuator malfunctions.

Acknowledgments

First of all, we would like to thank our supervisors, Kristoffer Bergman and Anja Hellander. We greatly appreciate all the time you have spent on helping us during our thesis, ranging from our discussions regarding the technical details, to the countless times you have read and given feedback on our report. A big thanks to our examiner, Daniel Axehill, for your help during the thesis.

We would also like to thank Torbjörn Crona for letting us do our thesis at at Saab Dynamics, and allowing us to freely choose the directions of the thesis.

Furthermore, we are grateful to our “Saab-fadder” Signe Dahlin for making us feel at home at Saab and for checking in on us to see how we were doing, as well as to all the employees at GN&C at Saab for lighting up our lunch breaks. We also like to thank our master’s thesis colleagues at Saab for our lunches and walks together.

Finally, a huge thanks to our friends and families for your everlasting and unconditional support throughout the years.

Linköping, June 2021
Åke Johansson & Joel Wikner

Contents

Notation

Mathematical notation

Notation	Meaning
$\boldsymbol{x}$	State vector sequence
$\boldsymbol{u}$	Control signal sequence
$\boldsymbol{X}$	Explanatory variables
z	Response variable
$\boldsymbol{\beta}$	Regression parameters
σ^2	Variance of a Normal distribution
$\mathcal{D}$	Measured data
N	Prediction horizon in MPC problem
M	Number of data points
$p(y\|x)$	Probability distribution of y given x
$\mathcal{N}(x\|\mu, \sigma^2)$	Normal distribution of x with mean μ and variance σ^2
$IG(x\|a_0, b_0)$	Inverse Gamma distribution of x with shape a_0 and scale b_0
$\mathcal{T}(x\|\mu, \sigma^2, \nu)$	Student's t-distribution of x with ν being the degrees of freedom
n_0	Number of effective data points attributed to the prior
$\mathcal{X}$	Permissible state values
$\mathcal{U}$	Permissible control signal values
K	Number of motion primitives used in the planned path
$\boldsymbol{m}$	Motion primitive – defined as a sequence of states and control signals
L	Length of a motion primitive
$h(\boldsymbol{x})$	Heuristic

Abbreviations

Abbreviation	Meaning
BLR	Bayesian linear regression
GP	Gaussian process
IPOPT	Interior point optimiser
LQR	Linear-quadratic regulator
MPC	Model predictive control
NIG	Normal inverse Gamma
OCP	Optimal control problem
ROS	Robot Operating System
UGV	Unmanned ground vehicle
WBLR	Weighted Bayesian linear regression

1 Introduction

The goal of this thesis is to develop a motion planning and control framework for an unmanned ground vehicle (UGV) with a partially unknown and time-variant dynamical model of the system. In order to increase the controller performance, the model is estimated online and then used by the motion planner and the controller. This thesis is conducted at Saab Dynamics AB in Linköping.

1.1 Background

The use of UGVs has surged in the last years, partly due to the large number of applications [30]. For example, UGVs can be used for mapping of large areas, searching terrain that could be hostile to humans and for transportation of goods [11], [15]. When developing the control architecture for such vehicles, there are generally three main components to consider [17]:

- The overall *mission planner*, giving the vehicle directions on what to do and where to go.
- The *motion planner*, computing a feasible path for the vehicle in order to move to the position provided by the mission planner.
- The *controller*, making sure that the vehicle follows the path given by the motion planner.

In this thesis, focus lies on the last two components, the motion planner and the controller, as well as an essential aspect of these: the *dynamical model* of the system, i.e. the differential equations describing the evolution of the states of the system. Since a large part of modern planning and control depends on known, accurate mathematical models describing the system dynamics, precise knowledge of the models can greatly improve the overall system performance when used correctly [12].

In some applications, the dynamical model of the system can be partially or completely unknown. Another possible case is that the model might change during a run, due to unforeseen events such as actuator or battery malfunctions. In such cases, machine learning algorithms can be used for identification of the dynamical model of the system. Recent results in machine learning have provided many effective methods for learning the dynamics of the system online, resulting in high-performance control. However, there exist some concerns in spite of the promising result, as these methods usually do not guarantee stability of the system [5]. This has resulted in multiple methods where classical control is combined with learning-based methods to improve performance while still taking safety into account [12]. One such method that has been studied lately is combining learning-based methods with a model predictive control (MPC) controller.

MPC is a powerful control methodology which optimises the control input at the current timestep, while also taking future steps into consideration, thus generating an optimal control sequence that satisfies a set of constraints. In order to accomplish this, a dynamical model of the system is used, and as a consequence the performance of the MPC controller is inherently linked to how well the model captures the true dynamics of the system [28]. Difficulties in modelling and estimating the system dynamics can limit the performance and usability of the MPC controller, paving the way for alternative methods for system identification [19]. Relating this task to machine learning, it is possible to use learning-based methods, exploiting system data to improve the dynamical model, while still taking safety constraints into consideration by using MPC as in [9] and [6].

The MPC controller is commonly used to follow, or track, a reference, e.g. the middle of the lane on a road. However, in certain environments there are no obvious references, for example in a parking lot or a mine. These environments are commonly referred to as *unstructured environments*, and within these, a reference path has to be synthesised in some way. This is where another area related to automatic control is helpful: *motion planning*. In recent years, *sampling-based* motion planning using differential constraints has been an important area of research due to interest in technologies such as self-driving cars [2]. The goal for such motion planners is to compute feasible paths with respect to the dynamical model of the system that is supposed to follow them, and to avoid collision with obstacles. The dynamical model of the system is used during the planning to forward simulate the system for certain control inputs [16]. Here, in similarity to the MPC controller, the dynamical model of the system is crucial in order to achieve good performance, which further motivates the use of learning-based methods to improve the dynamical model of the system.

1.2 Problem formulation

The objective of this thesis is to develop a motion planning and control framework for a UGV that has unknown and time-varying system dynamics, using learning-based methods to estimate and continuously improve the dynamical model of the system. The system considered in this thesis is a tracked UGV, called *Rover*, which has a partially unknown dynamical model, the *nominal model*, that is to be estimated and improved. The improved model will be used both within an MPC controller and a motion planner.

When choosing an approach, practical details such as computational time, simplicity and performance need to be taken into account. For example, the controller needs to send control signals at a sufficiently high rate to be able to react to and suppress disturbances, while the memory typically is limited on a UGV platform. Hence, a fine balance between complexity and performance is required.

The questions to be answered are:

- What methods can be used to identify the unknown dynamics of the system?
- Can the estimated model be used in sampling-based motion planning algorithms, and will it improve performance compared to using the nominal model?
- How does the performance of an MPC controller based on an estimated model compare to the performance of an MPC controller based on the nominal model?
- Can a machine learning algorithm be used to detect a dynamic change in the system model while the system is running, and adapt the model to this change?

1.3 Related work

The research that considers the problem of learning the model dynamics is concentrated on two different approaches using either a *robust* or a *stochastic* model of the system dynamics. According to [12], the robust approaches typically require a hard a priori bound to cover all uncertainty, while stochastic methods take the distributional information on the system uncertainty into account. Since covering all uncertainties with an a priori bound can be conservative in practice, this thesis focuses on the stochastic models instead.

The stochastic methods use either a *parametric* or a *nonparametric* model. The parametric models are used to find a set of parameter values that is consistent with the observed data to describe the system dynamics. Nonparametric approaches on the other hand, try to contain the function describing the system dynamics within bounds by forming a function estimate based on possibly noisy observations [12].

A commonly used approach for learning-based control is to use Gaussian process (GP) regression, which is a nonparametric approach, as in [26]. Another approach is to use the parametric method Bayesian linear regression (BLR) combined with an MPC to control a system [18]. A tutorial-like example is [13], where the method is applied to the calibration of a sonic nozzle. In [18], a weighted version of the BLR algorithm was implemented to model certain unknown parts of the system dynamics, in order to control a skid-steered ground vehicle tracking a given path.

In order to generate such a path for a system to follow, a motion planner is often used. An example of a sampling-based motion planner is the so-called *lattice-based* motion planning algorithm [3]. This method uses *motion primitives* to span a graph of possible paths by forward simulating the system given control inputs. This motion planner can advantageously be combined with learning-based methods for system identification in order to improve the result of the forward simulation. When a graph exists, graph-searching methods are employed in order to concatenate motion primitives during online planning to find collision-free and dynamically feasible paths for the system. One approach can be seen in [7], where an algorithm called hybrid A* is used.

1.4 Outline

Chapter 2 of this thesis is intended to introduce the relevant theory and background for the work. The system dynamics and the MPC controller will be investigated, along with the Bayesian linear regression algorithm as well as the motion planning. Chapter 3 gives an overall description of the whole implemented system, together with the simulators used. Chapter 4 covers the implementation of the learning-based planning and control subsystems, while Chapter 5 presents the obtained results. The discussion regarding the methods and the results is seen in Chapter 6, while the conclusions and future work are presented in Chapter 7.

1.5 Individual contributions

Joel has developed the MPC controller and implemented the known model simulation, while Åke has developed the BLR algorithm and implemented the unknown model simulation. Both have worked on the lattice-based motion planner.

This chapter, the performance evaluations in Chapter 5, the analysis of the results in Chapter 6 and the conclusion in Chapter 7 were written by both authors. The writing of the theory in Chapter 2 and the implementation details in Chapter 3 and Chapter 4 was divided between the authors according to their respective responsibilities.

2 Theory

This chapter gives an overview of the theory used in this thesis. First, a general dynamical model of a system is presented, followed by a closer look at a method to improve the model of the dynamics using a learning-based algorithm. After that, the model predictive controller is covered. Finally, the theory behind the motion planner is described.

2.1 Dynamical model

A dynamical model of a system can generally be expressed as:

$$\dot{\boldsymbol{x}} = f_c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\beta}) \tag{2.1}$$

where $\boldsymbol{x}$ is the state of the system, $\boldsymbol{u}$ is the control signals sent to the actuators and $\boldsymbol{\beta}$ is a set of model parameters, that could be unknown or time-varying. The function $f_c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\beta})$ is the continuous *system function*, describing the dynamical model of the system. Typical state variables of a system are the position in the plane, the heading, the velocity and the angular velocity (turn rate). In order to estimate model parameters that are unknown, methods described in Section 2.2 can be used.

2.2 Bayesian linear regression

This section covers the theory behind the Bayesian linear regression, which is a machine learning algorithm that can be used to identify and improve the system model. First, the standard approach within regression analysis when dealing with linear functions, namely linear regression, is described. This is followed by a detailed description of BLR and how the method can be applied.

2.2.1 Linear regression

Linear regression is a method used to model a linear relationship between a response variable and one or more explanatory variables [21]. The goal of the method is to find the so-called *regression parameters*, $\boldsymbol{\beta} = [\beta_0, \beta_1, ..., \beta_d]$, that best fit a linear function to the available data, given a model of the form:

$$z = \beta_0 + \sum_{i=1}^{d} \beta_i X_i + \epsilon, \tag{2.2}$$

where z is the response variable, or simply the *output*, $\mathbf{X} = [X_1, X_2, ..., X_d]$ is a vector containing the explanatory variables, also called the *input vector*, and ϵ is an unobserved random variable. If $\epsilon \sim \mathcal{N}(0, \sigma^2)$, i.e. a normally distributed random variable with mean zero and variance σ^2, then the problem is a so-called *Normal* linear regression problem [21].

Furthermore, linear regression can be slightly manipulated to also model non-linearities by replacing the input vector, $\boldsymbol{X}$, with a non-linear function of these inputs. The non-linear function is often referred to as a *basis function*, noted $\boldsymbol{\phi}(\mathbf{X})$. The model then becomes:

$$z = \beta_0 + \sum_{i=1}^{d} \beta_i \phi_i(\boldsymbol{X}) + \epsilon. \tag{2.3}$$

Note that the model is still linear in the regression parameters.

2.2.2 Bayesian point of view

As mentioned in Section 2.2.1, the goal of linear regression is to find the values of the regression parameters that best fit a linear function to the available data. Linear regression methods such as maximum likelihood estimation compute numerical values for these parameters. However, such methods do not account for any uncertainty in the parameter estimation. This is where *Bayesian* linear regression is useful, since it does exactly this. The outcome of a Bayesian linear regression is a probability distribution of the regression parameters and the model uncertainty, therefore including any uncertainties that are present.

In mathematical terms, this means that BLR estimates the distribution $p(\boldsymbol{\beta}, \sigma^2 | \mathcal{D})$, i.e. how the regression parameters, $\boldsymbol{\beta}$, and the model uncertainty, σ^2, are distributed, given a set of data, $\mathcal{D}$, that contains the observed response variables and explanatory variables. This can be used in an algorithm to make predictions on the output, z, given a new input vector, $\boldsymbol{X}$, and previously observed data, $\mathcal{D}$, i.e. to estimate the distribution $p(z|\boldsymbol{X}, \mathcal{D})$. The algorithm assigns a probability distribution to z, making it possible to assess the uncertainty in the model. If a numerical value is required, z can simply be estimated as the expected value of its probability distribution.

Assuming that the random variable ϵ is normally distributed as in Section 2.2.1, the distribution of the output can be written as [21]:

$$p(z|\boldsymbol{X}, \boldsymbol{\beta}, \sigma^2) = \mathcal{N}(z|\boldsymbol{\beta}^T \boldsymbol{X}, \sigma^2). \tag{2.4}$$

In order to find the predictive distribution for the output given a new input and previously observed data, the marginalisation rule for probabilities, $p(x) = \int p(x, y)dy$, can be applied. This gives:

$$p(z|\boldsymbol{X}, \mathcal{D}) = \int p(z, \boldsymbol{\beta}|\boldsymbol{X}, \mathcal{D})\, d\boldsymbol{\beta}. \tag{2.5}$$

The term inside the integral can be expanded using the product rule, $P(A, B) = P(A|B)P(B)$, which gives:

$$p(z|\boldsymbol{X}, \mathcal{D}) = \int p(z|\boldsymbol{X}, \boldsymbol{\beta}, \mathcal{D})p(\boldsymbol{\beta}|\boldsymbol{X}, \mathcal{D})\, d\boldsymbol{\beta}, \tag{2.6}$$

where the term $p(z|\boldsymbol{X}, \boldsymbol{\beta}, \mathcal{D})$ can be simplified to $p(z|\boldsymbol{X}, \boldsymbol{\beta})$ since the output does not depend on previous data $\mathcal{D}$, and the term $p(\boldsymbol{\beta}|\boldsymbol{X}, \mathcal{D})$ is the distribution of the parameters, which can also be simplified to $p(\boldsymbol{\beta}|\mathcal{D})$ since the parameter distribution is independent of the current input, $\boldsymbol{X}$. The simplified expression for the distribution of the output from (2.6) can then be written as:

$$p(z|\boldsymbol{X}, \mathcal{D}) = \int p(z|\boldsymbol{X}, \boldsymbol{\beta})p(\boldsymbol{\beta}|\mathcal{D})\, d\boldsymbol{\beta}. \tag{2.7}$$

Using Bayes' theorem, $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, to expand the second term in the integral in (2.7) results in the expression:

$$p(\boldsymbol{\beta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\beta})p(\boldsymbol{\beta})}{p(\mathcal{D})}. \tag{2.8}$$

The term $p(\mathcal{D}|\boldsymbol{\beta})$ in the numerator can be rewritten using the *likelihood function* as [21]:

$$p(\mathcal{D}|\boldsymbol{\beta}) = \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}). \tag{2.9}$$

where the likelihood function, $\mathcal{L}(\boldsymbol{\beta}|\mathcal{D})$, is the probability of obtaining a sample of the data, given a set of regression parameters, $\boldsymbol{\beta}$.

Since the model outputs are assumed to be independent, identically distributed input-output pairs, the likelihood of a parameter vector, $\boldsymbol{\beta}$, is the product of the individual probabilities:

$$\mathcal{L}(\boldsymbol{\beta}|\mathcal{D}) = \prod_{i=1}^{M} p(z_i|\boldsymbol{X}_i, \boldsymbol{\beta}) = \prod_{i=1}^{M} \mathcal{N}(z_i|\boldsymbol{\beta}^T \boldsymbol{\phi}(\boldsymbol{X}_i), \sigma^2), \tag{2.10}$$

where M is the number of data points.

2.2.3 Prior and posterior distributions

The second term in the numerator in (2.8), $p(\boldsymbol{\beta})$, describes the probability distribution of the parameters. This term is referred to as the *prior* and is an estimation of the parameter distribution before obtaining a new observation. The interpretation of the prior is that it is used as a way to bring already known knowledge about the system into the problem solution [21]. Hence, a good strategy to find a suitable prior is to use previous information about the system, e.g. previously conducted experiments [13].

The term on the left-hand side of (2.8), $p(\boldsymbol{\beta}|\mathcal{D})$, is called the *posterior* and contains an updated belief about the parameter distribution based on the information that has been acquired from the data points used in the regression. The posterior is equal to the product of the likelihood-function and the prior, normalised by a normalisation constant, $p(\mathcal{D})$. This normalisation constant is sometimes referred to as the *marginal likelihood* or the *model evidence* and can be expressed as:

$$p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{\beta})p(\boldsymbol{\beta})\, d\boldsymbol{\beta}. \tag{2.11}$$

This integral is generally analytically intractable, and is also often difficult to compute numerically [33], meaning the posterior distribution is in general not possible to compute. However, for some priors it is possible to compute the posterior distribution using *conjugate priors*. A conjugate prior is a certain prior distribution resulting in a posterior distribution of the same family as the prior distribution – e.g. if the conjugate prior is normally distributed, the posterior will also be normally distributed, albeit with different distribution parameters such as the variance [33]. By using conjugate priors, the marginal likelihood does not need to be computed, and instead distribution parameters are assigned new values according to distribution-specific update schemes, such as the one seen in Section 2.2.4.

Different priors are used in different situations, and in the case of a Normal linear regression problem with unknown noise variance as in (2.2) and the likelihood specified in (2.10), the conjugate prior is distributed according to a *Normal inverse Gamma* (NIG) distribution [21], defined as:

$$p(\boldsymbol{\beta}, \sigma^2) = NIG(\boldsymbol{\beta}, \sigma^2|\boldsymbol{\beta}_0, \boldsymbol{V}_0, a_0, b_0). \tag{2.12}$$

This distribution is actually two distributions combined, the Normal distribution and the inverse Gamma distribution, which can be written as individual distributions:

$$\begin{aligned} p(\sigma^2) &= IG(\sigma^2|a_0, b_0), \\ p(\boldsymbol{\beta}|\sigma^2) &= \mathcal{N}(\boldsymbol{\beta}|\boldsymbol{\beta}_0, \sigma^2\boldsymbol{V}_0), \end{aligned} \tag{2.13}$$

where $\boldsymbol{\beta}_0$ is a prior vector containing the expected values of the regression parameters, while $\boldsymbol{V}_0$ is the prior covariance matrix. The two prior parameters a_0 and b_0 are the shape and scale parameters of the inverse Gamma distribution respectively. To clarify, in total the prior consists of the expected values of the regression parameters, $\boldsymbol{\beta}_0$, the covariance matrix, $\boldsymbol{V}_0$, and the shape and scale parameters of the inverse Gamma distribution, a_0 and b_0.

2.2.4 Resulting posterior distribution

Using the prior in (2.13), the posterior will be NIG distributed (since the conjugate prior also is NIG distributed), but with new parameters. For M observed data points, the posterior distribution can be written as:

$$\begin{aligned} p(\boldsymbol{\beta}, \sigma^2|\mathcal{D}) &= NIG(\boldsymbol{\beta}, \sigma^2|\boldsymbol{\beta}_M, \boldsymbol{V}_M, a_M, b_M) \\ &= \mathcal{N}(\boldsymbol{\beta}|\boldsymbol{\beta}_M, \sigma^2 \boldsymbol{V}_M) IG(\sigma^2|a_M, b_M), \end{aligned} \tag{2.14}$$

where the parameters are updated according the following scheme [21]:

$$\begin{aligned} \boldsymbol{V}_M &= (\boldsymbol{V}_0^{-1} + \tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}})^{-1}, \\ \boldsymbol{\beta}_M &= \boldsymbol{V}_M(\boldsymbol{V}_0^{-1} \boldsymbol{\beta}_0 + \tilde{\boldsymbol{X}}^T \boldsymbol{z}), \\ a_M &= a_0 + \frac{M}{2}, \\ b_M &= \frac{1}{2}(\boldsymbol{\beta}_0^T \boldsymbol{V}_0^{-1} \boldsymbol{\beta}_0 + \boldsymbol{z}^T \boldsymbol{z} - \boldsymbol{\beta}_M^T \boldsymbol{V}_M^{-1} \boldsymbol{\beta}_M) \end{aligned} \tag{2.15}$$

where $\boldsymbol{V}_M$ is the posterior covariance matrix for the regression parameters, $\boldsymbol{\beta}_M$ the posterior mean of the regression parameters and a_M and b_M the two posterior parameters for the inverse Gamma distribution. The matrix $\tilde{\boldsymbol{X}}$ consists of the newly observed input vectors, while the vector $\boldsymbol{z}$ contains the corresponding outputs.

The posterior marginal distribution, given a dataset, for the variables in the joint distribution in (2.14) can be computed using the marginalisation rule. This gives the posterior marginal for the variance, σ^2, and the posterior marginal for the regression parameters, $\boldsymbol{\beta}$, which are given by the following distributions [21]:

$$\begin{aligned} p(\boldsymbol{\beta}|\mathcal{D}) &= \mathcal{T}(\boldsymbol{\beta}|\boldsymbol{\beta}_M, \frac{b_M}{a_M} \boldsymbol{V}_M, 2a_M), \\ p(\sigma^2|\mathcal{D}) &= IG(\sigma^2|a_M, b_M), \end{aligned} \tag{2.16}$$

where $\mathcal{T}$ is the Student's t-distribution. Both of these distributions use parameters from the update scheme in (2.15).

When the posterior is computed, it is used as prior for the next iteration of (2.15) in the BLR algorithm, using new data. Thus, the algorithm can be used with continuous streaming data, e.g. from sensors on a system, typically computing the new posterior each time new data is observed. Figure 2.1 shows how the distribution of the parameters changes as new data points are added.

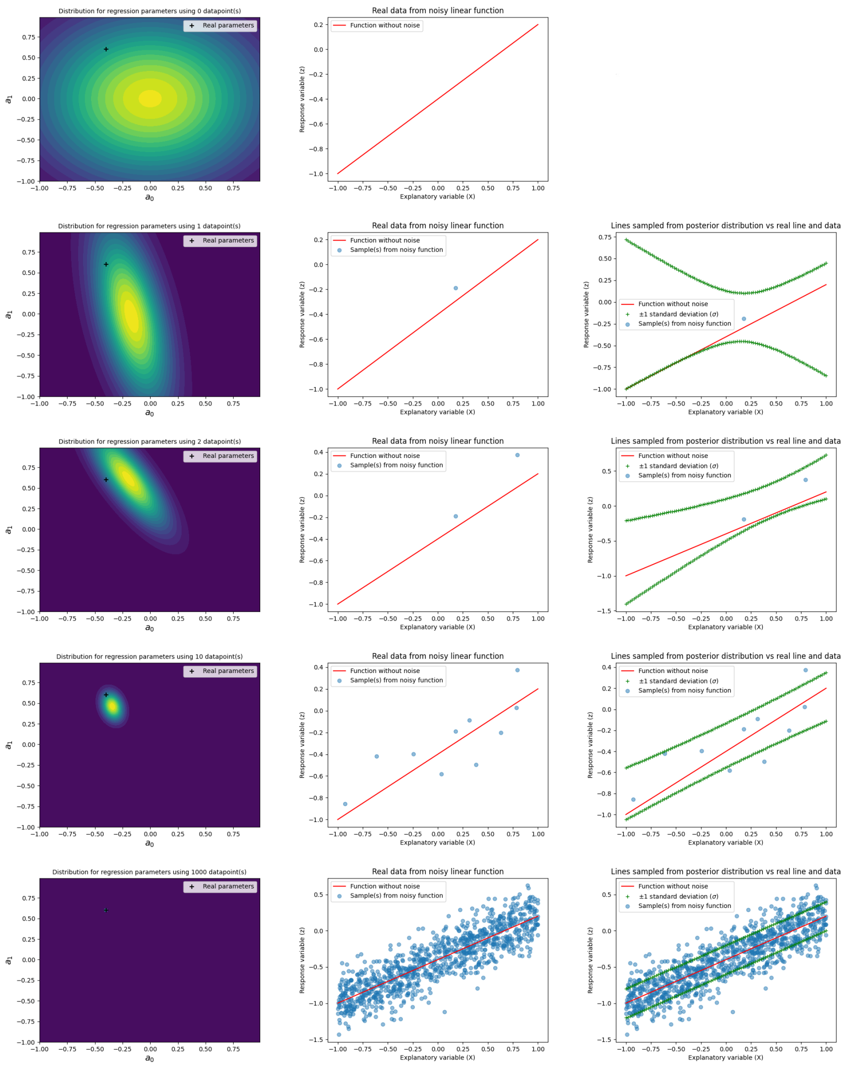

Figure 2.1: *This figure illustrates what happens when adding new data points to the regression algorithm. The response variable z and the explanatory variable X, are sampled randomly from a linear function of form $z = a_1X + a_0$ with added normally distributed noise, ϵ. The regression parameters are a_0 and a_1. The left-hand column shows the probability distribution of the regression parameters – yellow indicates a high probability while dark blue represents a low probability. The middle column shows the function without noise and the drawn samples. Finally, the right-hand column also shows the upper and lower bound one standard deviation away from the mean for a given X.*

2.2.5 Weighted Bayesian linear regression

For some applications, it is desirable to be able to determine if the model parameters dynamically have changed online during a run, and in that case also identify the new parameters. A problem with the current prior/posterior configuration and update scheme is that the BLR algorithm uses all the available previous data – which will be outdated if a dynamic change of the model parameters happens. If this is the case, it would be desirable to forget the previous data and instead use more recent data, obtained after the dynamic change, that better reflect the new model parameters. One solution to this problem is the use of *weighted* Bayesian linear regression (WBLR), an extension to the update scheme in (2.15) as follows [18]:

$$\begin{aligned}
\boldsymbol{V}_{0^*} &= \frac{n_0+1}{n_0}\boldsymbol{V}_M,\\
\boldsymbol{\beta}_{0^*} &= \boldsymbol{\beta}_M,\\
a_{0^*} &= \frac{n_0}{n_0+1}a_M,\\
b_{0^*} &= \frac{n_0}{n_0+1}b_M,
\end{aligned} \tag{2.17}$$

where n_0 is a user-defined value that determines the number of effective data points that are being attributed to the prior. Essentially, n_0 allows the algorithm to "forget" previous data points, solving the issue with too much old information in the prior [18]. The variables $(\cdot)_{0^*}$ denotes the prior for the next iteration of the algorithm.

Assigning a large value to n_0 means that a larger number of data points is used, resulting in a smaller difference between $(\cdot)_M$ and $(\cdot)_{0^*}$, leading to more smooth estimates of the model parameters [18]. When instead a small value of n_0 is used, fewer data points are included in the prior, resulting in less smooth parameter estimates but allowing them to change quicker [18]. The update scheme in (2.15) is always used, and if the algorithm starts with fewer than n_0 data points attributed to the prior, the posterior of the current data point will be the prior for the next data point, as mentioned in Section 2.2.4. When the number of data points attributed to the prior reaches or exceeds n_0, the next prior will be computed using the equations in (2.17). The parameter n_0 makes it possible to tune how fast the system adapts to new conditions in a computationally cheap manner. In GP regression on the other hand, new data points have two options, either to displace an already existing data point, or to increase the size of the model – which will increase the computational time, making it less flexible as more data points are added [18].

2.3 Model predictive control

In model predictive control, the control input to a system is computed by repeatedly solving an optimal control problem (OCP) over a finite time horizon. In order to solve this problem, the MPC controller uses a dynamical model of the system, a *prediction model*, while imposing constraints on the states and the control signals, as mentioned in Chapter 1.

2.3.1 Prediction model

The prediction model is what the controller needs in order to take future timesteps into account when computing an optimal control sequence. The model presented in (2.1) is discretised with timestep length Δt and used in the MPC formulation. It is generally written on the form [25]:

$$\boldsymbol{x}_{j+1} = f(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{\beta}_j) \tag{2.18}$$

where the sub-index j is the current timestep. The variable $\boldsymbol{x}_j$ is the vector containing the current states and $\boldsymbol{u}_j$ is the current input vector. The discrete function $f(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{\beta}_j)$ can be either linear or non-linear [25]. If the optimal control problem for the MPC controller is non-convex, which is the case when using a non-linear prediction model, the solution cannot be guaranteed to be *globally* optimal using standard non-linear programming [22]. Instead, *locally* optimal solutions are computed, which may affect the performance of the controller adversely.

If the model parameters, $\boldsymbol{\beta}$, are unknown, they can for example be estimated as regression parameters using the BLR algorithm described in Section 2.2.

2.3.2 Formulating the control problem

One crucial aspect of the MPC is the cost function, here denoted J_N, which should be minimised by the controller, given a set of constraints on $\boldsymbol{x}$ and $\boldsymbol{u}$. For the MPC controller, J_N usually consists of N terms, where N is the so-called *prediction horizon*. The minimisation of the cost function is generally not analytically tractable, meaning that the MPC problem needs to be solved numerically by an optimiser [25]. By minimising the cost function J_N over the control signal, an optimal control sequence is generated for the N next samples. The MPC problem can be formulated as:

$$\begin{aligned}
\underset{\{\boldsymbol{u}_k\}_{k=0}^{N-1}}{\text{minimise}} \quad & J_N = \sum_{k=0}^{N-1} \ell(\boldsymbol{x}_k, \boldsymbol{u}_k) + \ell_N(\boldsymbol{x}_N, \boldsymbol{u}_N) && \text{(2.19a)} \\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\beta}_k), k = 0, \dots, N-1 && \text{(2.19b)} \\
& \boldsymbol{x}_k \in \mathcal{X}, k = 1, \dots, N-1, && \text{(2.19c)} \\
& \boldsymbol{x}_0 = \hat{\boldsymbol{x}}, && \text{(2.19d)} \\
& \boldsymbol{u}_k \in \mathcal{U}, k = 0, \dots, N-1 && \text{(2.19e)}
\end{aligned}$$

where $\ell(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is the stage cost, $\ell_N(\boldsymbol{x}_k, \boldsymbol{u}_k)$ the terminal cost, $\mathcal{X}$ represents the closed set of permissible state values, $\hat{\boldsymbol{x}}$ is a measurement, or estimate, of the state at timestep j, and $\mathcal{U}$ is a compact set of permissible control signals [26].

The result from solving (2.19) is an optimal control sequence denoted $\{\boldsymbol{u}_k^\star\}_{k=0}^{N-1}$. The first value in this sequence, $\boldsymbol{u}_0^\star$, is sent as input to the system at the current timestep, after which a time update is performed [8]. The algorithm can be summarised as:

1. Measure or estimate the state, $\hat{\boldsymbol{x}}$, at timestep j.
2. Compute the optimal control sequence, $\{\boldsymbol{u}_k^\star\}_{k=0}^{N-1}$, by solving the optimisation problem in (2.19).
3. Send the first element, $\boldsymbol{u}_0^\star$, of the control sequence as input to the system.
4. Time update, $j := j + 1$.
5. Return to step 1.

A possible cost function in (2.19) is of the quadratic type, for example:

$$J_N = \sum_{k=0}^{N-1} \|\boldsymbol{x}_k^{\text{ref}} - \boldsymbol{x}_k\|_Q^2 + \|\boldsymbol{u}_k^{\text{ref}} - \boldsymbol{u}_k\|_R^2 \tag{2.20}$$

where Q and R are positive, semi-definite weighting matrices, defining the penalties for the errors, and the notation used is $\|\boldsymbol{x}\|_A^2 = \boldsymbol{x}^T A \boldsymbol{x}$, where A is a positive, semi-definite weighting matrix. Furthermore, $\boldsymbol{x}_k^{\text{ref}}$ is the state reference while $\boldsymbol{u}_k^{\text{ref}}$ is the control signal reference which are desired to be followed.

The cost function is used in the MPC problem formulation to achieve a desirable behaviour, while the prediction model is used to plan ahead using future states to ensure that the desired behaviour is realised. This further motivates the use of the methods in Section 2.2, aimed at improving the prediction model based on observed data, in order to further enhance the MPC controller.

2.4 Motion planning

In order to steer the system from a starting position to a desired position, a feasible path between the two states, with respect to the surroundings of the Rover and the dynamical model of the system, must be generated. This is done by a *motion planner*, computing the generated path, which is then used as a reference by the controller.

2.4.1 Problem formulation

The motion planning problem can be formulated as: given an input consisting of an initial vehicle state, $\boldsymbol{x}_s$, a goal state $\boldsymbol{x}_g$, and information about the surroundings, compute a sequence of vehicle states $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_D$ and a control sequence, $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{D-1}$, such that $\boldsymbol{x}_0 = \boldsymbol{x}_s$, $\boldsymbol{x}_D = \boldsymbol{x}_g$ and $\boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{\beta}_i)$ for all $i = 0, \ldots, D-1$. Here, $f(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{\beta}_i)$ is the function describing the dynamical model of the system, i.e. the prediction model in (2.18). The sequence of states and control signals must be feasible with respect to the vehicle's kinematic constraints,

constraints on the actuators, and obstacles that must be avoided by the vehicle. In general, it is also of interest to compute a path where some performance measure, such as the distance travelled, is minimised [7].

2.4.2 Solving the motion planning problem

A commonly used approach to solve these types of motion planning problems is to apply sampling-based motion planners [16]. These types of planning algorithms construct a directed graph with vertices and edges while simultaneously employing graph-searching algorithms in order to find an optimal solution to the motion planning problem. One method making use of this technique is the lattice-based motion planner. In a lattice-based motion planner, the control inputs are reduced to a discrete set [24], and a sequence of such inputs will cause the system to move, generating a sequence of states. The state and control signal sequence is often referred to as a *motion primitive*, which is defined as: $\boldsymbol{m} = \{(\boldsymbol{x}_k, \boldsymbol{u}_k)\}_{k=0}^{L-1} \in \mathcal{X} \times \mathcal{U}$, where L is the pre-defined length of the state and control sequence, $\mathcal{X}$ is the set of feasible vehicle states and $\mathcal{U}$ is the set of feasible control signals.

Generating the motion primitives

The motion primitives are sequences of control signals and states that are optimised for some performance measure and are used to build a graph, where vertices represent states of the system – such as its position and heading – while the edges correspond to feasible motions, that transition the system between states. The sequence of control inputs and states that results in a motion primitive is generated by solving a discrete optimal control problem [3]. The OCP has the following form:

$$\begin{aligned}
&\underset{\{\boldsymbol{x}_k\}_{k=0}^{L-1}, \{\boldsymbol{u}_k\}_{k=0}^{L-1}}{\text{minimise}} && J_L && \text{(2.21a)}\\
&\text{subject to} && \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\beta}_k), k = 0, \dots, L-1, && \text{(2.21b)}\\
& && \boldsymbol{x}_0 = \boldsymbol{x}_i, \quad \boldsymbol{x}_L = \boldsymbol{x}_f, && \text{(2.21c)}\\
& && \boldsymbol{x}_k \in \mathcal{X}, k = 1, \dots, L-1, && \text{(2.21d)}\\
& && \boldsymbol{u}_k \in \mathcal{U}, k = 0, \dots, L-1 && \text{(2.21e)}
\end{aligned}$$

where $\boldsymbol{x}_k$ is the current state of the vehicle and $\boldsymbol{u}_k$ is the current control signal. J_L is the cost function, specifying the performance measure for optimality of the motion primitives. An example of a general cost function can be seen in (2.20). The solution to (2.21) is a motion primitive containing a control and state sequence that – given a cost function – is optimal for bringing the system from $\boldsymbol{x}_i$ to $\boldsymbol{x}_f$. This cost function could for example be chosen in order to maximise the velocity or to keep the control signals as small as possible. Equation (2.21b) ensures that the optimiser uses the correct dynamical model of the system in order to forward simulate it.

When generating the motion primitives, the combination of initial and final states is selected in a finite number of ways. The motion primitives that move the system from the initial state to the final state are then computed. The requirements on $\boldsymbol{x}_f$ vary, for example could only the final heading of the system be specified, or the final velocity, e.g. in the case of a *stop primitive* – taking the system to a state with zero velocity. See more in Section 4.3.

For a position invariant system, it is only necessary to compute motion primitives with an initial state in the origin (i.e. the position variables are 0). These motion primitives can then be translated to all other positions in the search-space and reused. In addition, if the system is also rotational invariant, it is possible to exploit the rotational symmetries of the system. This means that it suffices to compute motion primitives from a few (or a single) initial headings, and then rotate the motion primitives to all other initial headings [3].

In conclusion, a motion primitive is generated by following the steps below:

1. Choose the combination of states, $\boldsymbol{x}_i$ and $\boldsymbol{x}_f$, that should be connected.
2. Solve the OCP in (2.21), thus generating an optimal control and state sequence, i.e. the motion primitive.

Planning

When applying the control sequence from a single motion primitive to the system, it will reach a new state. The goal is to apply motion primitives in an order such that the system moves from its initial state to the final state, while avoiding obstacles. In order to generate this sequence of motion primitives, knowledge about the surroundings of the system is required. This knowledge can for example be represented as an *occupancy grid*, which discretises the environment into an evenly-space field of grid elements, each of which indicates if an obstacle is present in that grid element or not [27]. The problem of finding the optimal order of motion primitives can be defined as an OCP:

$$\begin{aligned}
\underset{\{\boldsymbol{m}_k\}_{k=0}^{K-1}, K}{\text{minimise}} \quad & J_K = \sum_{k=0}^{K-1} L_{\boldsymbol{m}}(\boldsymbol{m}_k) && \text{(2.22a)}\\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{m}_k, \boldsymbol{\beta}_k), k = 0, \dots, K-1, && \text{(2.22b)}\\
& \boldsymbol{x}_0 = \boldsymbol{x}_s, \boldsymbol{x}_K = \boldsymbol{x}_g, && \text{(2.22c)}\\
& \boldsymbol{m}_k \in \mathcal{P}(\boldsymbol{x}_k), k = 0, \dots, K-1, && \text{(2.22d)}\\
& g(\boldsymbol{x}_k, \boldsymbol{m}_k, \boldsymbol{\beta}) \notin \mathcal{X}_{\text{obst}}, k = 0, \dots, K-1 && \text{(2.22e)}
\end{aligned}$$

where $\boldsymbol{m}_k$ is the motion primitive applied in timestep k, $L_m(\boldsymbol{m}_k)$ is the *stage cost* of a motion primitive, while $f(\boldsymbol{x}_k, \boldsymbol{m}_k, \boldsymbol{\beta}_k)$ returns the resulting state when the control sequence in a motion primitive is applied from the current state. $\mathcal{P}(\boldsymbol{x}_k)$ is the set of motion primitives that are valid from the current state, $\boldsymbol{x}_k$. The function $g(\boldsymbol{x}_k, \boldsymbol{m}_k, \boldsymbol{\beta}_k)$ returns the sequence of resulting states, $\{x\}_{k=0}^{L-1}$, when applying the control sequence in a motion primitive from the current state. Since the path needs to avoid any obstacles, a collision check is done in (2.22e), making sure that no infeasible paths are generated, with states present in $\mathcal{X}_{\text{obst}}$, which is the set containing all the non-admissible states. Finally, J_K is the cost function. If the model parameters, $\boldsymbol{\beta}$, used in both (2.21) and (2.22), are unknown, they can be estimated using the methods in section 2.2. Solving the OCP in (2.22) results in a sequence of motion primitives of length K.

Graph traversal

An optimal way of moving the system between two states that are not directly connected in the graph, i.e. the solution to (2.22), can be found by employing graph-searching algorithms. There is no shortage of algorithms capable of handling this task [31]. One of the more popular, called A^*, was first developed by three Americans in the 1960's as a way to, e.g., find the shortest path between two cities, rendering it a good option for solving the motion planning problem [10].

In order to be able to solve the lattice based motion planning problem using graph-search methods, the search-space is discretised. This is generally done using a grid with a pre-defined fidelity for the A* algorithm [4]. The algorithm also needs some measurement when deciding optimality. In this case, the A* algorithm uses an estimate of the objective function value, called J_{est}. This estimate is calculated as: $J_{est} = J_{come} + h(\boldsymbol{x}_k)$, where J_{come} is the cost to reach the current state from the initial state, often called the *cost-to-come*. This cost-to-come is the sum of the stage costs of the motion primitives applied from the initial state to the current state. See more in Section 4.3.3. The function $h(\boldsymbol{x}_k)$ – sometimes denoted $h(\boldsymbol{x}_k, \boldsymbol{x}_g)$ – is a *heuristic* used to estimate the cost to go from the current state to the final state, and is often referred to as the *cost-to-go*. If the algorithm uses the distance of the motion primitive as the stage cost, an example of an, albeit very simple, heuristic is the Euclidean distance between the current and the final node. Two important conditions to consider when choosing an heuristic function, $h(\boldsymbol{x}_k)$, is to make sure it has the two following properties [14]:

- Admissibility: $h(\boldsymbol{x}_k) \leq h^*(\boldsymbol{x}_k)$,
- Consistency: $h(\boldsymbol{x}_k) \leq k(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) + h(\boldsymbol{x}_{k+1})$,

where $h^*(\boldsymbol{x}_k)$ is the *optimal* cost-to-go from the current state, $\boldsymbol{x}_k$, to the final state, $\boldsymbol{x}_g$. The function $k(\boldsymbol{x}_k, \boldsymbol{x}_{k+1})$ is used to describe the cost of the edge going from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$, using the specified cost function. The condition on the heuristic to be admissible means that it must never over-estimate the cost-to-go, while the condition on the heuristic to be consistent means that the cost-to-go from one state must be smaller than or equal to the sum of the cost-to-go from the state of its child vertex and the edge cost of travelling between the two states. If both of these conditions are met, the optimality guarantees are maintained [3].

The A* algorithm returns a sequence of vertices and edges, containing states and control signals respectively, that is used to form a path from the initial state to the final state.

An extension to the A* algorithm is the so-called *hybrid* A* algorithm. The difference between these two algorithms is the discretisation of the search-space, more specifically how the states relate to each other in the resulting grid elements. The standard A* algorithm only allows states to end up in the middle of the grid element, while the hybrid A* allows states to end up at any continuous point within a grid element. For a graphic clarification, see Figure 2.2.

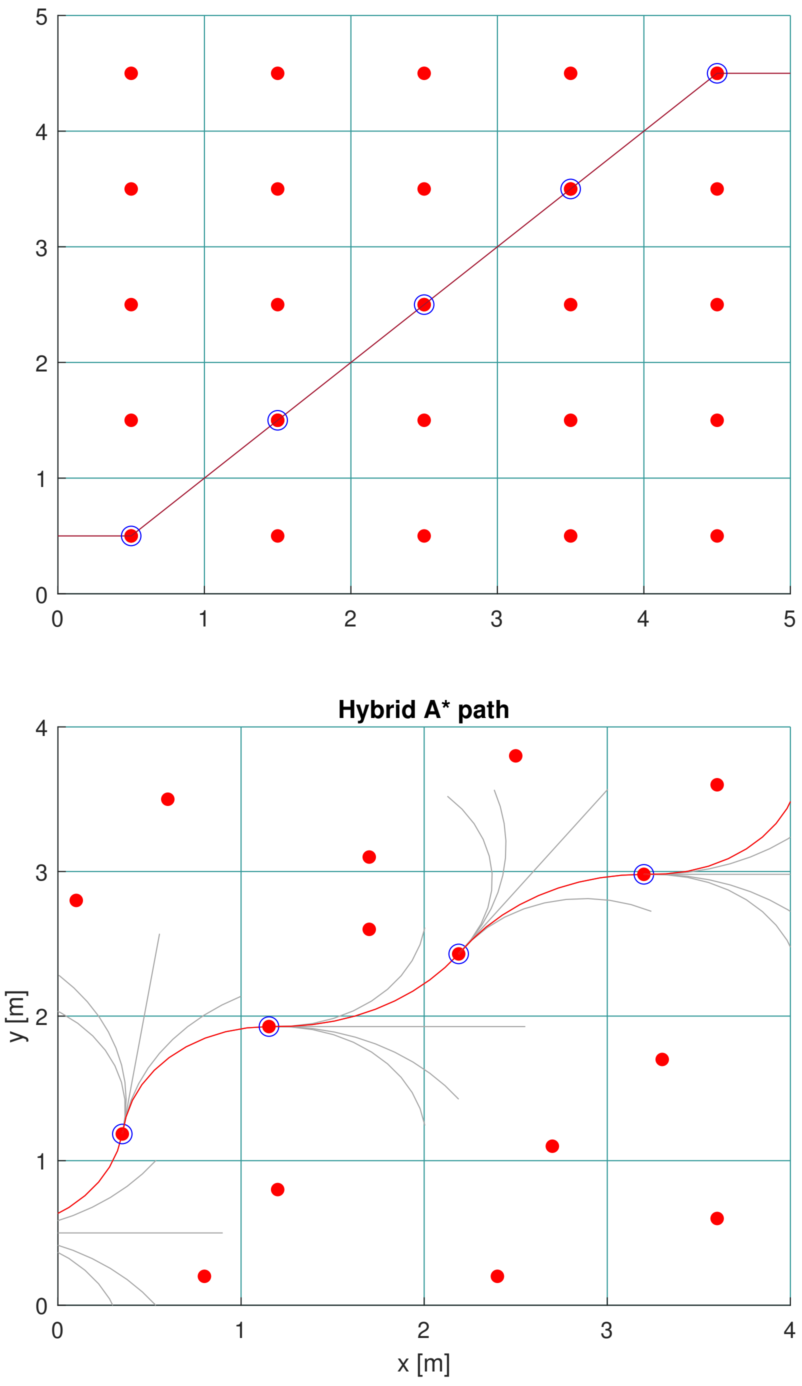

Figure 2.2: *The discretisation of the search space for regular A* versus hybrid A*. Top: The A* algorithm only ends up in the centre of the grid elements. Bottom: The hybrid A* algorithm can end up anywhere within a grid element.*

3 System description

This chapter gives a description of how the Rover is modelled in this thesis and the simulators used to evaluate the performance of the system used to control the Rover. Lastly, an overview of this system is provided.

3.1 Dynamical model of the Rover

The dynamical model of the Rover used in this thesis is based on a differential drive model. The model consists of two parts, where the first one represents the dynamical model of the position and heading of the Rover [18], defined as:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{3.1}$$

where x and y represent the 2-dimensional position of the Rover, θ is its heading, v is the absolute velocity and ω the angular velocity of the Rover.

The second part of the model instead represents the dynamical model of the absolute and angular velocities of the Rover [18], i.e. the actuator dynamics. It is defined as:

$$\begin{aligned} \dot{v} &= v^{\text{cmd}} w_1^v + v w_2^v + \eta^v \\ \dot{\omega} &= \omega^{\text{cmd}} w_1^\omega + \omega w_2^\omega + \eta^\omega \end{aligned} \tag{3.2}$$

where v^{cmd} is the commanded absolute velocity and ω^{cmd} is the commanded angular velocity, sent to the actuators of the Rover. Hence, the combination of

(3.1) and (3.2) can be discretised and compactly written as $f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\beta})$, where $\boldsymbol{x}$ is the state vector defined as $\boldsymbol{x} = [x, y, \theta, v, \omega]^T$, $\boldsymbol{u}$ is the control signal defined as $\boldsymbol{u} = [v^{\text{cmd}}, \omega^{\text{cmd}}]^T$, representing the commanded speed and turn rate. Finally, the model parameters are defined as $\boldsymbol{\beta} = [w_1^v, w_2^v, w_1^\omega, w_2^\omega]^T$. As mentioned in Section 2.1, these model parameters are in this model of the Rover unknown, and in some cases also time-varying. The variables η^v and η^ω represent the model uncertainties, and are defined as zero-mean normally distributed random variables with variances σ_v^2 and σ_ω^2, respectively. Since the model parameters are unknown, (3.2) can be seen as a Normal linear regression problem, defined in (2.2). The model parameters, or regression parameters, can therefore be estimated using the methods discussed in Section 2.2.

The model is used both in the proposed motion planning and control framework, described in Chapter 4, but also for simulating the system. This is further described in the next section.

3.2 Simulation

In order to facilitate the development of the framework as well as the performance evaluation of the system, simulations are used instead of hardware. Two different types of simulators using different models of the Rover are used in this thesis, which are described in this section.

3.2.1 Known model simulation

The first type of simulator is implemented in Python, built upon the known model of the Rover specified in (3.1) and (3.2). These equations are used together with a forward Euler method [25] in order to simulate the motion of the Rover. The *true* model parameters that are used within this simulator are user-defined and modifiable in order to easier verify, e.g., the parameter estimation under ideal conditions using the BLR algorithm described in Section 2.2. The model uncertainties, η_v and η_ω described in Section 3.1, have the user-defined variances σ_v^2 and σ_ω^2, respectively. Furthermore, no time-critical aspect has to be considered since the simulator waits until receiving the control signals in each iteration of the simulation.

3.2.2 Unknown model simulation

The second type of simulator is built upon an unknown model of the Rover in the simulation program *Gazebo* [23]. This unknown model is defined by a Gazebo plug-in called `gazebo_ros_diff_drive` used to simulate a differentially steered vehicle. As this model is unknown, it is a good intermediate step between an exact simulation, in which the model and the true model parameters are known, and an actual physical Rover (hardware), for which the true model is unknown and has to be approximated in the motion planner and in the MPC controller. A visualisation of the simulation in Gazebo can be seen in Figure 3.1.

Beyond the fact that this simulator is based upon an unknown model, there is also a time-critical aspect to take into account. This is because the simulation of the Rover is run separately from the control algorithm, using Robot Operating System (ROS) communication to send control signals to the Rover, as it would be done in a physical implementation. Overall communication specified in Figure 3.2 is handled using ROS.

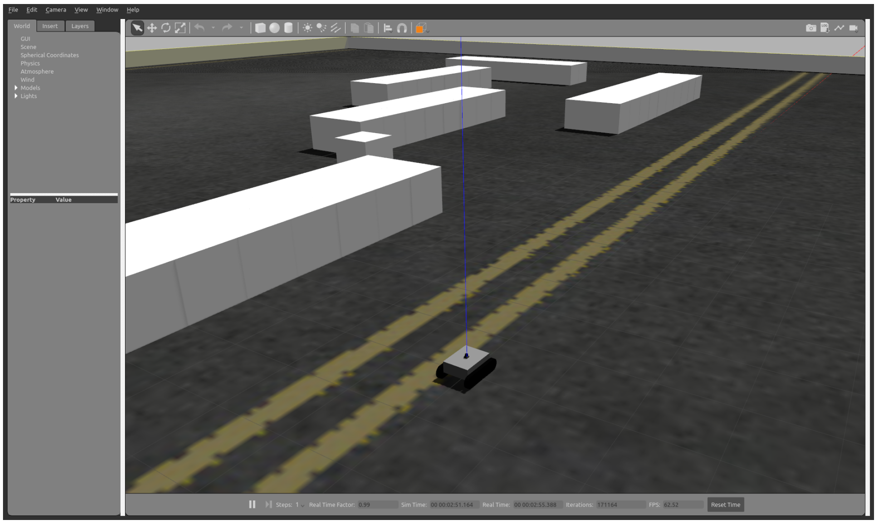

Figure 3.1: *The simulation environment in Gazebo. The model of the Rover is seen in the test environment that contains obstacles.*

3.3 System overview

The simulators need to be able to communicate, i.e. to send and receive data, with the other components of the system, called *subsystems*. This section gives an overview of the subsystems, and how they cooperate and interact with one another.

The complete data flow during a run of the system is shown in Figure 3.2. The input to the system is a desired goal state, that can either be user-defined or computed by a higher-level planner, as well as information about the surrounding environment, in the form of an occupancy grid. The goal state and the occupancy grid are sent to the motion planner, which generates motion primitives before computing a feasible path to the goal state, as seen in Section 2.4. The path consists of a sequence of states and control signals and is sent to the MPC controller that, at each time step, computes an optimal control signal, as described in Section 2.3, that is sent to the actuators of the Rover.

The applied control signals will cause the rover to move towards the goal state. During the movement, the parameter estimation subsystem collects the relevant data and uses it to compute new estimates of the model parameters,

β, as described in Section 2.2. These estimates are sent to the motion planner, resulting in the next plan from the planner being improved, as well as to the MPC controller, leading to an improved path following for the Rover. More detailed descriptions of these subsystems are seen in Chapter 4.

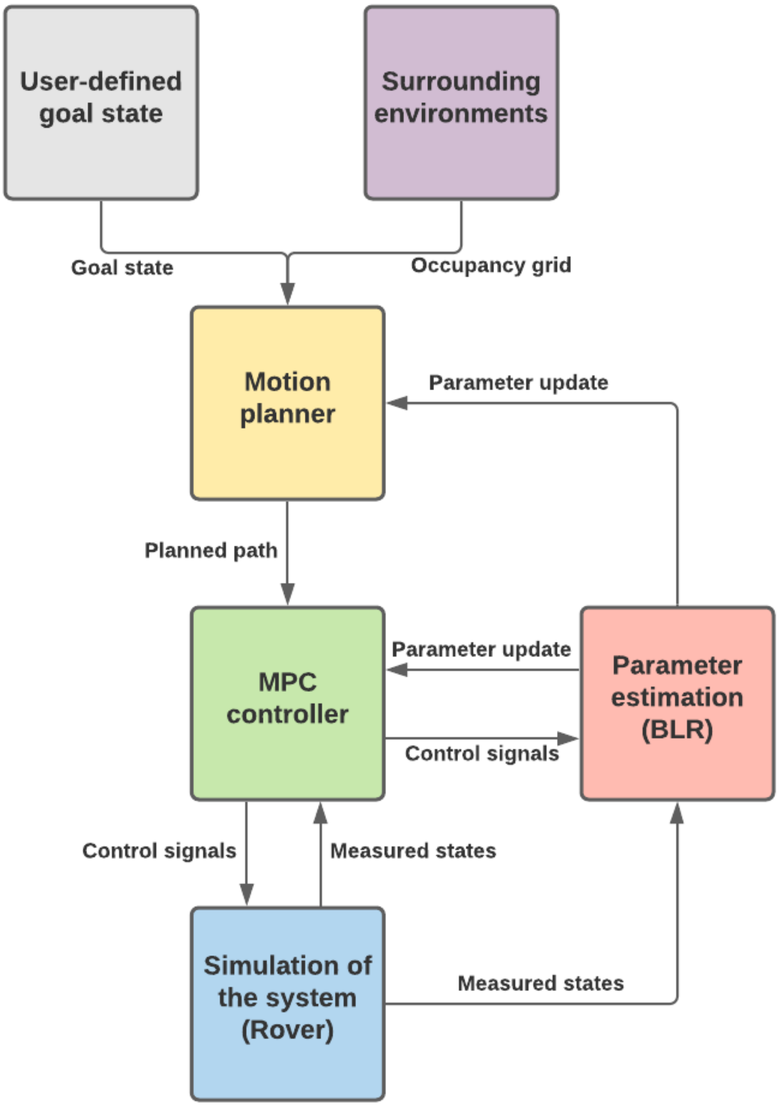

Figure 3.2: *Flow chart of the communication between subsystems during a run.*

4 Learning-based control and planning

This chapter presents the implementation details for the learning-based motion planning and control framework developed in this thesis. It consists of a subsystem that estimates the model parameters, as well as an MPC controller and a motion planner.

4.1 Bayesian linear regression

This section covers how the BLR algorithm that is used to estimate the model parameters is implemented. When exploiting data from the system in order to identify and improve the model parameters, the BLR algorithm is applied to the dynamical model of the Rover in (3.2). The goal is to identify the four unknown parameters, $w_1^{\omega}, w_2^{\omega}, w_1^{v}$ and w_2^{v} called *regression parameters*, which are denoted $\boldsymbol{\beta}$ in Section 2.2, starting from the prior and then continuously update the parameter estimates based on new data. The BLR algorithm computes the posterior distributions of these parameters, together with the distribution of the model uncertainty parameters, σ_v^2 and σ_ω^2 from (2.13). The update from prior to posterior when new data is received is done using the update scheme given in (2.15).

4.1.1 The importance of the prior

When using BLR to estimate unknown model parameters, the initial guess on the parameter distributions – i.e. the prior – is highly important. For example, two BLRs over the same set of data using two numerically different priors can yield two numerically different results [20], which can be observed when using two numerically different priors in the update scheme in (2.15).

The initial prior guess used in the BLR algorithm is NIG distributed and consists of multiple variables:

- The expected values of the model parameters, $\boldsymbol{\beta}_0^{\omega}$ and $\boldsymbol{\beta}_0^{v}$.
- The covariance matrices, $\boldsymbol{V}_0^{\omega}$ and $\boldsymbol{V}_0^{v}$.
- The inverse gamma parameters, a_0^{ω}, b_0^{ω}, a_0^{v} and b_0^{v}.

This selection of prior guesses is also used to initialise the model parameters in the MPC controller and the motion planner. The prior guess can be either inaccurate or accurate with respect to the true model parameters used by the simulators. In the case of unknown model simulation, discussed in Section 3.2.2, the model used in BLR is most likely not the same as the actual model used by the Gazebo plug-in. Hence, the term *accurate model parameters* denotes the parameters that best fit the data, estimated using the BLR algorithm, for the unknown model simulation. The impact of using accurate or inaccurate priors is covered in Chapter 5.

In order to obtain the best possible prior guess for an unknown model simulation, a number of initial identification simulations were conducted. These simulations were performed with the purpose of exciting the system as intelligently as possible, meaning that the reference path for the Rover had turns in both positive and negative directions in order to give the system a varying angular velocity. During the first experiment, the prior was chosen based on intuition of the system, but during the subsequent experiments, the last posterior from the previous run was used as the initial prior in the current run. This was repeated until convergence, i.e. until the difference between the prior and the posterior was smaller than a reasonable threshold. The last obtained posterior was then used as prior for the BLR algorithm, as well as the initial model parameter guess in the MPC controller and the motion planner.

The values assigned to the prior expected values of the model parameters, $\boldsymbol{\beta}_0$ vary depending of the type of prior used. If an accurate prior is used, the expected values are sufficiently close to the true model parameters, found during the identification simulations. When instead an inaccurate prior is used, $\boldsymbol{\beta}_0^{v}$ and $\boldsymbol{\beta}_0^{\omega}$ are assigned values that are not close to the true model parameters. The values of the prior covariance matrices and the values of the prior IG parameters, used to generate the results in Chapter 5, are always the same, defined as:

$$\begin{aligned}
\boldsymbol{V}_0^{\omega} &= \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \\
\boldsymbol{V}_0^{v} &= \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \\
a_0^{\omega} &= 3.1 \\
b_0^{\omega} &= 1.5 \\
a_0^{v} &= 2.1 \\
b_0^{v} &= 0.5
\end{aligned} \tag{4.1}$$

4.1.2 Obtaining the data

In order to update a prior into a posterior, the BLR algorithm needs to obtain a set of data points. One data point consists of three numerical values, two explanatory variables and one response variable, originating from one of the dynamical equations in (3.2). The input to the BLR algorithm is one or multiple data points, while its output is the estimates of the model parameter distributions.

When estimating the model parameters for the dynamical model of the absolute velocity of the Rover, i.e. w_1^v, w_2^v and σ_v^2, the variables of interest are: $\dot{v}$, v^{cmd} and v, while the variables $\dot{\omega}$, ω^{cmd} and ω are of interest when estimating the model parameters for the dynamical model of the angular velocity of the Rover, i.e. w_1^ω, w_2^ω and σ_ω^2. The state variables v and ω are obtained from the simulation of the system, or from a state estimator (such as an extended Kalman filter) in case of a physical Rover. The command variables v^{cmd} and ω^{cmd} are obtained from the output of the MPC controller. The last two variables, $\dot{v}$ and $\dot{\omega}$, are not given explicitly in the case of an unknown model simulation, which means that they need to be estimated. This is achieved using the Euler forward method.

Once the relevant variables for either the absolute velocity or the angular velocity are computed, they are sent as input to the BLR algorithm, converting the prior into a posterior that contains the updated information about the distribution of the regression parameters.

4.1.3 Sending the parameter estimates

When the probability distributions of the regression parameters have been computed, the BLR algorithm performs a check to determine if the expected value of each parameter should be transmitted to the MPC controller and the motion planner. Deciding upon whether or not to send the parameter estimates, the BLR algorithm needs to balance the risk of sending inaccurate parameter estimates versus making the other subsystems use older parameter estimates, that may be outdated. Since outlier data points could affect the estimates, the new parameter estimates after adding the outlier point to the regression might have changed too much, resulting in worse performance from the subsystems receiving the estimates. On the other hand, it is not desirable that those subsystems use older parameter estimates that might become more and more inaccurate.

Managing this task involves computing how much the model uncertainty and the expected values of the parameter estimates have changed. If the expected values have changed more than a pre-defined percentage, there is a risk that outlier data have effected the estimates too much. If that was to happen, the uncertainty in the system would also increase. Therefore, in order to send the parameters, the expected values of the parameter estimates are not allowed to change more than q percent from their respective values that were last sent, while the same is true for the system uncertainty. However, it is possible that the parameters do change quickly even in the absence of outliers. Therefore, the BLR algorithm will send the parameter estimates regardless if the parameters have changed more than q percent or not, when n_{iter} algorithm iterations have passed since the last time the parameter estimates were sent. The parameters q and n_{iter} allow the receiving algorithms to use recent parameter estimates, while also reducing the risk of sending inaccurate model parameter estimates, caused by outlier data. The parameter values used in this thesis are: $q = 20$ % and $n_{\text{iter}} = 10$. The same concept is also used in the WBLR algorithm.

4.2 Model predictive controller

This section describes in detail how the MPC controller is implemented.

The formulation for the MPC controller used to generate control sequences to the system is defined as:

$$\begin{aligned}
\underset{\{\boldsymbol{u}_k\}_{k=0}^{N-1}}{\text{minimise}} \quad & J_N \\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\beta}_k), k = 0, \dots, N-1 \\
& \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \\
& v_k^{\text{cmd}} \in [v_{\min}^{\text{cmd}}, v_{\max}^{\text{cmd}}], k = 0, \dots, N-1 \\
& \omega_k^{\text{cmd}} \in [\omega_{\min}^{\text{cmd}}, \omega_{\max}^{\text{cmd}}], k = 0, \dots, N-1
\end{aligned} \tag{4.2}$$

where the dynamical model $f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\beta}_k)$, states $\boldsymbol{x}$, control inputs $\boldsymbol{u}$ and model parameters $\boldsymbol{\beta}$ are defined as in Section 3.1, and $f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\beta}_k)$ is discretised using Runge-Kutta approximation of the fourth order (RK4) [32] with timestep length Δt. Furthermore, $(\,\cdot\,)_{\min}^{\text{cmd}}$ and $(\,\cdot\,)_{\max}^{\text{cmd}}$ are the lower and upper bounds respectively for the control signals, $(\,\cdot\,)_k^{\text{cmd}}$ in (3.2) and $\boldsymbol{x}_{\text{init}}$ is the measured current state of the Rover. The cost function, $J_N(k)$, used in the MPC is defined as:

$$\begin{aligned}
J_N = \sum_{k=0}^{N} & \|\theta_{\text{ref},k} - \theta_k\|_{\gamma_\theta} + \|x_{\text{ref},k} - x_k\|_{\gamma_x} + \|y_{\text{ref},k} - y_k\|_{\gamma_y} + \\
& \|\dot{\omega}_k^{\text{cmd}}\|_{\gamma_{\dot{\omega}}^{\text{cmd}}} + \|\dot{v}_k^{\text{cmd}}\|_{\gamma_{\dot{v}}^{\text{cmd}}} - \|v_k\|_{\gamma_v}
\end{aligned} \tag{4.3}$$

with the same notation as in (2.20) but where the notations $\dot{v}_k^{\text{cmd}}$ and $\dot{\omega}_k^{\text{cmd}}$ are the 1-step difference for each of the variables e.g.:

$$\dot{v}_k^{\text{cmd}} = \frac{v_k^{\text{cmd}} - v_{k-1}^{\text{cmd}}}{\Delta t}. \tag{4.4}$$

This measure is used to make the control signals *smoother*. In addition, $(\,\cdot\,)_{\text{ref}}$ is the reference for the states $\boldsymbol{x}$ and control inputs $\boldsymbol{u}$, computed by the motion planner. The parameters $\gamma_{(.)}$ are tuned to achieve the desired system behaviour, and can be seen in Table 4.1, together with other MPC specific parameters. The OCP in (4.2) is computed using the interior point optimiser (IPOPT) solver [29] in the optimisation software CasADi [1].

Parameter values for the MPC controller		
Parameter	*Known model*	*Unknown model*
γ_θ	15	40
γ_x	20	35
γ_y	20	35
$\gamma_{\dot\omega}^{\text{cmd}}$	0.5	3
$\gamma_{\dot v}^{\text{cmd}}$	0.5	5
γ_v	15	5
$v_{\text{min}}^{\text{cmd}}$	0	0
$v_{\text{max}}^{\text{cmd}}$	2.1	2.1
$\omega_{\text{min}}^{\text{cmd}}$	-2	-2
$\omega_{\text{max}}^{\text{cmd}}$	2	2
v_{max}	2.1	2.1
N	100	35
Δt	0.1	0.2

Table 4.1: *Parameter values used in the MPC controller for the two simulators, using a known and an unknown model, respectively.*

4.3 Motion planner

In this section, the details regarding how the motion planner is implemented are presented. This includes a description of the state space discretisation, how the motion primitives are computed and how the graph search is performed.

4.3.1 State space discretisation

The hybrid A* algorithm, discussed in Section 2.4.2, needs to be able to check if states are considered as equal. This is done by assigning a state-dependent ID to each state. By discretising the state space into a 5D-grid, one dimension for each state variable, x, y, θ, v and ω, with a pre-defined fidelity, a state is assigned an ID depending on the value of its state variables. Two states containing state variables that belong to the same grid element (with respect to the fidelity, i.e. contained in the grid element) will have identical IDs, and therefore considered as equal by the hybrid A* algorithm. The fidelity used is 0.8 m for x and y, $\frac{\pi}{6}$ rad for θ, 0.5 m/s for v and finally 0.5 rad/s for ω.

4.3.2 Motion primitives generation

As mentioned in Section 2.4.2, each motion primitive is generated by solving the OCP defined in (2.21), using CasADi [1]. The output is an optimal state and control signal sequence that connects two different Rover states.

The states that are to be connected to the origin mainly consists of constraints on the heading, but also on the velocity and the angular velocity. The constraints on the heading specifies that the Rover must initially have heading $\theta_0 = 0$ as well as final heading $\theta_f \in \{\frac{\pi}{2}, \frac{\pi}{4}, 0, -\frac{\pi}{4}, -\frac{\pi}{2}\}$. The initial and final angular velocity, ω_0 and ω_f, are always constrained to be 0, so that the Rover does not have any initial turning speed when creating the next vertex in the planning process. In general, one OCP is solved for each constraint on θ_f. There is also a constraint on the final velocity, $v_f = v_{\max}$, causing the Rover to always hold the maximum velocity at the end of a motion primitive. In order to reduce the complexity of the motion planning problem, no motion primitive taking the Rover to a halt is computed, as this is instead handled by the MPC controller. The other state variables, x and y, are left as free variables to be optimised. Finally, there are constraints on the two control signals, v^{cmd} and ω^{cmd}, which are $(\,\cdot\,)^{\text{cmd}}_{\min}$ and $(\,\cdot\,)^{\text{cmd}}_{\max}$, the same that are used in (4.2).

A special case is when $\theta_f = 0$, for which two OCPs need to be solved, one for the initial condition on the Rover's velocity, $v_0 = 0$, and one for $v_0 = v_{\max}$. The reason is that the Rover always starts a run with zero velocity, and a motion primitive instead constructed to start with the velocity $v_0 = v_{\max}$ will cause the forward simulation to be less accurate.

In order to make the planning more versatile, so-called *parallel movement primitives* – moving the Rover a certain length orthogonal to its initial heading before ending up with the same initial heading – are added by solving four OCPs. The constraints used are $\theta_0 = \theta_f = 0$, $v_0 = v_f = v_{\max}$, $\omega_0 = \omega_f = 0$ and a specific constraint on the final position along the y-axis, $y_f = d_{\text{lat}}$, meaning that the state variable y is no longer a free variable. The constant d_{lat} is a parameter specifying the distance of the parallel movement, with $d_{\text{lat}} \in \{-1, -0.5, 0.5, 1\}$.

When simulating the Rover using (3.1) and (3.2) in the OCP solver, i.e. (2.21b), the Euler discretisation forward method is used. This method is numerically less accurate than e.g. a RK4. However, when comparing computational speed with the numerical variation for the two methods, it was decided that the Euler method is the better option.

The cost function used in the OCP minimises the difference in the angular velocity control signals, i.e. $\frac{\omega_k^{\text{cmd}} - \omega_{k-1}^{\text{cmd}}}{\Delta t}$, and also maximises the velocity of the Rover. It is defined as:

$$J_L = \sum_{k=0}^{L} \|\dot{\omega}_k^{\text{cmd}}\|_{\gamma_{\dot{\omega}}^{\text{cmd}}} - \|v_k\|_{\gamma_v} \tag{4.5}$$

with the same notation as in (2.20) and (4.4). This cost function ensures that a high velocity is maintained, while also generating a smooth path for the Rover to follow. The generated motion primitives can be seen in Figure 4.1.

4.3.3 Computing the path

Once the motion primitives are constructed, the motion planner uses them in order to solve the motion planning problem in (2.22), generating a path. The graph is simultaneously constructed and searched, where the motion primitives are edges that connect vertices in the graph. The vertices, n, contain a state with an ID, a parent vertex, a cost-to-come, an estimate of the final cost and the motion

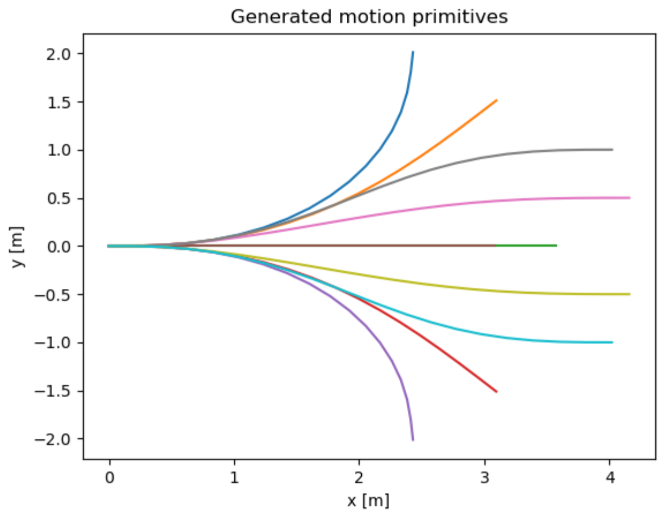

Figure 4.1: *A graphical representation of the generated motion primitives. Each colour represents a motion primitive.*

primitive that was used to move the Rover from the state in the parent vertex to the state in the current vertex, called its *motion*. The method to assign an ID to a state is described in Section 4.3.1. The cost-to-come, J_{come}, of a vertex is the stage cost of the motion associated with the vertex, added to the cost-to-come of its parent vertex. The estimated final cost of the vertex, J_{est}, is calculated as the sum of the cost-to-go and the cost-to-come of the same vertex.

When a vertex is expanded in the hybrid A* algorithm, each valid motion primitive is simulated by applying its corresponding control signal sequence. While computing the resulting state path, the algorithm simultaneously checks for collisions with obstacles. The collision check is conducted by checking if the state corresponding to the vertex and an obstacle both are contained within a grid element of a pre-defined occupancy grid, which is described in Section 2.4.2. The quantity that the hybrid A* algorithm uses to determine optimality is chosen as the length of the path. For a motion primitive, its stage cost is defined as its resulting path length, while the heuristic is defined as the Euclidean distance between the current state and the goal state.

The graph-search algorithm used in this thesis as the base for the hybrid A* algorithm, is inspired by the A* algorithm tailored for lattice-based motion planning used in [3]. The input to the hybrid A* algorithm is the initial state, $\boldsymbol{x}_s$, the goal state, $\boldsymbol{x}_g$, and the occupancy grid containing the obstacles, $\mathcal{X}_{obst}$, while the output is a sequence of states and control signals that forms a feasible path from $\boldsymbol{x}_s$ to $\boldsymbol{x}_g$. In order for this path to be feasible with respect to the Rover, the model parameter estimates, $\boldsymbol{\beta}$, are sent from the BLR algorithm. The hybrid A* algorithm, inspired by [3], is seen in Algorithm 1.

There are two sets used in the hybrid A* algorithm, the closed set, $\mathcal{V}_{closed}$, to keep track of the visited vertices and the open set,$\mathcal{V}_{open}$, to gather the vertices left for exploration. The vertices in the open set are sorted according to J_{est}, meaning that the vertex containing the most promising state is always selected for expansion [3].

Since this thesis does not focus on resolution optimal paths, the hybrid A* algorithm is timed out after a specified time period. If a feasible, sub-optimal path has been generated during this time, but it is not yet guaranteed to be resolution optimal, this path is used when the algorithm times out.

4.3.4 Making the plan dynamically feasible

Since some of the system parameters are unknown and estimated online using BLR, their estimates will most likely change during the course of the run – unless the initial guess is sufficiently close to the true values of the model parameters. In addition, the dynamical model of the Rover might change during runs if for example something on the Rover breaks. This means that executing the plans based on the initial parameter estimates may not lead to a feasible path. To counter this problem, it is possible to use one of three different approaches. One of the approaches is that the function, $f(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\beta})$, that calculates the next state when generating the planned path is updated with the new model parameters. The second approach is to recalculate the OCPs and optimise the motion primitives with respect to the new model parameters to re-plan the path. And thirdly, it is possible to combine the two approaches. The third option is chosen, since the hybrid A* algorithm was shown to have problems finding a feasible path when using a different set of model parameters in (2.22b) compared to the function computing the motion primitives in (2.21b).

The BLR algorithm is responsible for sending the latest parameter estimates to the MPC controller and the motion planner, so the motion planner only needs to use the parameter estimates it currently is assigned. For more information about sending the parameters, see Section 4.1.3.

4.3.5 Complete plan

Once the plan that consists of a sequence of states, $\boldsymbol{x}_s, \boldsymbol{x}_1, ..., \boldsymbol{x}_g$, and a sequence of control signals, $\boldsymbol{u}_0, \boldsymbol{u}_1, \dots, \boldsymbol{u}_{D-1}$, has been generated, it is sent as a reference to the MPC controller. The MPC controller will then repeatedly compute control inputs to the system, which ensure that the reference path is followed to the best of its ability.

Algorithm 1: The hybrid A* graph-search algorithm

```
Initialisation: $n_s.\boldsymbol{x} \leftarrow \boldsymbol{x}_s, \boldsymbol{x}_s.id \leftarrow id(\boldsymbol{x}_s),$
                $\boldsymbol{x}_g.id \leftarrow id(\boldsymbol{x}_g), \nu_{open} \leftarrow n_s, \nu_{closed} \leftarrow \emptyset$
while $\nu_{open} \neq \emptyset$ do
    $n_k \leftarrow \nu_{open}.pop()$
    $\nu_{closed}.push(n_k)$
    if $n_k.\boldsymbol{x}.id = \boldsymbol{x}_g.id$ then
        return back_track($n_k$)
    end
    for each $\boldsymbol{m} \in \mathcal{P}(n_k.\boldsymbol{x})$ do
        $\boldsymbol{x}_{next} \leftarrow f(n_k.\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\beta})$
        $\boldsymbol{x}_{next}.id = id(\boldsymbol{x}_{next})$
        if any $n_j.\boldsymbol{x}.id = \boldsymbol{x}_{next}.id, n_j \in \nu_{closed}$ then
            continue
        else if $g(n_k.\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\beta}) \in \mathcal{X}_{obst}$ then
            continue
        else
            $J_{come} \leftarrow n_k.J_{come} + L_{\boldsymbol{m}}(\boldsymbol{m})$
            $J_{est} \leftarrow J_{come} + h(\boldsymbol{x}_{next}, \boldsymbol{x}_g)$
            if not any $n_j.\boldsymbol{x}.id = \boldsymbol{x}_{next}.id, n_j \in \nu_{open}$ then
                $n_{next}.\boldsymbol{x} \leftarrow \boldsymbol{x}_{next}$
                $n_{next}.J_{come} \leftarrow J_{come}$
                $n_{next}.J_{est} \leftarrow J_{est}$
                $n_{next}.parent \leftarrow n_k$
                $n_{next}.motion \leftarrow \boldsymbol{m}$
                $\nu_{open}.push(n_{next})$
            else if any $n_j.\boldsymbol{x}.id = \boldsymbol{x}_{next}.id$ and $J_{come} < n_j.J_{come}, n_j \in \nu_{open}$
              then
                $\nu_{open}.update\_cost(n_j, J_{come}, J_{est})$
                $\nu_{open}.update\_parent(n_j, n_k)$
                $\nu_{open}.update\_motion(n_j, \boldsymbol{m})$
            end
        end
    end
end
```

5 Results

This chapter contains the results from the simulations, both for the known model simulation, where the true model parameters are user-defined, and the unknown model simulation, where the true model is unknown. Results are obtained from scenarios which consist of the Rover navigating an area with obstacles, using the MPC controller and the motion planner, either with the estimated model or the nominal one.

5.1 Learning-based MPC performance

When evaluating the performance of the learning-based MPC controller, two measures are chosen for benchmarking: the average velocity throughout the run, and the deviated area between the planned path and the actual path the Rover travelled, i.e. the integral of the absolute reference error in position.

Since the MPC controller is supposed to keep the velocity of the Rover as high as possible, the performance is considered to be better the higher the average velocity. Furthermore, the Rover also needs to follow its given path, especially in an environment with a lot of obstacles, which is why smaller deviations from this planned path indicate better performance. However, it is more difficult to follow a path when driving at a higher velocity. Therefore, both of these measures are taken into consideration when comparing different controllers.

The scenarios are split into two sets, one set that evaluates the MPC using the known model simulation, and one using the unknown model simulation. Each of these sets is then further divided into subsets, treating the case of an inaccurate prior, an accurate prior, and a model with a dynamic change during the run (initially using an accurate prior). The definitions of an accurate and an inaccurate model prior are presented in Section 4.1.1.

Each of these subsets is evaluated both with and without the BLR algorithm. In addition, the dynamic change model is also evaluated using the WBLR algorithm. See Figure 5.1 for a graphic clarification.

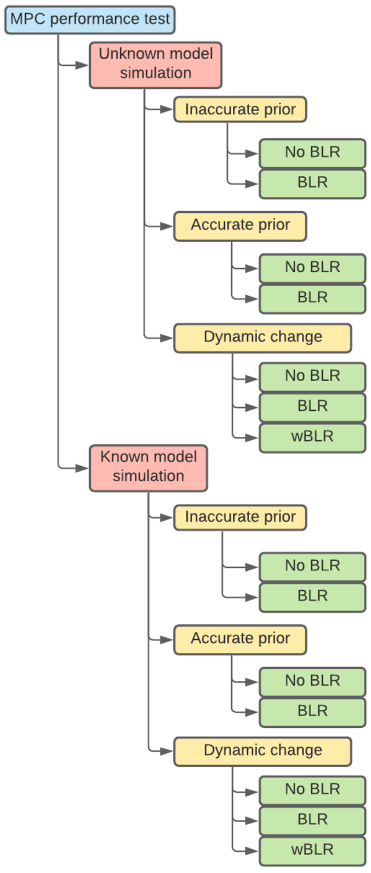

Figure 5.1: *Structure of tests for MPC performance evaluation.*

5.1.1 Known model simulation

This section presents the results of the known model simulation. Table 5.1 summarises the benchmark results for the known model simulation.

Inaccurate prior without BLR

Figure 5.2 illustrates how the MPC performs without the BLR algorithm, with an inaccurate prior. Observe that there is a collision with an obstacle, but as neither this simulator, nor the MPC controller, take collisions into consideration,

Table 5.1: *Summary of the benchmark results for the MPC using the known model simulation.*

MPC benchmark values		
Test type	*Average velocity [m/s]*	*Area deviated [m^2]*
Inaccurate prior		
No BLR	2.074	9.540
BLR	2.040	3.640
Accurate prior		
No BLR	2.034	3.439
BLR	2.038	3.272
Dynamic change		
No BLR	1.531	20.859
BLR	1.522	19.887
WBLR	2.030	10.954

the Rover continues on the path. Table 5.2 shows the true model parameters used in the simulation, and the inaccurately guessed prior.

Table 5.2: *Parameters used in known model simulation for evaluate with an initial inaccurate prior.*

Model parameter values				
Parameter type	w_1^v	w_2^v	w_1^ω	w_2^ω
True parameters in simulation	3	-3	2.1	-3.8
Inaccurate prior guess	5	-5	1	-8

Inaccurate prior with BLR

Figure 5.3 shows how well the MPC follows the planned path when used together with BLR. Expected values of the model parameters $[w_1^v, w_2^v]$ and $[w_1^\omega, w_2^\omega]$ can be seen in Figure 5.4 and 5.5 together with their standard deviations.

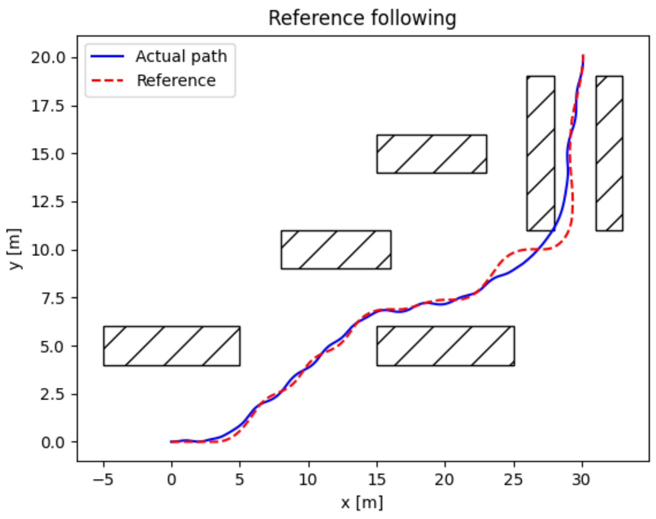

Figure 5.2: *Reference following using an inaccurate prior. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the* MPC *controller.*

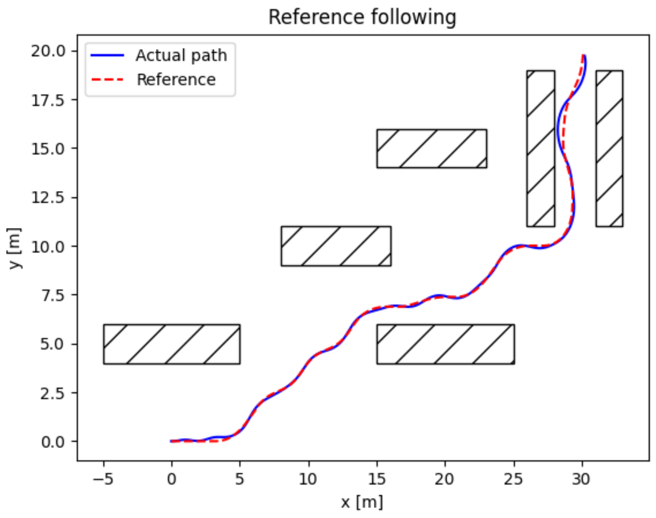

Figure 5.3: *Reference following using an inaccurate prior in the* MPC*, together with* BLR*. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the* MPC *controller.*

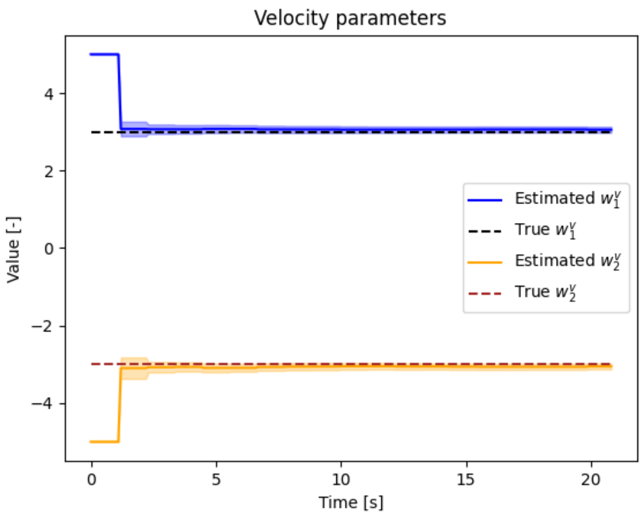

Figure 5.4: *Velocity model parameters compared to true simulation parameters, when using an inaccurate prior with BLR. The intervals show the standard deviation of the parameter estimates.*

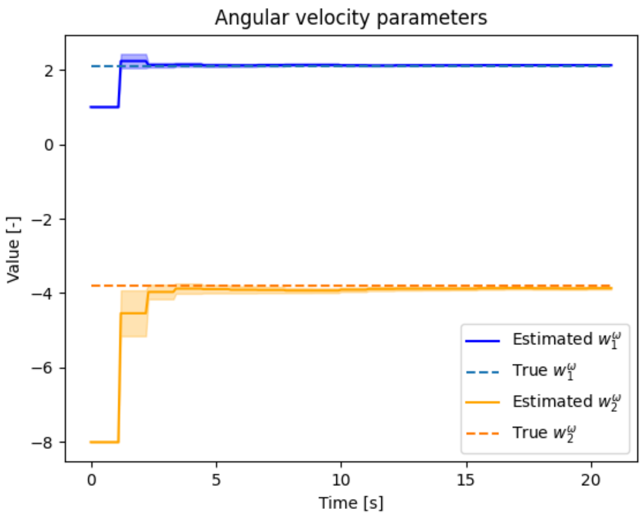

Figure 5.5: *Angular velocity model parameters compared to true simulation parameters, when using an inaccurate prior with BLR. The intervals show the standard deviation of the parameter estimates.*

Accurate prior without BLR

Figure 5.6 illustrates how the MPC performs without the BLR algorithm. The prior is the same as the true model parameters used in the simulator, which are specified in Table 5.2.

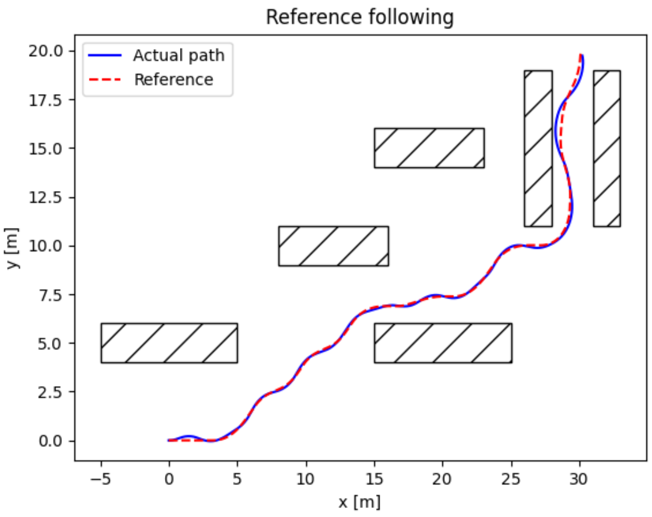

Figure 5.6: *Reference following using an accurate prior in the MPC. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the MPC controller.*

Accurate prior with BLR

Figure 5.7 illustrates how the MPC performs in combination with the BLR algorithm. Expected values of the model parameters $[w_1^v, w_2^v]$ and $[w_1^\omega, w_2^\omega]$ can be seen in Figure 5.8 and 5.9 together with their standard deviations.

Dynamic change – without BLR

In the simulation, a dynamic change occurs 10 seconds into the run, by scaling the angular velocity control input by a factor 0.4, corresponding to a malfunction in the Rover's actuators, making the vehicle under-steered. The prior is the same as the true model parameters used in simulation, specified in Table 5.2. The resulting reference following can be seen in Figure 5.10, where the blue diamond represents the location of the dynamic change.

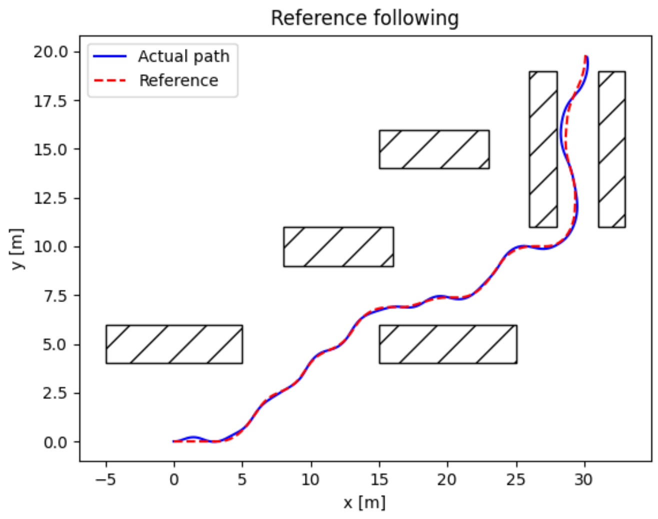

Figure 5.7: *Reference following using an accurate prior in the MPC, together with BLR. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the MPC controller.*

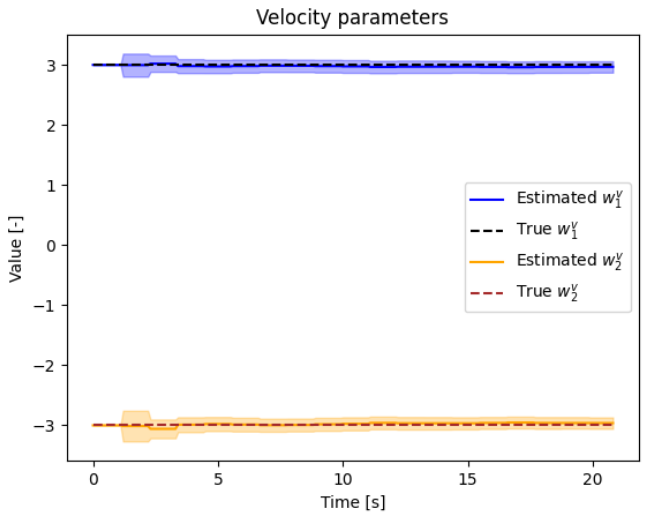

Figure 5.8: *Velocity model parameters compared to true simulation parameters, when using an accurate prior with BLR. The intervals show the standard deviation of the parameter estimates.*

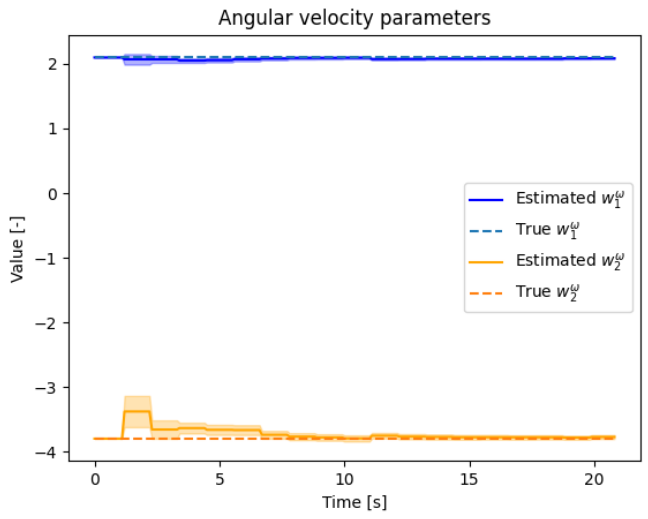

Figure 5.9: *Angular velocity model parameters compared to true simulation parameters, when using an accurate prior with BLR. The intervals show the standard deviation of the parameter estimates.*

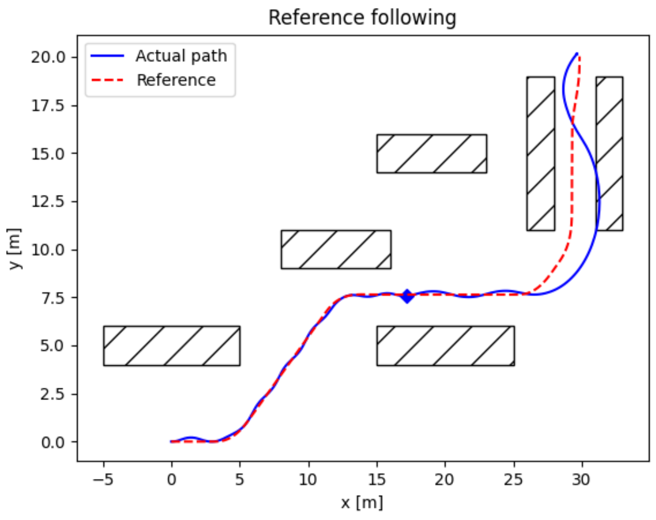

Figure 5.10: *Reference following during a dynamic change without BLR. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the MPC controller. The blue diamond indicates where the dynamic change occurs*

Dynamic change – with BLR

The result from using the BLR algorithm to compensate for the dynamic change can be seen in Figure 5.11, where the blue diamond again indicates where the dynamic change occurs.

Figure 5.12 and 5.13 show the expected values of the model parameters, together with their standard deviations. Furthermore, the initial true parameters used in simulation are shown as a dashed line. Finally, Figure 5.14 illustrates the expected variance of the angular velocity model, together with its standard deviation.

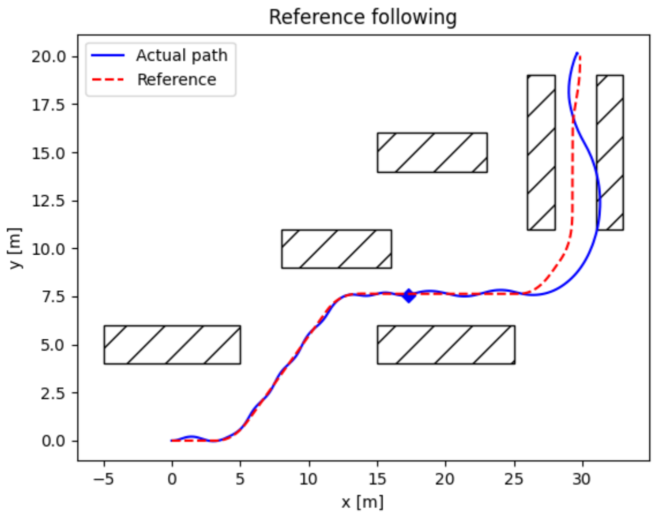

Figure 5.11: *Reference following during a dynamic change with BLR. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the MPC controller. The blue diamond indicates where the dynamic change occurs.*

Dynamic change – using wBLR

The result of using the WBLR algorithm in combination with the MPC, with $n_0 = 2$, which corresponds to 2 effective data points being attributed to the prior, can be seen in Figure 5.15, where the blue diamond indicates where the dynamic change occurs.

Figure 5.16 and 5.17 show the expected values of the model parameters and their standard deviation, and Figure 5.18 shows the expected value of the model variance, where the blue area signifies the estimated standard deviation of the model variance.

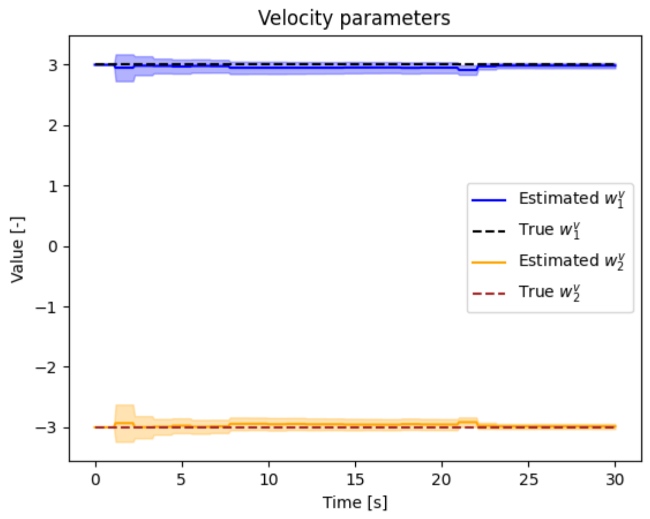

Figure 5.12: *Velocity model parameters compared to true simulation parameters, during a dynamic change with* BLR*. The intervals show the standard deviation of the parameter estimates.*

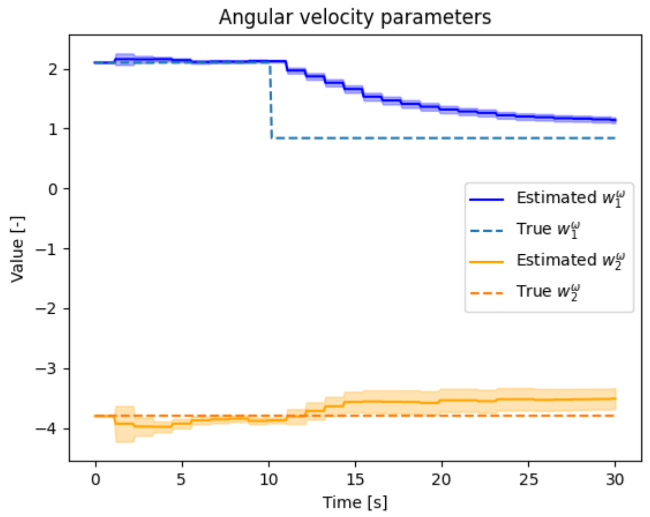

Figure 5.13: *Angular velocity model parameters compared to true simulation parameters, during a dynamic change with* BLR*. The intervals show the standard deviation of the parameter estimates.*

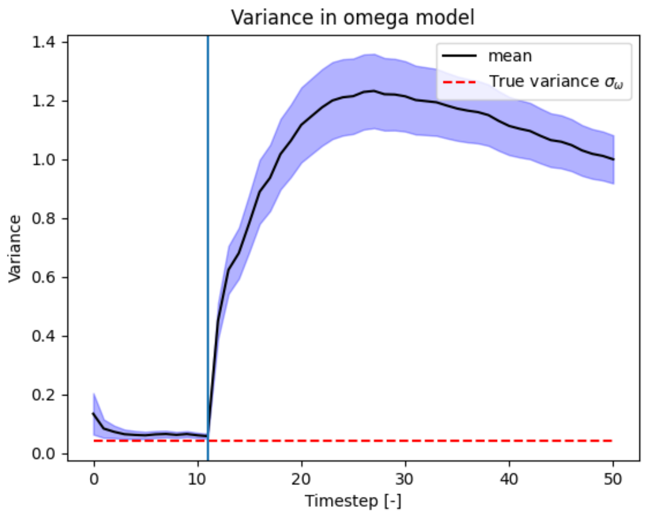

Figure 5.14: *Expected variance of the angular velocity model, during a dynamic change with* BLR*. The blue area shows the standard deviation of the estimated variance. The vertical line indicates where the dynamic change occurs.*

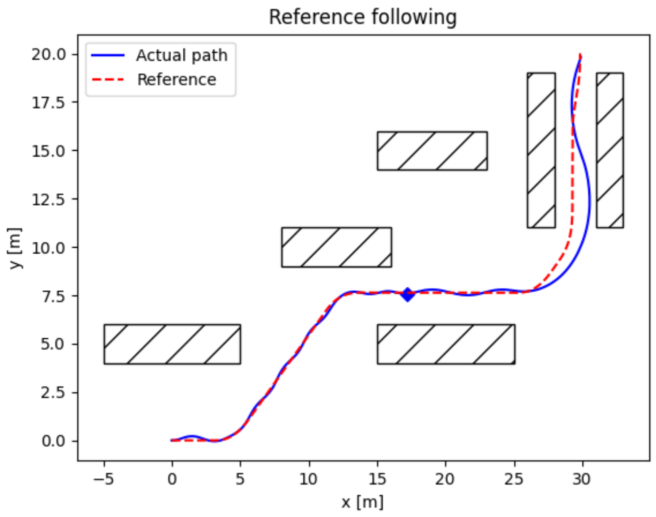

Figure 5.15: *Reference following during a dynamic change with* WBLR*. The red line is the reference path from the motion planner, and the blue line is the path executed by applying the* MPC *controller. The blue diamond indicates where the dynamic change occurs*

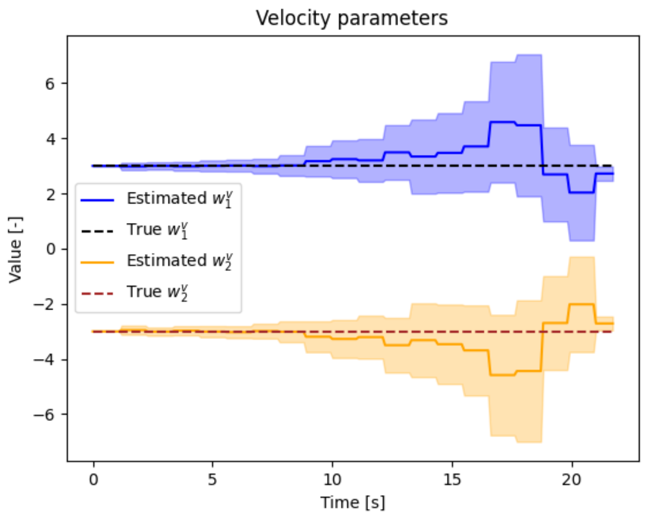

Figure 5.16: *Velocity model parameters compared to true simulation parameters, during a dynamic change with* WBLR*. The intervals show the standard deviation of the parameter estimates.*

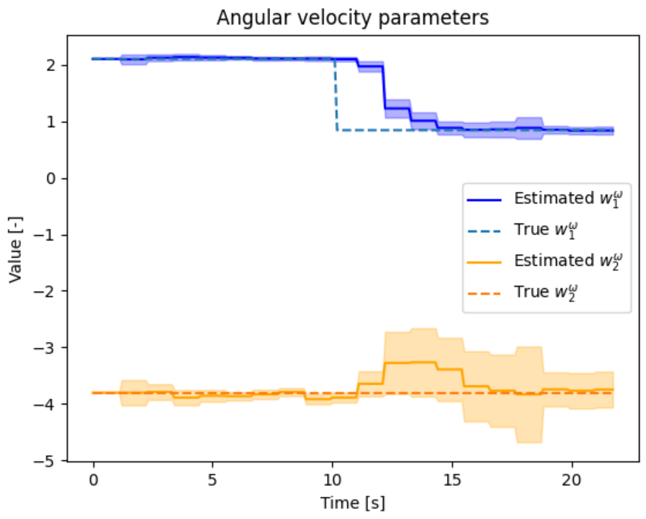

Figure 5.17: *Angular velocity model parameters compared to true simulation parameters, during a dynamic change with* WBLR*. The intervals show the standard deviation of the parameter estimates.*

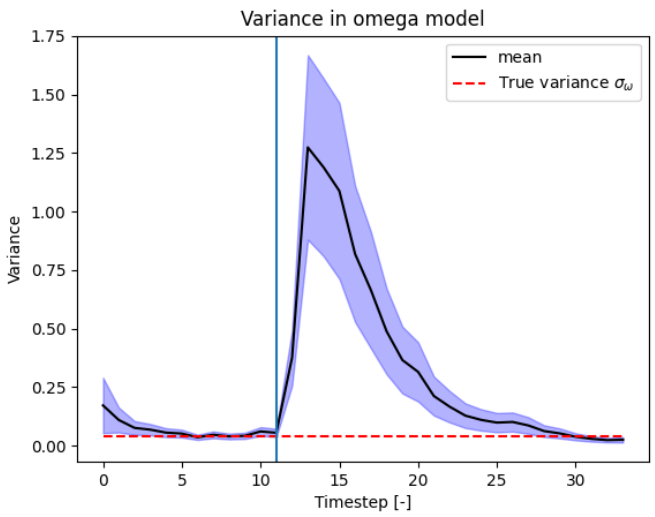

Figure 5.18: *Expected variance of the angular velocity model, during a dynamic change with WBLR. The blue area shows the standard deviation of the estimated variance. The vertical line indicates where the dynamic change occurs.*

5.1.2 Unknown model simulation

The performance measurements for each test for the unknown model simulation are summarised in Table 5.3. The benchmark values for each measurement are the mean of the measurement values from five test runs.

Table 5.3: *Summary of the benchmark results for the MPC using the unknown model simulation.*

MPC benchmark values		
Test type	*Average velocity [m/s]*	*Area deviated [m^2]*
Inaccurate prior		
No BLR	1.83	5.12
BLR	1.93	1.60
Accurate prior		
No BLR	1.93	1.60
BLR	1.92	1.59
Dynamic change		
No BLR	N/A	N/A
BLR	1.65	3.26
WBLR	1.57	1.70

Inaccurate prior without use of BLR

This section shows the results for the MPC controller using an inaccurate prior without the use of the BLR algorithm. The model parameter values are seen in Table 5.4. The path the Rover travelled is seen together with the planned path in Figure 5.19. Since no BLR algorithm is used, the model parameters remain constant during the run.

Table 5.4: *Parameters used in unknown model simulation for test with an initial inaccurate prior.*

Model parameter values				
Parameter type	w_1^v	w_2^v	w_1^ω	w_2^ω
True parameters estimated by BLR	4	-4	2.0	-3.6
Inaccurate prior guess	2	-2	10	-10

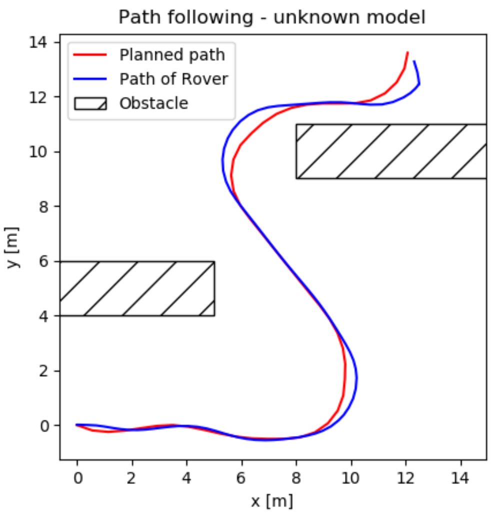

Figure 5.19: *The path of the Rover when the MPC uses an inaccurate prior and the BLR algorithm is disabled.*

Inaccurate prior when using BLR

This section shows the results for the MPC controller using an inaccurate prior while also using the BLR algorithm to further improve the model. The path the Rover travelled is seen together with the planned path in Figure 5.20. The parameters used in the MPC controller for the velocity and the angular velocity models are seen in Figure 5.21 and Figure 5.22, together with their respective standard deviations.

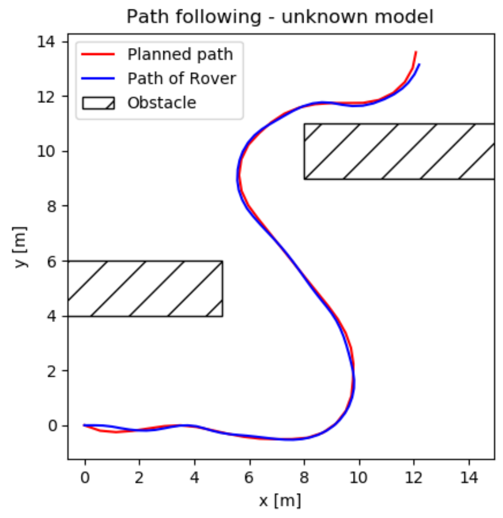

Figure 5.20: *The path of the Rover when the MPC uses an inaccurate prior with the BLR algorithm enabled.*

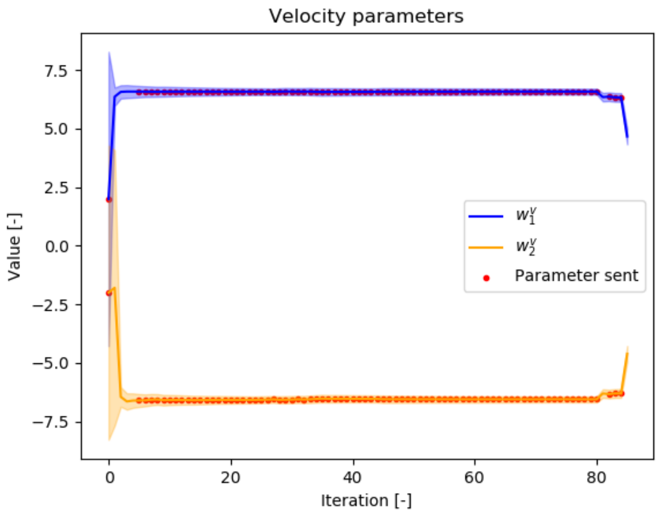

Figure 5.21: *Evolution of the velocity model parameters when using an inaccurate prior together with the BLR algorithm. The intervals show the standard deviation of the parameter estimates.*

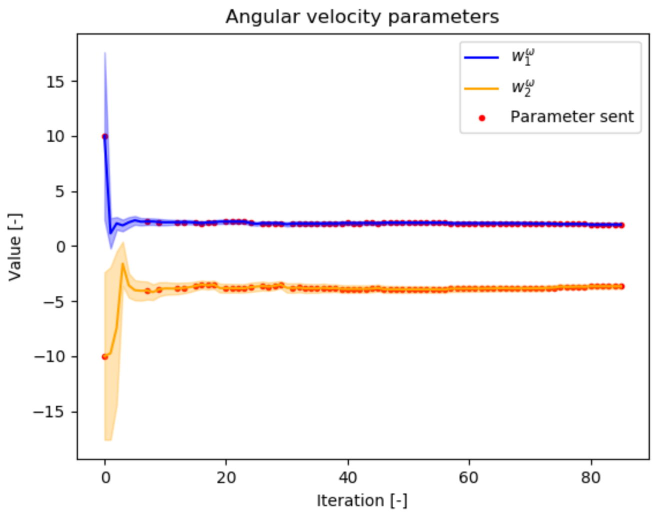

Figure 5.22: *Evolution of the angular velocity model parameters when using an inaccurate prior together with the BLR algorithm. The intervals show the standard deviation of the parameter estimates.*

Accurate prior without use of BLR

This section shows the results for the MPC controller using an accurate prior without use of the BLR algorithm. The path the Rover travelled is seen together with the planned path in Figure 5.23. Since no BLR algorithm is used, the model parameters remain constant during the run.

Accurate prior when using BLR

This section shows the results for the MPC controller using an accurate prior while also using the BLR algorithm to further improve the model. The path the Rover travelled is seen on top of the planned path in Figure 5.24. The parameters used in the MPC controller for the velocity and the angular velocity models are seen in Figure 5.25 and Figure 5.26, together with their respective standard deviations.

Dynamic change without BLR

When investigating the performance of the MPC controller when a dynamic change occurs, the model parameters that the controller uses initially accurately represent the model of the Rover *before* the dynamic change.

When not using the BLR algorithm, the Rover crashes into one of the obstacles due to inability to follow the planned path when the dynamical model of the Rover changes during a run, hence why the average velocity and the area deviated is marked as N/A in Table 5.3.

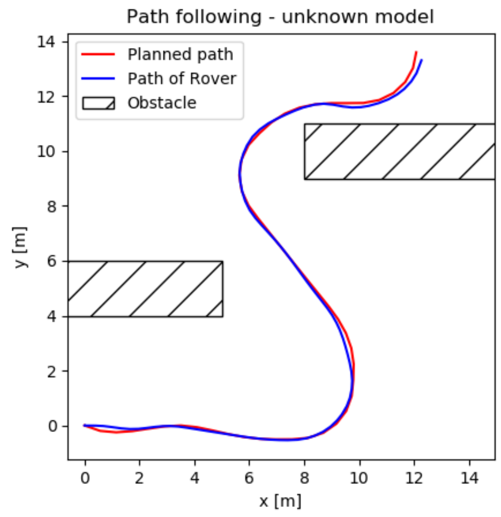

Figure 5.23: *The path of the Rover when the* MPC *uses an accurate prior and the* BLR *algorithm is disabled.*

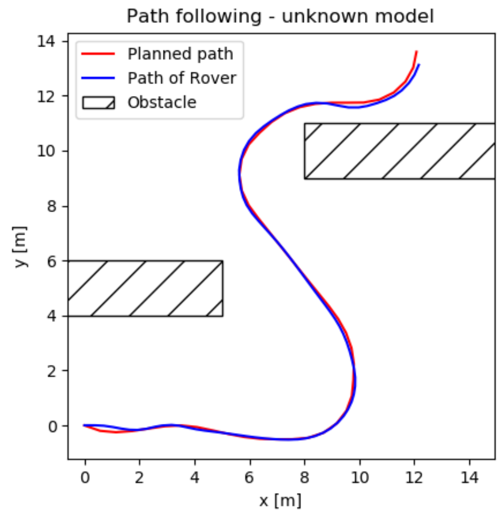

Figure 5.24: *The path of the Rover when the* MPC *uses an accurate prior with the* BLR *algorithm enabled.*

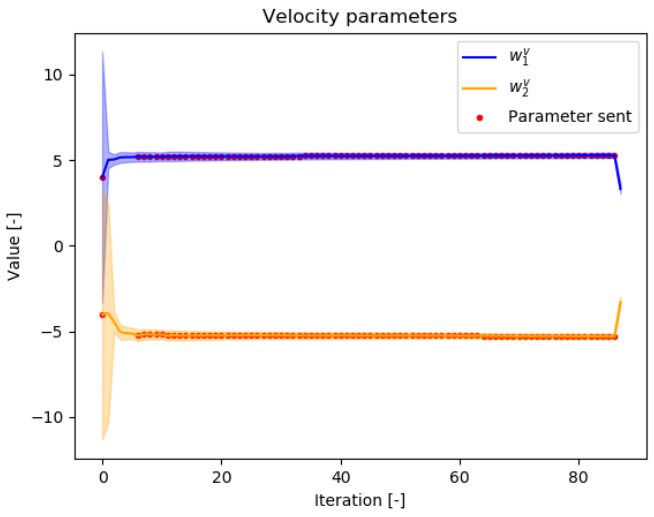

Figure 5.25: *Evolution of the velocity model parameters when using an accurate prior together with the BLR algorithm. The intervals show the standard deviation of the parameter estimates.*

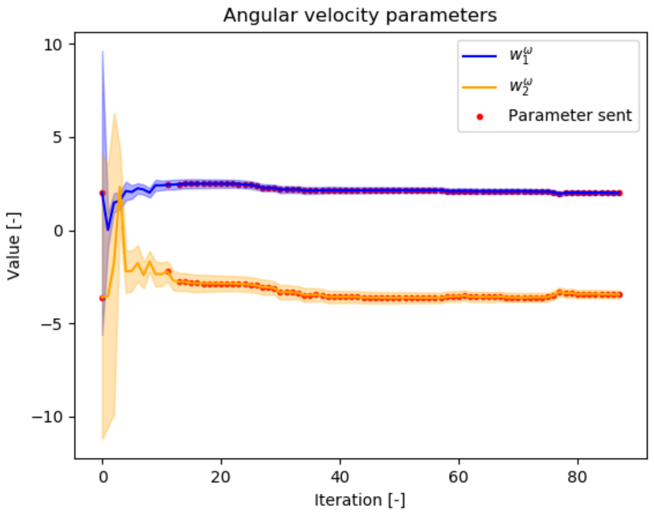

Figure 5.26: *Evolution of the angular velocity model parameters when using an accurate prior together with the BLR algorithm. The intervals show the standard deviation of the parameter estimates.*

The path the Rover travelled is seen together with the planned path in Figure 5.27. Since no BLR algorithm is used, the model parameters remain constant throughout the run.

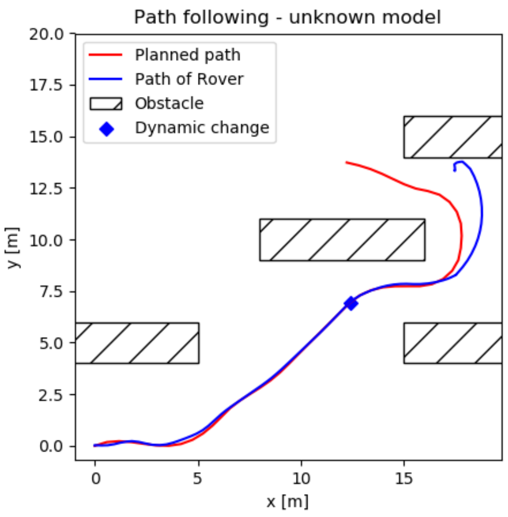

Figure 5.27: *The path of the Rover when a dynamic change occurs mid-run. The MPC uses an accurate prior with the BLR algorithm disabled.*

Dynamic change using BLR

In this section, the BLR algorithm is used to improve the estimates of the model parameters, both before and after the dynamic change.

The path the Rover travelled is seen together with the planned path in Figure 5.28. The parameters used in the MPC controller for the velocity and the angular velocity models are seen in Figure 5.29 and Figure 5.30, together with their respective standard deviations.

Dynamic change using wBLR

In this section, the WBLR algorithm is used to improve the estimate of the model parameters, both before and after the dynamic change. The parameter n_0 was chosen as 50.

The path the Rover travelled is seen together with the planned path in Figure 5.31. The parameters used in the MPC controller for the velocity and the angular velocity models are seen in Figure 5.32 and Figure 5.33, together with their respective standard deviations.

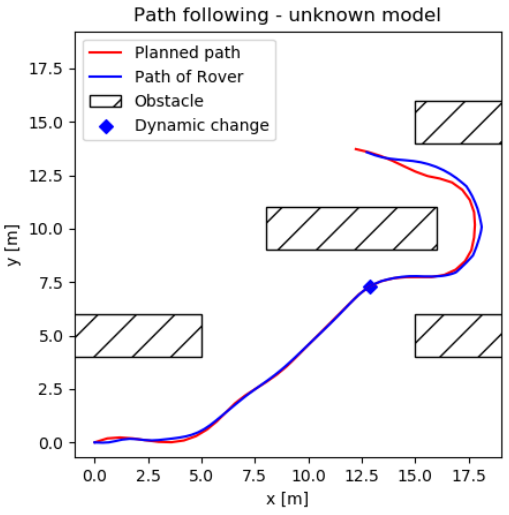

Figure 5.28: *The path of the Rover when a dynamic change occurs mid-run. The MPC uses an accurate prior with the BLR algorithm enabled.*

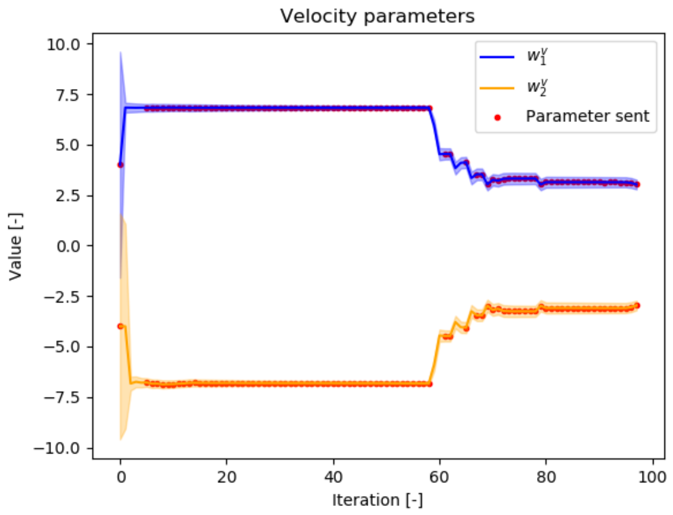

Figure 5.29: *The evolution of the velocity parameters for a dynamic change when using the BLR algorithm. The intervals show the standard deviation of the parameter estimates.*

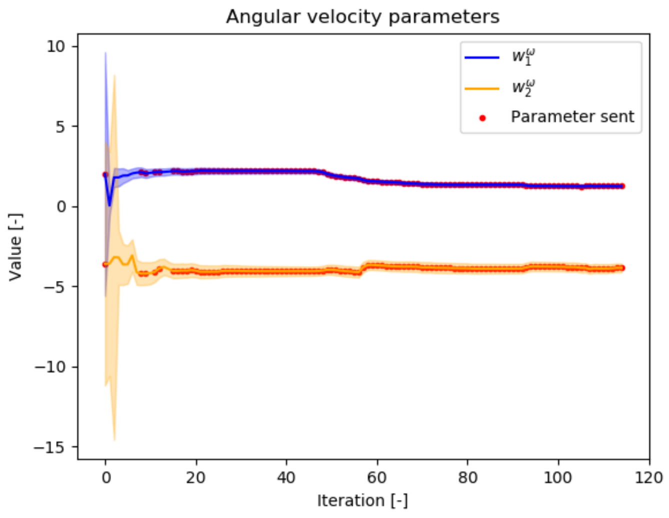

Figure 5.30: *The evolution of the angular velocity parameters for a dynamic change when using the* BLR *algorithm. The intervals show the standard deviation of the parameter estimates.*

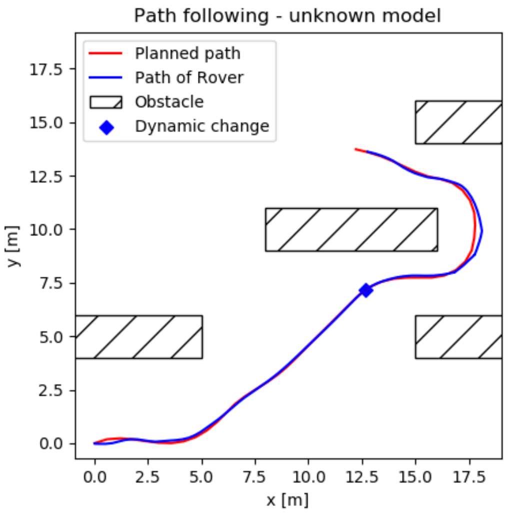

Figure 5.31: *The path of the Rover when a dynamic change occurs mid-run. The* MPC *uses an accurate prior with the* WBLR *algorithm enabled.*

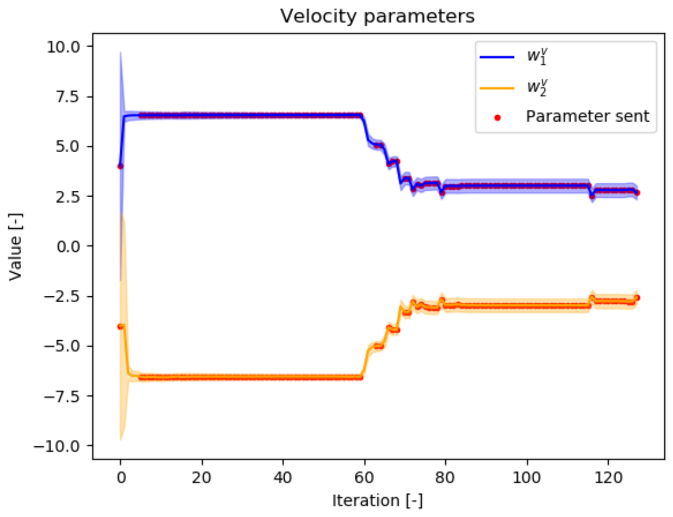

Figure 5.32: *The evolution of the velocity parameters for a dynamic change when using the* WBLR *algorithm. The intervals show the standard deviation of the parameter estimates.*

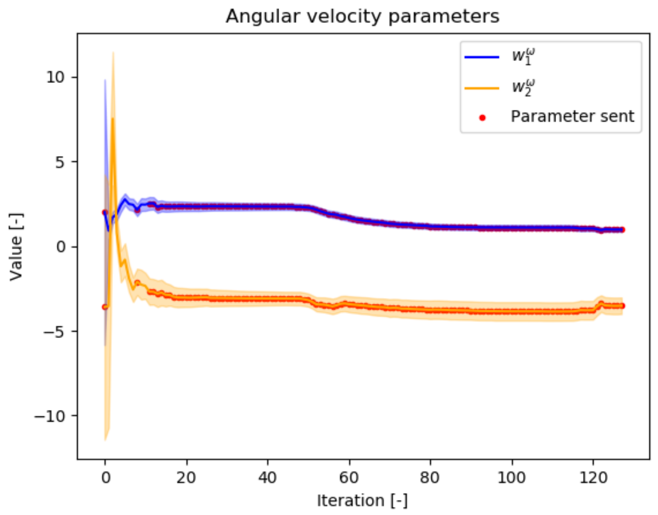

Figure 5.33: *The evolution of the angular velocity parameters for a dynamic change when using the* WBLR *algorithm. The intervals show the standard deviation of the parameter estimates.*

5.2 Motion planner performance

This section presents the results regarding the performance of the motion planner. A desirable quality of the motion planner is that it computes a feasible path in the shortest time possible. Therefore, the measures used are the time the planner takes to find the first state that corresponds to the goal state (possibly a sub-optimal path), the time until the algorithm terminates, meaning that the vertex containing the goal state is popped from the open list (resolution optimal), and how well the MPC is capable of making the Rover follow the planned path. The last measure is composed of the same measures used to benchmark the MPC controller in Section 5.1, namely the average velocity of the Rover and the area the actual travelled path deviates from the planned path.

The tests are performed with an MPC controller, in combination with the BLR algorithm, initialised with accurate model parameters generated from the BLR algorithm in the unknown model simulation (Gazebo). The dynamical models in the OCP generating the motion primitives and in the hybrid A* algorithm use either an accurate prior, computed by the BLR algorithm, or an inaccurate prior, which both can be seen in Table 5.4. To give fair comparison between the accurate and the inaccurate prior, the path is always planned between the same states. Simulations are performed five times for the inaccurate and accurate prior, and the performance measures are taken as the average over these runs. The values of these measures are summarised in Table 5.5.

Table 5.5: *Summary of the benchmark results for the motion planner.*

Motion planner benchmark values		
Performance measure	**Inaccurate prior**	**Accurate prior**
Time to first complete path [s]	22.90	6.49
Time to optimal path [s]	24.58	8.80
Average velocity [m/s]	1.92	1.94
Area deviated [m^2]	2.82	1.73

5.2.1 Inaccurate prior

The resulting planned path when an inaccurate prior was used can be seen in Figure 5.34, while the path the Rover drove is seen in Figure 5.35.

5.2.2 Accurate prior

Since the motion planner does not re-plan the path online, the way the BLR algorithm's parameter estimates are used is by saving the latest computed estimates for each run, then using said values when planning the next path. The resulting planned path is seen in Figure 5.36, while the path the Rover travelled can be seen in Figure 5.37.

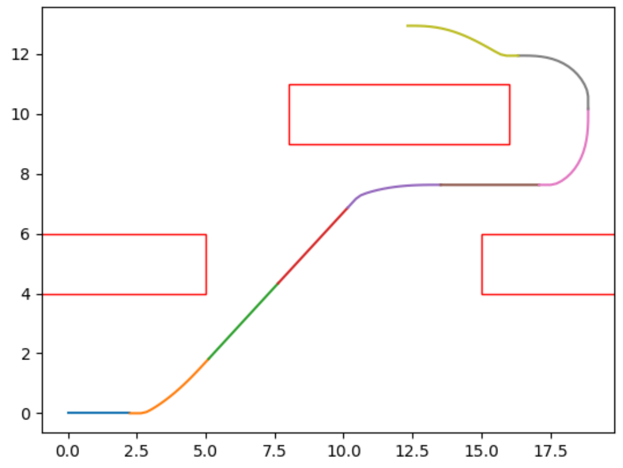

Figure 5.34: *The planned path when using inaccurate model parameters. The colours indicate where a motion primitive starts and ends.*

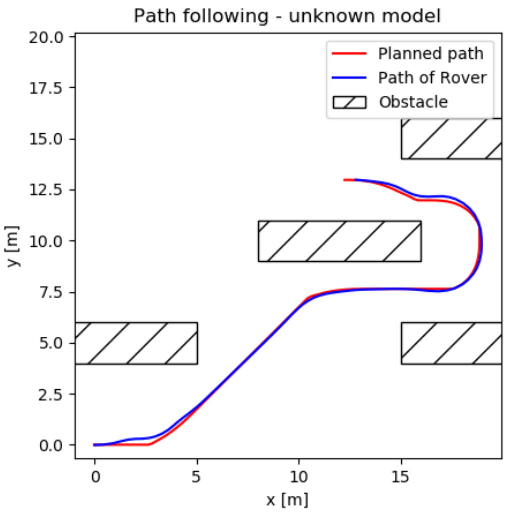

Figure 5.35: *How the Rover managed to follow the planned path when an inaccurate prior was used to generate the path.*

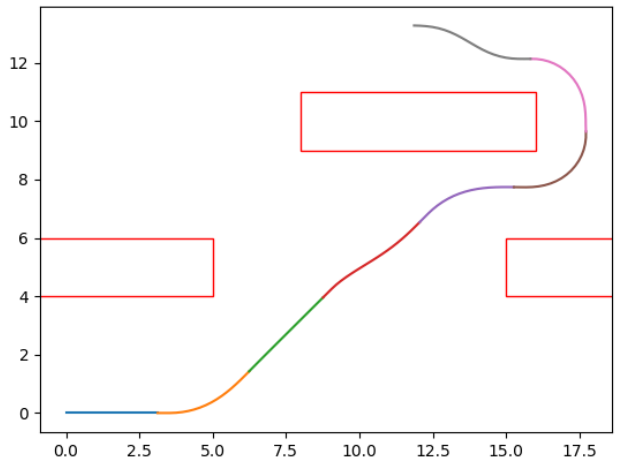

Figure 5.36: *The planned path when using accurate model parameters. The colours indicate where a motion primitive starts and ends.*

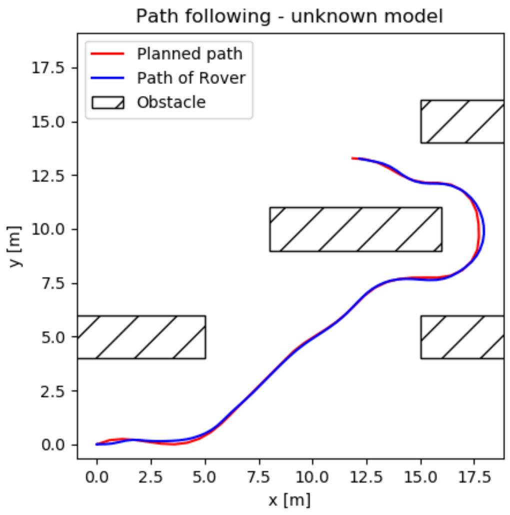

Figure 5.37: *How the Rover managed to follow the planned path when an accurate prior was used to generate the path.*

6 Discussion

In this chapter, the choice of implementation methods from Chapter 3 and Chapter 4, as well as the results from Chapter 5 are discussed.

6.1 Implementation methods

This section covers the choice of implementation methods. The reasoning behind choosing the methods from Chapter 3 and Chapter 4 are covered, as well as how these methods affect the results presented in Chapter 5.

The model of the Rover was chosen because of its simplicity: it is a vehicle in the 2D-plane, it is linear in the velocity and the angular velocity and it has the ability to rotate without translating, just like a real Rover. The fact that the model is partly linear means that simpler methods for both control and machine learning are available, and less computational power is required, meaning that control signals from the MPC controller can be computed faster and sent at a higher rate to the lower-level controllers, often increasing the performance of the system as it can react more rapidly to disturbances.

To estimate the model parameters, a BLR algorithm is used. This is due to its low computational complexity, even for large amounts of data. Another benefit is that the BLR algorithm is more intuitive when it comes to weighing the data compared to e.g. Gaussian process regression, meaning that the importance on the regression for each data point can explicitly be taken into account by using a parameter as in the WBLR algorithm seen in Section 2.2.5.

When choosing a controller for the Rover, one that uses a dynamical model of the system to compute the control signals is needed in order to exploit the machine learning algorithm. For example, an LQR regulator can be used in combination with the BLR algorithm, requiring only a solution to the Riccati equation to be computed. However, the ability to impose constraints on control signals or using a non-linear model, which is more realistic in an actual physical application where control signals often are saturated and non-linear models are used, is not possible for the LQR algorithm. Therefore, the MPC controller was chosen, for which constraints on both the control signals and the states can be imposed, while also the model is allowed to be non-linear.

The usage of motion primitives in the motion planning algorithm is motivated by the fact that it is a well-used method to take the dynamical model of the system into consideration during planning. Since the resulting problem is in the form of a standard graph search problem, there exist algorithms, e.g. A*, that are able to compute resolution optimal solutions based on the library of motion primitives. Since the system might be time-variant, an input sampling based motion planner together with the hybrid A* algorithm is chosen as it allows the system to re-plan using updated estimates of the model parameters without having to re-discretise the search space and recompute the motion primitives.

6.2 Results

In this section the obtained results seen in Chapter 5 are discussed. The discussion covers the quality of the results and how the results can be interpreted.

6.2.1 MPC performance

By using the results in Section 5.1, this section analyses how well the MPC controller, paired with the BLR algorithm, performs during the simulations. The analysis starts with the inaccurate prior, followed by the accurate prior and finally the simulation with a dynamic change is examined.

Inaccurate priors

When the MPC controller uses the model with inaccurate model parameters, it is of interest to compare how it performs when receiving updated model parameter estimates from the BLR algorithm, with the performance when only using the initial, inaccurate prior.

Starting without using the BLR algorithm, Figure 5.2 shows the path that the Rover travelled (blue) compared to the planned path (red) when using the known model simulation, while Figure 5.19 shows the reference following, but for the unknown model simulated using Gazebo. As no BLR algorithm was used to update the model parameter estimates, the numerical parameter values do not change during the run, resulting in poor reference tracking.

How the Rover managed to follow the planned path when instead using the BLR algorithm can be seen in Figure 5.3 for the known model simulation, and Figure 5.20 for the unknown model simulation. Here, the parameter estimates do change which can be seen in Figure 5.4 and Figure 5.5 for the known model simulation. The figures show both the estimated parameters as well as the true parameters – since the model is known – and it can be seen that the BLR algorithm identifies the true parameters after only a few data points. When simulating an unknown model, the true model parameters are unknown, but the evolution of the parameter estimates can be seen in Figure 5.21 and Figure 5.22, as well as the instances when they are sent to the MPC controller and the motion planner. These parameter estimates seem to converge towards values that more accurately describe the model of the Rover, since the path following from the MPC is improved with the evolution of the model parameter estimates.

The benchmark values for this test can be seen in Table 5.1 for the known model simulation and Table 5.3 for the unknown model simulation. When comparing the results from these two tables, there is a clear improvement in path following for both the known model simulation and the unknown model simulation when using the BLR algorithm, while the average velocity stays roughly the same in the two cases. This indicates that the MPC controller does perform better when paired with a BLR algorithm.

Accurate priors

When using already (sufficiently) accurate model parameters, the difference in performance between using the BLR algorithm or not is, unsurprisingly, almost non-existent. Since the initial parameter estimates lie close enough to true parameter values, there is little room for any improvement.

This is supported by looking at Table 5.1 and Table 5.3 as the average velocity and area deviated with and without BLR is nearly identical for both known model simulation and unknown. The reference tracking of the Rover when using the known model simulation can be seen in Figure 5.6 when the BLR algorithm is disabled and Figure 5.7 when this algorithm is enabled. When instead using the unknown model simulation, the Rover's path compared to the planned path can be seen in Figure 5.23 when not using the BLR algorithm, and in Figure 5.24 when using BLR. The path following seems to be identical whether or not BLR is used, regardless of what kind of simulation that is being used.

What these results indicate, is that the usage of a BLR algorithm will not impact the performance negatively in the case of accurate priors. This means that a BLR algorithm can be executed in the background, collecting data, computing and sending the parameter estimates to the MPC controller at any time, without disturbing the system or decreasing its performance noticeably.

Dynamic change model

This is, in many cases, the most interesting scenario for the application of a BLR algorithm to aid the MPC controller. What happens during this test is that the true model parameters of the system change during the run, i.e. the dynamic change mentioned in Section 5.1.1.

When the BLR algorithm is disabled during the run, it can be seen in Figure 5.10 that the Rover collides with an obstacle and continues on the path as there is no collision check for the known model simulation. When instead using the unknown model simulation, Figure 5.27 shows that the Rover in a similar fashion collides with one of the obstacles. In this simulation, however, collision checks are enabled, causing the Rover to crash, ending up upside-down with no connection between its tracks and the ground, rendering it unable to complete the run. Since the Rover does not complete the run, the average velocity and the deviation from the planned path can not be calculated, which is why no data is available for this case in Table 5.3.

When instead using the BLR algorithm to estimate the updated model parameters, the Rover is able to reach the goal, completing the run without crashing – which itself is a great improvement – as seen in Figure 5.28 for unknown model simulation. It does however still collide with obstacles for known model simulation, as seen in Figure 5.11. When comparing these figures to Figure 5.15 (known model simulation) and Figure 5.31 (unknown model simulation) that show the result from using the WBLR algorithm to estimate the model parameters sent to the MPC controller, it can be seen that the WBLR algorithm enables the MPC to perform even better than the regular BLR algorithm. This is supported by the benchmark values in Table 5.1 and Table 5.3, where the area deviated has clearly decreased using WBLR, while the average velocity is roughly the same in the two for the unknown model simulation. There is however a difference in velocity for the known model simulation, as can be seen in Table 5.1, where using WBLR results in a higher average velocity.

Another interesting aspect is to compare the parameter evolution for BLR, seen in Figure 5.12 and Figure 5.13 for known model simulation, and the parameter development for WBLR seen in Figure 5.16 and Figure 5.17. The parameter adaptation is seen to be much faster for WBLR, while it also is much more uncertain as the standard deviation for the model parameters is larger. This is quite reasonable, as WBLR cannot rely as much on "older" data as ordinary BLR to compute the model parameters.

In the case of the unknown model simulation, the WBLR algorithm behaves in similar ways. The model parameters estimates adapt quicker when using the WBLR algorithm, but the uncertainty in the parameters is higher than the parameter uncertainty the BLR algorithm computes. This can be seen for the angular velocity parameters when comparing Figure 5.30 with Figure 5.33.

6.2.2 Motion planner performance

This section covers the performance of the motion planner with and without the BLR algorithm, using the results from Section 5.2.

From the results in Table 5.5, it is possible to compare the benchmark values for the model using inaccurate parameters with the one using accurate parameters. Studying the planning times seen in Table 5.5, it is clear that in this case, the use of the accurate model parameters computed by the BLR algorithm gives a much faster planning time than when using inaccurate model parameters, which of course is desirable. Both the time to find *any* feasible path to the goal state and the *optimal* path is faster for the planner using the accurate model. However, there are no guarantees that one model would give a lower planning time than another, as this depends on, e.g., the initial and final states, the fidelity and the heuristic used.

When studying Table 5.5 and comparing the planned path in Figure 5.34, that is generated from the motion primitives using inaccurate priors, with the planned path in Figure 5.36, generated with motion primitives using an accurate prior, it is clear that the motion primitives generated using the accurate model parameters allow for better system performance.

Considering how the Rover managed to follow the two paths in Figure 5.35 (inaccurate prior) and Figure 5.37 (accurate prior), the accurate model seems to generate a path that is easier for the system to follow, which can be confirmed examining the values for the deviated area in Table 5.5.

The average velocity that the Rover manages to keep is slightly higher when following the path generated using the accurate prior, than when following the path generated using the inaccurate prior. It is therefore possible to conclude that the over-all performance does benefit from an updated set of model parameters in the motion planner.

The case where the system experiences a change of its dynamics during the run has not been considered for the motion planner. The reason is that the motion planner would not benefit from updated model parameter estimates during execution of a plan, since it does not re-plan the path during a run. This is discussed further in Section 7.2.

7 Conclusions and future work

In this thesis, the control of a UGV with a partially unknown and time-variant dynamical model, using learning-based motion planning and control, has been studied. The purpose of this chapter is to summarise these studies and the results gathered, as well as answering the questions in the problem formulation.

The model parameters in the unknown model of the Rover are estimated using the machine learning algorithm BLR, while the motion planner uses motion primitives in combination with a hybrid A* algorithm to plan a path. The performance is evaluated by using either accurate or inaccurate priors in the BLR algorithm, the MPC controller and the motion planner, together with either a time-invariant or a time-variant dynamical model of the Rover. The BLR algorithm was either enabled, meaning it sends the estimated model parameters to the MPC and the motion planner, or disabled, meaning that the MPC controller and the motion planner use the initial prior throughout the simulation. The answers to the questions posed in Chapter 1 are given in Section 7.1.

7.1 Answers to the problem formulation

What methods can be used to identify the unknown dynamics of the system?

When conducting the literature study during the early stages of the thesis, *two* methods for improving the dynamical model of the Rover were investigated. The first, and also the seemingly most commonly used method, is the GP regression method, where the function describing the dynamics of the Rover are estimated. The advantage of this method is that a large amount of papers has already exploited it, describing the necessary theory and how it could be implemented. One of the disadvantages is that the models using GP regression tend to scale badly with the amount of data points, requiring more computational power. This method is also a nonparametric method, and since the dynamical equations in

(3.2) are assumed, it makes sense to instead investigate and use the parametric Bayesian linear regression method.

Since the BLR method is parametric, all the required information needed to model the dynamics is contained in the parameters, which are estimated. This simplifies the procedure of providing the MPC controller and motion planner with the parameter estimates, since they use the estimates as model parameters. Furthermore, the computational power scales linearly with the number of data points for the BLR algorithm, since the number of operations are always the same, regardless of the number of data points already used by the algorithm – but the sizes of the matrices may vary depending on how many data points are added at once. A drawback of this method is the lack of available papers on the subject, but the few that exist treat the topic in an explanatory manner.

Can the estimated model be used in sampling-based motion planning algorithms, and will it improve performance compared to using the nominal model?

Since no other papers treating this subject can be found, this thesis have potentially made some new contributions to this field as it is indeed possible to use estimated models online in a sampling based motion planner. The motion primitives computed with more accurate parameters allow the planner to generate a path that is better adapted to the dynamical model of the Rover. This means that the MPC is capable of making the Rover follow the path more accurately – resulting in an increased performance of the system.

How does the performance of an MPC controller based on an estimated model compare to the performance of an MPC controller based on the nominal model?

In Chapter 6, it is shown that the performance of the MPC controller is improved when simultaneously running the BLR algorithm to provide new parameter estimates. The main improvements are seen when using an inaccurate prior or when a dynamic change of the model parameters is introduced during a simulation. However, the BLR algorithm does not improve the MPC performance when the controller initially uses an accurate prior, but it does not decrease the performance either. This means that the algorithm can run in the background without affecting performance until, e.g., a change of the model parameters happens, in which case the BLR algorithm computes new parameter estimates, sending them to the MPC controller.

Can a machine learning algorithm be used to detect a dynamic change in the system model while the system is running, and adapt the model to this change?

The BLR algorithm constantly computes new estimates of the system dynamics model parameters, adding the latest data point each iteration. When the new estimates are sent to the MPC controller and the motion planner, their respective models will adapt, leading to a better performance when encountering a dynamic change. In order to better estimate the new model parameters, the WBLR algorithm should be used. In this case, the n_0 parameter can be used to tune how fast the system is adapted to the new conditions. If it is desired to also *detect* when a dynamic change happens, a detection algorithm that uses, e.g., the model variance estimates and the model parameter estimates could be developed.

7.2 Future work

The results support that the BLR algorithm can indeed improve the control performance of a system such as the Rover's. However, this is so far more of a proof-of-concept, since nothing is yet implemented on physical hardware. This section presents some of the possible future work within this area, and gives some possible directions the work in this thesis can take if it was to be continued.

At the beginning of this thesis, the plan was to implement a system that could be used on actual hardware in the form of a physical Rover, without too much modification. However, this thesis is limited to software simulation of the system, but it would be interesting to perform some real-world experiments to verify that the methods work on hardware too, with data from real sensors.

Another interesting idea is to investigate how the methods in this thesis work on other systems that are time-variant, i.e. with a dynamical model that changes over time. One intriguing example could be a rocket, where the mass decreases over time due to the consumption of fuel and the aerodynamic properties change with the composition of the air surrounding the rocket.

There is no method presented in this thesis to safely detect, and much less guarantee, that a dynamic change of the model parameters has happened. As such, it would be of interest to explore algorithms that can use data such as estimated model variance and model parameter variance to determine if a change has happened, and in that case evaluate if the new parameters should be used in the system. This is especially interesting if the machine learning algorithm is expected to be used on a system which is time-variant.

Finally, in order to further benefit from the parameter estimates that the BLR algorithm computes, the implementation of an overall mission planner would be interesting to investigate. As of now, the motion planner only re-plans using new parameter estimates when it is given a new goal. With a mission planner, it becomes possible to temporarily stop the Rover and then re-plan the path to the current goal using the latest (and most likely more accurate) parameter estimates. In addition, it is possible to plan a new path during a run by planning a new path to the goal state from a new state further ahead on the reference, called the *staging state*. The staging state must be selected such that the planner is able to update the path before reaching this state. The new path then replaces the original path between the staging state and the goal state, thus making it possible to compute an improved reference path online, while still executing the mission without interruption.

Bibliography

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[2] Kristoffer Bergman. *On motion planning using numerical optimal control*, volume 1843. Linköping University Electronic Press, 2019.

[3] Kristoffer Bergman. *Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments*. PhD thesis, Linköping University Electronic Press, 2021.

[4] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved optimization of motion primitives for motion planning in state lattices. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2307–2314. IEEE, 2019.

[5] Felix Berkenkamp and Angela P Schoellig. Learning-based robust control: Guaranteeing stability while improving performance. In *IEEE/RSJ Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[6] Vishnu R Desaraju, Alexander Spitzer, and Nathan Michael. Experience-driven predictive control with robust constraint satisfaction under time-varying state uncertainty. In *Robotics: Science and Systems*, 2017.

[7] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The international journal of robotics research*, 29(5):485–501, 2010.

[8] Martin Enqvist et al. Industriell reglerteknik. *Kurskompendium. Linköping: Reglerteknik, Institutionen för systemteknik*, 2014.

[9] Guilherme AA Gonçalves and Martin Guay. Robust discrete-time set-based adaptive predictive control for nonlinear systems. *Journal of Process Control*, 39:111–122, 2016.

[10] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Donald M Hassler, Cary Zeitlin, Robert F Wimmer-Schweingruber, Bent Ehresmann, Scot Rafkin, Jennifer L Eigenbrode, David E Brinza, Gerald Weigle, Stephan Böttcher, Eckart Böhm, et al. Mars' surface radiation environment measured with the mars science laboratory's curiosity rover. *science*, 343(6169), 2014.

[12] Lukas Hewing, Kim P Wabersich, Marcel Menner, and Melanie N Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:269–296, 2020.

[13] Katy Klauenberg, Gerd Wübbeler, Bodo Mickan, Peter Harris, and Clemens Elster. A tutorial on bayesian normal linear regression. *Metrologia*, 52(6): 878, 2015.

[14] Richard E Korf and Michael Reid. Complexity analysis of admissible heuristic search. In *AAAI/IAAI*, pages 305–310, 1998.

[15] Volvo Lastvagnar. Veras första uppdrag. `https://www.volvotrucks.se/sv-se/news/press-releases/2019/jun/pressrelease-190613.html`, 2019. Accessed: 2021-05-17.

[16] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[17] Magnus Minnema Lindhé. *Communication-aware motion planning for mobile robots*. PhD thesis, PhD Dissertation, KTH, Stockholm, 2012.

[18] Christopher D McKinnon and Angela P Schoellig. Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks. *IEEE Robotics and Automation Letters*, 4(2):2180–2187, 2019.

[19] Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999.

[20] Satoshi Morita, Peter F Thall, and Peter Müller. Evaluating the impact of prior assumptions in bayesian biostatistics. *Statistics in biosciences*, 2(1): 1–17, 2010.

[21] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[23] Open Source Robotic Foundation (OSRF). Gazebo. `http://gazebosim.org/`, 2002–2021.

[24] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[25] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

[26] Roushan Rezvani Arany. Gaussian process model predictive control for autonomous driving in safety-critical scenarios, 2019.

[27] Marwan Salem. Building an efficient occupancy grid map based on lidar data fusion for autonomous driving applications, 2019.

[28] Sergey Vichik and Francesco Borrelli. Solving linear and quadratic programs with an analog circuit. *Computers & Chemical Engineering*, 70:160–171, 2014.

[29] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[30] Linus Wiik and Jennie Bäcklin. Collaborative exploration of unknown terrain utilizing real-time kinematic positioning. Master's thesis, Linköping University, Automatic Control, 2020.

[31] Wikipedia contributors. Graph traversal — Wikipedia, the free encyclopedia, 2020. URL `https://en.wikipedia.org/w/index.php?title=Graph_traversal&oldid=997453401`. [Online; accessed 17-March-2021].

[32] Wikipedia contributors. Runge–kutta methods — Wikipedia, the free encyclopedia, 2021. URL `https://en.wikipedia.org/w/index.php?title=Runge%E2%80%93Kutta_methods&oldid=1021130819`. [Online; accessed 18-May-2021].

[33] G Alastair Young, Thomas A Severini, George Albert Young, RL Smith, Robert Leslie Smith, et al. *Essentials of statistical inference*, volume 16. Cambridge University Press, 2005.