POMDP based online parameter estimation for autonomous passenger vehicles

Researching online tyre parameter estimation performance by improving the trajectory using a POMDP algorithm.

Quinn Vroom - 4225554

Master of Science Thesis

©Tesla inc.

TU Delft Delft University of Technology

Department of Cognitive Robotics - Vehicle Engineering

POMDP based online parameter estimation for autonomous passenger vehicles

Researching online tyre parameter estimation performance by improving the trajectory using a POMDP algorithm.

Master of Science Thesis

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Quinn Vroom - 4225554

May 28, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright © Department of Cognitive Robotics (CoR)
All rights reserved.

Abstract

The internal model is an important piece of the control system of an autonomous driving vehicle. In order for the model to deliver accurate predictions, a valid model structure and well chosen parameters are needed. Model parameters can be highly fluctuating or complex to predict, especially when looking into tyre ground surface interaction models. Instead of predicting parameter values beforehand, they could be estimated and updated in real-time. Fluctuation or incorrectness can be adjusted while driving. However, this uncertainty in parameter value must be accounted for when applying control. Solving this problem by regarding the uncertainty in parameters of the internal vehicle model as a POMDP has been researched in this paper. The research question being: is it worthwhile to use the POMDP approach for online parameter estimation of autonomous passenger vehicles? To answer this multiple sub-questions have been composed. We start off looking into: what is the most suitable vehicle model? Different vehicle models and tyre models were compared. Literature showed the bicycle model in combination with the linearized tyre model to be most suitable for autonomous passenger vehicles. The next question is: What is the most promising algorithm? Using literature, suitable algorithms for solving this POMDP have been found and compared. From three compelling algorithms, the one best fitting the autonomous driving criteria was chosen. Knowing the model and the algorithm for the simulation the next question became: Does the algorithm perform on a vehicle model? To answer this question, the simulation has been implemented in MATLAB and performance has been tested. The results showed significant increase in parameter estimation performance. Within 2 timesteps the estimate had converged correctly. The next question is: Does the algorithm perform within realistic bounds? To answer this question, the same simulation as before has been used, but now with saturation on the steering input. This showed parameter estimation performance increase compared to the original trajectory, but not as overwhelming as without saturation. The next question is: Does the algorithm suffer from high noise? To answer this question, the same simulation has been used, but now with different levels of noise. The results showed parameter estimation performance significantly affected by increasing noise. The final sub-question is: Does the algorithm suit increasing model complexity? To answer this question, the amount of parameters have been increased in the simulation and there has been looked into the large matrices that accompany the algorithm. Results showed that increasing the

complexity has a significant effect on the size of the simulation and algorithm matrices. In conclusion, from all of these experiments arose some very interesting results. This produced a useful insight into the strengths and weaknesses of the POMDP algorithm performing on a passenger vehicle, answering the research question. This also led to various recommendations for future research. Interesting would be altering the belief filter to enhance performance on bounded steering input and increased sensor noise.

Table of Contents

Preface

Before you lies my Master of Science graduation thesis, where I have researched applying POMDP machine learning to the task of autonomous driving while having uncertainty over model parameters. The idea of doing my thesis on this subject came after a discussion with my supervisor prof.dr.ir. Martijn Wisse. As a Vehicle Engineer student, I was very interested in autonomous driving vehicles. At the time, my supervisor was looking into new control algorithms very similar to solving POMDPs. It sparked my interest to see what performance could be reached when applying this scheme to estimating parameters of autonomous driving passenger vehicles.

Before I started on my thesis I was unfamiliar with POMDPs and how to solve them. Performing a literature study, implementing and testing a model, and writing a thesis was difficult, but it has allowed me to answer the research question that we identified. Fortunately, my supervisors were always there for all my questions regarding POMDPs and conducting research.

I would like to thank my supervisors prof.dr.ir. Martijn Wisse and dr.ir. Matthijs Spaan for their guidance and support during this process. I would like to thank my fellow students for their support and our daily card game during the break. I would also like to thank my parents, Gerrit and Monique Vroom, my brother, Nino Vroom, and my girlfriend, Brenda Paardekooper, for their patience and understanding.

I hope you enjoy your reading.

Quinn Vroom

Delft, University of Technology
May 28, 2019

Chapter 1

Introduction

In the last decade, there has been increased development for autonomous driving vehicles. With brands as Tesla equipping cars with autopilot since 2014 [1], research is accelerating. There are different aspects that come into play when trying to let a vehicle drive autonomously, such as recognition, prediction and control. For this research we are interested in the control strategy.

An issue with real life systems is knowing the exact value of the parameters. When looking into cars for example, parameter values will change over time. Parameters such as the dimensions of a vehicle will not change, parameters such as mass will fluctuate, but tyre road friction will change heavily over time. Tyre road surface interaction is dependent on multiple factors: tyre size, tyre brand, tyre wear, temperature, weather, road surface. Accounting for all of these factors in dynamic predictions requires a complex model combined with dozens of sensors measuring the conditions.

A non regarded way of finding a control strategy for autonomous passenger vehicles is by regarding the situation as a Partially Observable Markov Decision Process (POMDP). When regarding the situation as a POMDP, uncertainty in state transition and measurement is accounted for. Thus, regarding it as a POMDP makes it suited to deal with uncertain situations. A typical goal of POMDP based vehicle navigation is decreasing uncertainty in the current position estimate of the system [2]. However, more recent studies show that it can also be used to decrease uncertainty in model parameter estimates [3]. By solving the POMDP, an improved control strategy is created. This way a (slightly) deviating path is chosen that improves model parameter inference performance. Hence, the research question becomes:

Is it worthwhile to use the POMDP approach for online parameter estimation of autonomous passenger vehicles?

This question is still very broad, because there are dozens of different algorithms and implementations of the POMDP approach. To answer the research question, we have to cut it into several smaller pieces. The first part is to determine which vehicle model suits an autonomous driving simulation best, followed by establishing the POMDP based algorithm

that is most promising to use for this situation. Luckily, this has already been answered by my literature study. After these two points, the building blocks of the simulation are there. This leads to implementation and testing the algorithms performance, if it performs within realistic bounds, if it can handle high noise and lastly if it suits increasing model complexity. In summary, this produces the following sub-questions:

1. *What is the most suitable vehicle model?*
2. *What is the most promising algorithm?*
3. *Does the algorithm perform on a vehicle model?*
4. *Does the algorithm perform within realistic bounds?*
5. *Does the algorithm suffer from high noise?*
6. *Does the algorithm suit increasing model complexity?*

The goal of this thesis is to find an answer to the main research question by using the answers of these sub-questions. Chapter 2 explains the basic working principle of a POMDP to better understand how to regard the problem as a POMDP. Chapter 3 focuses on modelling vehicle dynamics and answering sub-question 1. Chapter 4 contains the findings of the literature study and explains the chosen algorithm, this will answer sub-question 2. Chapter 5 discusses implementation details and assumptions for the model and algorithm. In chapter 6 we will test the algorithm on a simulation and answer sub-question 3. In chapter 7 we will discuss the effect of bounding the steering input and answer sub-question 4. In chapter 8 we test the effect of increasing noise on performance and answer sub-question 5. Chapter 9 discusses the suitability of the algorithm with regards to increasing model complexity and answers the final sub-question. The 10th and last chapter contains the conclusions and lists the recommendations for future research.

Chapter 2

How does a POMDP work?

In this chapter we will look into POMDPs. The goal of this research is improving parameter estimation performance by using a POMDP based algorithm. Before we can look into different algorithms to solve a POMDP, we need to understand what a POMDP is. More importantly, we need to know how to regard something as a POMDP.

2-1 The abbreviation

POMDPs are classed under machine learning, specifically reinforcement learning. The abbreviation POMDP stands for Partially Observable Markov Decision Process. A POMDP can be solved, which means finding a policy, or control strategy, that minimizes cost or maximizes reward. The total cost is calculated using a predefined function. This cost function can be tuned to accommodate different objectives for the policy to achieve. A POMDP is an extension of the regular Markov Decision Process (MDP) used in machine learning. Similar to an MDP, the dynamics are known, however the underlying state of the system cannot directly be observed by the agent, hence the Partially Observable part [4].

2-2 Structure of the POMDP

A POMDP can be written down as a tuple: $(X, U, Z, T, O, C, \gamma)$. X represents the state space of a system (or robot), U represents the action space of the system and Z represents the observation space of the system. The system starts in a certain state $x \in X$, it performs an action $u \in U$ to reach a new state x', then receives an observation $z \in Z$. The system cannot directly measure its state, but has to make do with the observations. In the tuple, T is a probability function which represents the transition $T(x, u, x') = P(x \mid x, u)$, also called the dynamics of the system. This function describes the probability of reaching a certain new state given the current state and the action, this accounts for things as control uncertainty and changes of the surroundings. O is the observation model given by $S(x', u, z) = P(z \mid x', u)$,

which represents the sensor uncertainty. C is the cost function $C(x, u)$, which represents the cost for the system for taking action u in state x. The last parameter, γ, is the discount factor. γ ensures a finite horizon by making earlier rewards more valuable. The value of γ can be $[0, 1)$. In a block scheme this becomes the model that can be seen in figure 2-1.

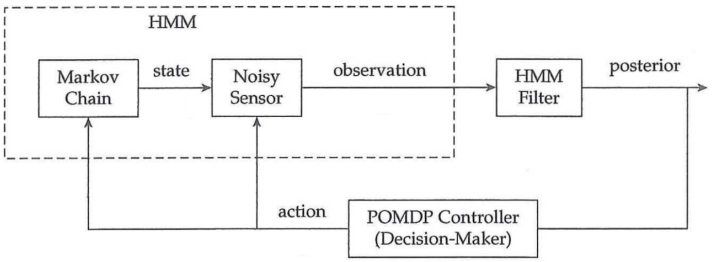

Figure 2-1: Basic POMDP overview [4]

Here, the highlighted section represents the dynamical model combined with the noisy sensors. Out of the dynamics, here Markov Chain, comes the real state x, then the state is measured by the noisy sensor and observation z is received. This is called the Hidden Markov Model, because the model is hidden behind the observation. The so called "HMM Filter" in this diagram maintains the observations over the time and translates it into a belief state $\hat{x}$. Then, using the current belief, the controller decides which action to take according to the precalculated policy. The action u not only affects the dynamics of the system, but also the noisy sensors. The objective is to minimize the expected cumulative cost. For a finite horizon this objective becomes:

$$J_\mu(\pi_0) = \mathbb{E}_\mu \left\{ \sum_{k=0}^{N-1} C(x_k, u_k) + C_N(x_N) \mid \pi_0 \right\} \tag{2-1}$$

Where N is the total time elapsed and π_0 the initial policy. The controller chose the actions using to the policy μ to reach this cumulative cost. The goal of the controller is to find a policy which reaches the objective, which means finding the optimal policy. The optimal policy μ^* is given by:

$$\mu^* = \text{argmin } J_\mu(\pi_0) \tag{2-2}$$

Which holds for any initial prior π_0.

2-3 POMDP based autonomous vehicle

The explanation of a POMDP has been general thus far. To get a better sense of how to regard something as a POMDP and how this relates to autonomous vehicles, the tuple will be explained for a passenger vehicle. The state x contains the current position in the world, for example defined in 2D position coordinates plus a rotational coordinate. The actions u of an autonomous vehicle can contain things like steering input, throttle and brake pressure. The observations z come from the sensors in the vehicle such as GPS for the coordinates and accelerometers for rotation angle. The transition function T is represented

by the dynamics of the vehicle. A vehicle model of choice can be used here. The observation model O describes how accurate a sensor delivers observations. Sensors like GPS output measurements containing noise, while sensors like accelerometers do not directly observe the desired state, causing increased deviation between observation and state. The cost function C can be chosen according to the desired goal, such as minimizing the variance in a parameter estimate. The discount factor γ is less important for autonomous driving with uncertain parameters and can most likely be left at a value of 1. Belief over the state $\hat{x}$ is often done using a Kalman Filter. The dynamical vehicle model can be embedded into the Kalman Filter to maintain an accurate belief from the noisy sensors observations.

Chapter 3

Vehicle model selection

In this chapter we will look into dynamic modelling of passenger vehicles. The goal is to find an answer to the first sub-question: *What is the most suitable vehicle model?*. To answer this, we need to get an insight into different types of vehicle models and tyre road surface interaction models. This is important, because the model choice directly influences which and how many parameters have to be estimated. At the end of the chapter a suitable vehicle model and tyre model are chosen.

3-1 Vehicle model complexity

The field of interest of this research is autonomous driving, so the dynamics of passenger vehicles are the ones that need to be identified. Luckily, a lot of research has already been done for modelling the dynamics of ground vehicles. The shape and size of the model of the vehicle can be chosen in different ways. Dependent on the task, the level of complication can be varied [5]. Choosing a more complex model will not automatically result in better performance. More complex means more parameters to infer and/or tune. For this research a small and computationally inexpensive model is preferred. The sooner the algorithm can guide towards an accurate estimate of the model parameter(s), the better it will perform overall. In the following subsections two different ways to model a vehicle are introduced and briefly explained, also considering their respective practical applications.

3-1-1 Bicycle model

A very simple and effective way to model a 4 wheel passenger vehicle is the so called "bicycle model", which is widely used in theoretical and practical studies involving passenger vehicles [5] [6] [7] [8]. The bicycle model consists of a single front wheel and a single rear wheel, just like a bicycle. It is a top-down view of the bicycle, so it is only allowed to move in x- and y-direction. The bike cannot fall over or lean into a corner and there is also no weight transfer in longitudinal direction. Normally, for the tyre dynamics a linearized model is chosen, but

this can be interchanged by a more sophisticated one, more on this in section 3-2. How the bicycle model looks, can be seen in figure 3-1 below.

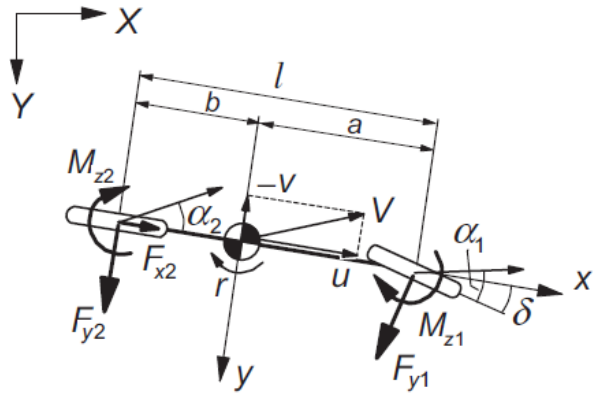

Figure 3-1: Bicycle model of a passenger vehicle [9]

To derive the 2 DOF equations of motion for the bicycle model, the longitudinal velocity is considered to be constant and the lateral component of the longitudinal front wheel force F_{x1} is neglected. This leads to the following differential equations [9]:

$$\dot{v} = -\Big(\frac{C_1 + C_2}{mu}\Big)v + \Big(-u + \frac{bC_2 - aC_1}{mu}\Big)r + \frac{C_1}{m}\delta \tag{3-1}$$

$$\dot{r} = \Big(\frac{bC_2 - aC_1}{I_z u}\Big)v + \Big(-\frac{b^2C_2 - a^2C_1}{I_z u}\Big)r + \Big(\frac{aC_1}{I_z}\Big)\delta \tag{3-2}$$

Where C_1 and C_2 are parameters of the linearized tyre model. The A and B matrix, of a linearized dynamical system, can easily be extracted from these two equations and used for simulation. Even though this model is heavily simplified, the bicycle model is very effective in practice [5] [6] [7] [8].

3-1-2 8 DOF vehicle model

A more complicated way of modelling a passenger vehicle is also an option. Here we look at the 8 DOF example from Smith et al. [10]. In this case, the track width is considered, which means that the vehicle is modelled with four wheels, instead of just two like in the bicycle model. Forward velocity and wheel dynamics are also considered to be variables in this model. Furthermore, lateral and longitudinal weight transfer, roll and pitch motion respectively, are added to the model. Lastly, engine rotational speed en driving torque are considered in this model. How this model looks can be seen in figure 3-2 below.

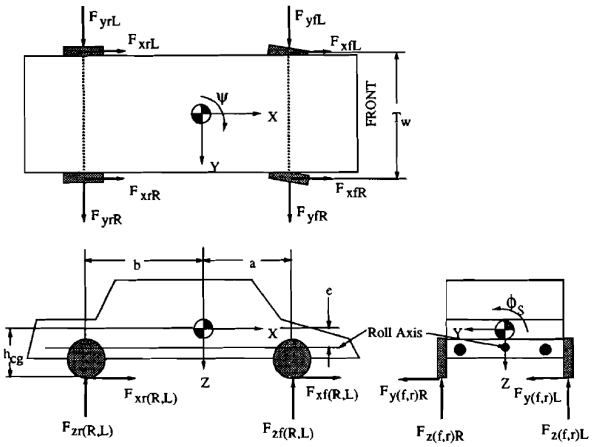

Figure 3-2: 8 DOF model of a passenger vehicle [10]

The corresponding equations of motion can be found in Smith et al. [10]. This more complex model will produce more accurate and realistic results than the bicycle model [10]. The main difference appears when looking at high-g maneuvers. Here the dynamical behaviour becomes more and more nonlinear and harder to predict with simplified models. For normal driving conditions however, a more complex models performs similar, but does require more parameters to be specified for the model to work properly. This is often easier said than done. Here it becomes obvious why practical applications involving normal autonomous driving tend to apply simple models like the bicycle model [5] [6] [7] [8].

3-2 Tyre model

Next, we will discuss different ways of modelling tyre road friction dynamics. As explained before, a fast and small model is preferred. The sooner the estimate is correct, the better. Apart from choosing the level of complexity of the vehicle model, the fitting tyre dynamics also have to be chosen. Of course, the most simple way to do this is by ignoring slip all together. This leads to the model becoming more of a kinematic model than a dynamic model. This is not relevant, as the goal is to find performance on a dynamic model, so this is not considered in the following subsections. There will be two different tyre models introduced and briefly explained, also considering their usual respective practical applications.

3-2-1 Linear tyre model

A very simple way to model the tyre behaviour under slip, is using a linearized tyre model. The characteristics of a tyre can be seen in figure 3-3. On the left side of the figure is the tyre behaviour in lateral direction and on the right side is the behaviour in longitudinal direction. For the bicycle model, only the lateral behaviour is considered. The sideslip angle α represents the angle between the direction the tyre is pointed and the direction the tyre is moving in. In case of the front wheel(s), the steering angle also influences the sideslip angle α. Looking at the graph, in this example there is a maximum side force F_y at a sideslip angle of around 8°. After this sideslip angle, the lateral force decreases again. The linearized tyre model considers

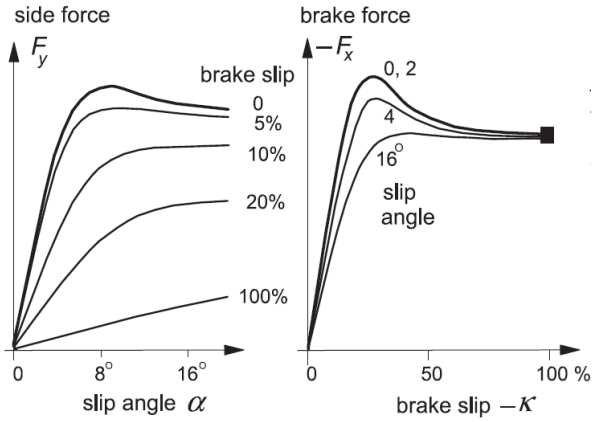

Figure 3-3: Typical tyre characteristics, showing the lateral steering force and the longitudinal brake force against the tyre slip under normal conditions [9]

calm, highway driving scenario's and thus simplifies the sideslip curve to just the first part. Before the maximum, the curve is very close to linear. For the longitudinal behaviour, the same principle holds. The definition of slip in this case is a little different though. Slip is based on the difference in rotational wheel speed and forward wheel speed. A slip κ of 0 means that the wheels rotation is equivalent to its forward movement, a slip κ of 1 means that the wheel is fully rotating, but not moving forward at all. Depending on the slip κ, a different longitudinal force F_x is exerted on the ground. As with the sideslip α, the graph of the slip κ shows that before the maximum, the graph is close to linear and after the maximum it decreases gradually. The linearized tyre models thus take the form of:

$$F_y = C_\alpha \cdot \alpha \tag{3-3}$$

$$F_x = C_\kappa \cdot \kappa \tag{3-4}$$

Where C_α and C_κ are constants, matching the gradient of the beginning of the non-linear curves.

3-2-2 Magic Formula

The linear tyre model is a very simplified way of modelling the tyre behaviour. A big downfall of this method is the fact that the optimal point, at which the force exerted on the ground is the highest, is not included. So when looking at more dynamic driving scenario's the optimal slip cannot be assessed with the linearized tyre model. A very sophisticated example of a nonlinear tyre model is the Magic Formula [9]. The Magic Formula has seen many iterations over the years, but is based on relatively simple equations. The Magic Formula is stated as follows:

$$y = D\sin[C\arctan\{Bx - E(Bx - \arctan Bx)\}] \tag{3-5}$$

Where B, C and D are tuning parameters for the exact shape of the curve. The Magic Formula has the same basic shape as the curves in figure 3-3. The function can be used to generate an output Y that represents either F_x or F_y, or even other variables. The exact values of the parameters of the Magic Formula are based on empirical data. Different tyres have been tested under different conditions to get an insight into the behaviour and estimate the

necessary parameters. The Magic Formula is a very accurate model and also a fairly quick one due to the simplicity. However, to know the exact value of the parameters, extensive knowledge and testing of the used tyre is needed. If the parameters are not known exactly, the benefit over the linearized tyre model becomes less compelling.

3-3 Conclusion

From the reviewed methods in the mentioned papers it becomes clear, that the bicycle model is the most appropriate one for this research. Due to its simplicity, it is computationally inexpensive and has a fairly accurate prediction. The same principles hold for the tyre model. The appropriate tyre model corresponding to the bicycle model is the linearized one. Under normal driving conditions, this will lead to fairly accurate results. Choosing for method like the Magic Formula generates an increase in parameters and will thus be more computationally expensive and bigger in size. Being able to update the tyre parameter estimation more quickly shows more potential for rapidly changing road conditions. Returning to the sub-question, using the bicycle model together with the linearized tyre model is most suitable for this research.

Chapter 4

Algorithm selection

In this chapter we will look into POMDP algorithms. The goal is to find an answer to the second sub-question: *What is the most promising algorithm?*. This chapter consists of two parts. First, we describe how we chose the most promising algorithm for solving our POMDP. This chapter appeared earlier as my Literature Thesis [11]. The second part focuses on the working principle and the most important equations of this algorithm.

4-1 What is the most promising algorithm?

To identify an algorithm that suits the goal of estimating parameters online of an autonomous driving vehicle, literature has been reviewed. There are a couple of requirements belonging to this specific task, which lead to criteria on which the algorithms can be compared. The criteria are briefly described below, followed by a summary of the comparison of the algorithms and a conclusion on the most promising one.

4-1-1 Requirements and criteria

First of all, the algorithm should be able to handle a continuous state space and observation space. The action space would preferably also be continuous, but discrete is also manageable. The steering input for example is bounded and can be approximated efficiently by discrete steps in the angle. For the current state, in the state space as well as the observational space, it is undesirable to define boundaries. Discretizing all of the possible states is not suitable and very computationally demanding.
The next part is about the manner in which the POMDP is solved and a policy is obtained. The task of online parameter estimation demands a policy that can be solved for online. When the parameter estimate suddenly needs to change, a corresponding policy has to be ready.
Another criterion is about the parameters that need to be estimated. Firstly, the amount of parameters. When estimating the lateral tyre stiffness, this means that it has to estimate

this for four tyres individually. Considering the bicycle model, this naturally reduces to two tyres. The amount of parameters should thus be more than one, preferably four or more. The initial estimate of the parameter is also an important factor. Tyre road interaction can be very different than expected. For example when the temperature has changed rapidly from one point to another. The deviation between the initial estimate and the true value can differ a lot, thus the algorithm should handle big deviations well.
The last criterion is the experiments the algorithm has already been successfully tested on. When the algorithm performs well on a dynamical system closely resembling a passenger vehicle, it is more likely to perform well for the desired task. This of course is not a must, but it adds to how promising an algorithm is.

4-1-2 Comparison and choice

From the literature study, three different algorithms were received. The three different papers were compared using the requirements and criteria above, which led to the following results.

	Continuous space	Solve for policy online	Amount of parameters to infer	Initial estimate deviation	Experiments
Webb et al. [3]	yes	yes	multiple	large	double integrator, differential drive, double pendulum
Bai et al. [12]	partially, action discrete	no	single	small, predescribed	double pendulum
Slade et al. [13]	yes	yes	single	large	double integrator, planar manipulation

Table 4-1: Three promising POMDP algorithms compared by requirements and criteria

Looking at the three different algorithms in table 4-1 and comparing them to the requirements and criteria, it becomes clear that the one from Webb et al. [3] is the most promising to use. This algorithm checks all the boxes, from continuous spaces to initial estimate deviation. On top of that, the experiments it is tested on are very close in resemblance the autonomous driving passenger vehicle task. Looking at the performance of the algorithm, in the paper it showed a lot of parameter inference potential. It was tested on three different experiments with different levels of complexity. Returning to the sub-question, the algorithm from Webb et al. [3] is the most promising choice to implement on our vehicle model.

4-2 How does the algorithm work?

As discussed before, using the literature research we have established that the algorithm in Webb et al. [3] shows promising result for autonomous driving passenger vehicles. In order to understand this algorithm and the results it produces after implementation, we have to clarify

the working principles of the method. This way, we can draw well-motivated conclusion from the results of the experiments. The algorithm is more thoroughly explained in Van den Berg et al. [14] and Van den Berg et al. [15]. First, the original algorithm will be explained, and then the modifications done in Webb et al. [3] that assist in parameter estimation. Note that this is only a brief explanation of the algorithm, the complete set of equations can be found in appendix A.

4-2-1 System and Kalman Filter

The algorithm starts off with the dynamics of the system. The system is defined in the following matter.

$$x_{t+1} = f[x_t, u_t] + m, \quad m \sim \mathcal{N}[0, M[x_t, u_t]] \tag{4-1}$$

$$z_t = h[x_t] + n, \quad n \sim \mathcal{N}[0, N[x_t]] \tag{4-2}$$

Where x_{t+1} represents the discretized system dynamics and z_t represents the sensor data, which is based on the current state. Note that both the system and the sensors are influenced by noise, which is assumed to be of Gaussian nature.

On top of the system, there is a Kalman Filter present. The job of the Kalman Filter is to maintain a belief over the current state, based on the sensor data z_t and an internal model of the system. The equations of the Extended Kalman Filter are stated below.

$$\hat{x}_{t+1} = f[\hat{x}_t, u_t] + K_t(z_{t+1} - h[f[\hat{x}_t, u_t]]) \tag{4-3}$$

$$\Sigma_{t+1} = \Gamma_t - K_t H_t \Gamma_t \tag{4-4}$$

where

$$\Gamma_t = A_t \Sigma_t A_t^T + M[\hat{x}_t, u_t] \tag{4-5}$$

$$K_t = \Gamma_t H_t^T (H_t \Gamma_t H_t^T + N[\hat{x}_t])^{-1} \tag{4-6}$$

$$A_t = \frac{\partial f}{\partial x}, \quad H_t = \frac{\partial h}{\partial x}[f[\hat{x}_t, u_t]] \tag{4-7}$$

Where $\hat{x}$ and Σ represent the state estimation and its variance respectively. Note that there is assumed to be no noise on the control input u when passed on to the Extended Kalman Filter.

4-2-2 Value iteration

The goal of solving the POMDP is finding the optimal policy. In our case case this policy will lead to the fastest tyre parameter inference performance. The effectiveness of a policy is defined by the predefined cost function, c_t. The designer chooses the shape of the cost function and tunes the parameters, in order to accommodate for the desired goals. The optimal policy is the one that has the least cost over the whole trajectory. This means minimizing the following equation.

$$E_{z_1,\ldots,z_l}[c_l[\mathcal{X}_l] + \sum_{t=0}^{l-1} c_t[\mathcal{X}_t, u_t]] \tag{4-8}$$

Where $\mathcal{X}_t$ represents the belief at time t. The difference between $\hat{x}_t$ and $\mathcal{X}_t$ is that the latter represents the whole belief: state *and* uncertainty. A general approach is using value iteration. A value functions is different than a cost function. A value function not only accounts for the current cost, but also for (a part of) the expected future cost. The value is calculated for each step of the trajectory using backwards recursion.

$$v_l[\mathcal{X}] = c_l[\mathcal{X}] \tag{4-9}$$

$$v_t[\mathcal{X}] = \min_u(c_t[\mathcal{X}, u] + \underset{z}{E}[v_{t+1}[\beta[\mathcal{X}, u, z]]]) \tag{4-10}$$

$$\pi_t[\mathcal{X}] = \arg\min_u(c_t[\mathcal{X}, u] + \underset{z}{E}[v_{t+1}[\beta[\mathcal{X}, u, z]]]) \tag{4-11}$$

Where β represents the discrete belief dynamics and π_t represents the optimal policy/control at time t. By working backwards, the value function of the next time is already available for the current time step. This allows the POMDP to be solved.

4-2-3 Algorithm

To apply the method of value iteration to the system, there has to be a model for the belief dynamics. The belief is updated via the Extended Kalman Filter, as discussed above. The belief is stochastic, due to the fact that the measurement z cannot be known in advance. Assuming the stochastic belief dynamics can modelled as a Gaussian leads to the following simplified equations.

$$\hat{x}_{t+1} = f[\hat{x}, u] + w, \quad w \sim \mathcal{N}[0, W[\hat{x}_t, \Sigma_t, u_t]] \tag{4-12}$$

$$\Sigma_{t+1} = \Phi[\hat{x}_t, \Sigma_t, u_t] \tag{4-13}$$

where

$$W[\hat{x}_t, \Sigma_t, u_t] = K_t H_t \Gamma_t \tag{4-14}$$

$$\Phi[\hat{x}_t, \Sigma_t, u_t] = \Gamma_t - K_t H_t \Gamma_t \tag{4-15}$$

Note that these are based on the belief the Kalman Filter holds over the time. The method requires a nominal trajectory defined as a series of beliefs and control inputs.

$$traj_{nominal} = (\bar{x}_0, \bar{\Sigma}_0, \bar{u}_0, ..., \bar{x}_l, \bar{\Sigma}_l, \bar{u}_l) \tag{4-16}$$

The algorithm uses an iterative approach based on linearized approximations and cost functions about the nominal trajectory that converges to a locally optimal control policy:

$$\mathbf{u} = L_t(\hat{\mathbf{x}} - \bar{\mathbf{x}}_t) + \mathbf{l_t} + \bar{\mathbf{u}}_\mathbf{t} \tag{4-17}$$

See appendix A for a complete derivation from nominal trajectory to control policy.

4-2-4 Reaching convergence

The algorithm works in multiple steps. First, there has to be a nominal trajectory for the system. This nominal trajectory will be improved by the algorithm in a couple of steps. Using backwards recursion the two input correcting matrices L_t and l_t are calculated for each

timestep, from $t = L$ until $t = 0$. Then forward integration, from $t = 0$ until $t = L$, of the deterministic dynamics is used to form a new trajectory. Simultaneously, the total cost of this new trajectory is calculated and compared to the previous trajectory. If the total cost is not lowered, the same forward integration is performed where l_t is cut in half. The correcting factor l_t will always point towards the (local) optimum, but the magnitude can be too great. This causes overshoot and will result in possibly worse performance. To counter overshoot in this case, the correcting factor is made smaller in order to not overshoot. When a lower cost is found, the whole process will run again. As said before, l_t shows in which direction the input u should be corrected. This also means that it will decrease when the current trajectory is closer to an (sub)optimal one. Convergence can thus be seen in the magnitude of l_t [14] [15].

4-2-5 Adding parameter estimation

The algorithm is designed to minimize the cost function by altering the input. However, it can also be used for online parameter estimation as shown in Webb et al. [3]. The trick to do this, is by changing two simple things. First, the state is augmented with the parameter. The discretized dynamics of the parameter(s) k are defined below.

$$k_{t+1} = k_t + \theta, \quad \theta \sim \mathcal{N}[0, \Theta] \tag{4-18}$$

Adding this to the discretized dynamics of the system x will lead to the augmented state y.

$$y = \begin{bmatrix} x \\ k \end{bmatrix} \tag{4-19}$$

This means that the dynamical model of the system is increased by including the parameter into the state. The second thing to change is the cost function. The required value of k is of course unknown, but high variance of k can be punished. After changing these things, the algorithm will now not only reach the requested state, but also try to minimize the uncertainty in its parameter estimation. Thus, when performing a task it will try to estimate the true value to minimize cost. The cost function will look like equation 4-20 and 4-21 below.

$$c_T(\hat{\mathbf{y}}, \Sigma) = (\hat{\mathbf{x}} - \mathbf{x}^*)^T Q_T (\hat{\mathbf{x}} - \mathbf{x}^*) + \operatorname{tr}(S_T \Sigma_{\mathrm{x}}) + \operatorname{tr}(T_T \Sigma_{\mathrm{k}}) \tag{4-20}$$

$$c_t(\hat{\mathbf{y}}, \Sigma, \mathbf{u}) = (\hat{\mathbf{x}} - \mathbf{x}^*)^T Q_t (\hat{\mathbf{x}} - \mathbf{x}^*) + \mathbf{u}^T R_t \mathbf{u} + \operatorname{tr}(S_t \Sigma_{\mathrm{x}}) + \operatorname{tr}(T_t \Sigma_{\mathrm{k}}) \tag{4-21}$$

4-3 Conclusion

Returning to the sub-question, *What is the most promising algorithm?*. From the reviewed literature it became clear that the algorithm from Webb et al. [3] was the most promising one. It was the most suitable one, due to the estimation of multiple parameters and the comparable experiments it was tested on. Knowing this, a brief insight into the equations of the algorithm has been shown. The parameter estimation has been done by including the parameter into the state and thus augmenting it. In the cost function a term is used to punish the uncertainty over the parameter estimate, to reward inference behaviour.

Chapter 5

MATLAB implementation

In the previous chapter, the mathematics of the algorithm have been explained. In this chapter the implementation into Matlab is discussed. There are a couple of things that are changed, simplified and/or added to make it perform optimally. First, the MATLAB code is discussed, then ensuring convergence of the algorithm, followed by discretization of the system dynamics and lastly the tune-ability of the parameters.

5-1 MATLAB code

The implemented simulation in MATLAB consists of multiple parts. It starts off with running *sim_matrix_preparation.m*. Here the matrices F_t until Z_t are calculated in symbolics, see appendix A. The symbolic matrices are necessary, because the algorithm consists of several linearized matrices that need to be calculated at each timestep. The m-file uses the dynamic equations of the system, the corresponding parameter values, the timestep dt and the cost function shape. All the other things are adjustable without re-running the file. Once this m-file has generated *Calc_F_till_Z.m* it is time for the next step. The following step is running *main_sim.m* to generate an initial trajectory. By inputting the parameters of the cost function, parameters of the system, parameters of the noise and a control input a trajectory is created. The trajectory is stored together with the cumulative cost. The initial trajectory will form the starting point for the algorithm. The algorithm optimizes the trajectory iteratively, according to the cost function. This step is only needed once per simulation.

The preparation of the simulation is now finished. The main script starts off with loading the initial trajectory and defining all tune-able parameters. The main script contains two loops. The first loop optimizes the trajectory by running *obtain_new_policy.m* until convergence, while the second loop performs the new trajectory on the simulation and plots the results. *obtain_new_policy.m* also contains two loops. The first loop performs backwards recursion to find L_t and l_t by calling *control_input.m*. The second loop performs forward integration on the deterministic dynamics, by applying the control calculated using L_t and l_t. If cost is not decreased, there is assumed to be overshoot. l_t is cut in half and the forward integration is

re-run until cost is decreased. In *control_input.m* L_t and l_t are calculated using the linearized matrices from *Calc_F_till_Z.m*.

In pseudocode this becomes:

Preparation:
sim_matrix_preparation.m
main_sim.m (initial trajectory)

Main script:
Loop:
  obtain_new_policy.m:
    Loop:
      control_input.m: (backwards recursion)
        calc_F_till_Z.m
        calculate L_t and l_t
    Loop:
      forward integration
      if cost not decreased:
        l_t in half and repeat
    if cost and l_t not converged:
      repeat loop
Loop:
  run new policy on simulation

5-2 Ensuring convergence

As discussed previously, the algorithm will show convergence, when l_t goes to zero or under a predefined threshold. Note that l_t is a 2D matrix, where the columns represents the time and the rows (n) the correction on each input. To get an insight of the size of it, the magnitude of the $n \times 1$ vector is calculated for each timestep and these are all added up for the whole time range. When working with the algorithm, it became apparent that this total magnitude will converge. However, with saturated steering input, as will be explained in chapter 7, it did not show convergence towards zero. To cease the algorithm when convergence has been reached, the following method has been implemented. It will look at the last three time-steps, and if the change in total magnitude is less then 1% for all three, it will finish. As a safety measure it will also cease if the total cost difference over the last three iterations is less than 1%. This implementation can be found in the second loop of *obtain_new_policy.m*.

5-3 Discretization of vehicle dynamics

For the vehicle, a linear bicycle model is most appropriate. The model has to be discretized for it to work with the POMDP algorithm. The simplest way to perform discretization is with Euler's method. A more refined and accurate discretization would be achieved with Runge-Kutta 4. This is also what Webb et al. [3] suggests. Discretization using Euler is done

in the following fashion.

$$x_{t+1} = x_t + f(t, x) \cdot \Delta t \tag{5-1}$$

While discretizing using Runge-Kutta 4 is a bit more complicated, as shown below.

$$k_1 = h \cdot f(t, x) \tag{5-2}$$
$$k_2 = h \cdot f(t + h/2, x + k_1/2) \tag{5-3}$$
$$k_3 = h \cdot f(t + h/2, x + k_2/2) \tag{5-4}$$
$$k_4 = h \cdot f(t + h, x + k_3) \tag{5-5}$$
$$x_{t+1} = x_t + 1/6 \cdot (k_1 + 2k_2 + 2k_3 + k_4) \tag{5-6}$$

Where $f(t, x)$ represents the differential equation(s) of the system dynamics and h the time step size of the simulation. In case of our dynamics, the equation is written like $f(x, u)$, because time is not directly visible in the differential equation. As can be seen, the accurateness of both models is highly dependent on the timestep. The smaller the timestep, the more accurate results it will output. While Runge-Kutta 4 is generally more accurate, Euler is a lot smaller and computationally inexpensive.

5-3-1 Comparing results

To compare performance of these different discretization methods, the linear bicycle dynamics have been discretized in two manners. Firstly using Euler's method and secondly using a Runge-Kutta 4th order method.
The vehicle dynamics of choice are given in equation 3-1 and 3-2. These equations represent $f(t, x)$ in the discretization equations. Looking back at the course Vehicle Dynamics A [16] for parameters corresponding to an average passenger vehicle gives the following table.

a	1.1	m
b	1.60	m
m	1600	kg
I_z	2100	kgm^2
C_1	2· 57000	N/rad
C_2	2· 47000	N/rad
u	40	km/h
Δt	0.01	s

Table 5-1: Passenger vehicle simulation parameters

Let's compare performance. As discussed before, for each method a directly identical simulation has been run, without any noise in the system. For the (steering) input a sinusoidal wave with an amplitude of 30° has been used. The lateral velocity v and the yaw rate r have been plotted in figure 5-1. Next to this, the absolute error between the two methods has been plotted for both states, v and r. When looking at the results, the sinusoidal shape of the input signal shines through. The smallest error seems to appear around the peaks of the lateral velocity and yaw rate, thus when the steering input changes its rotational direction. However, looking the size of the error it seems to be in the same magnitude as typical sensor noise [17]. Mean and variance of the error are given in table 5-2.

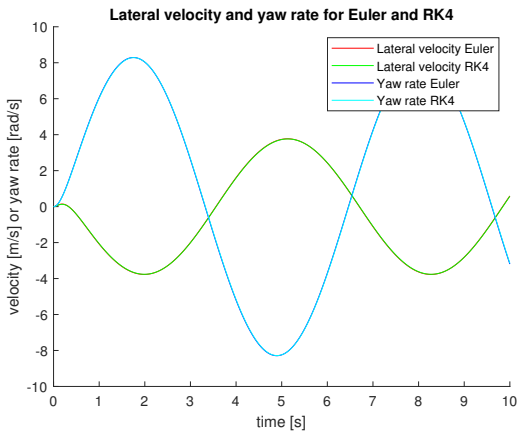

Figure 5-1: State values during simulation

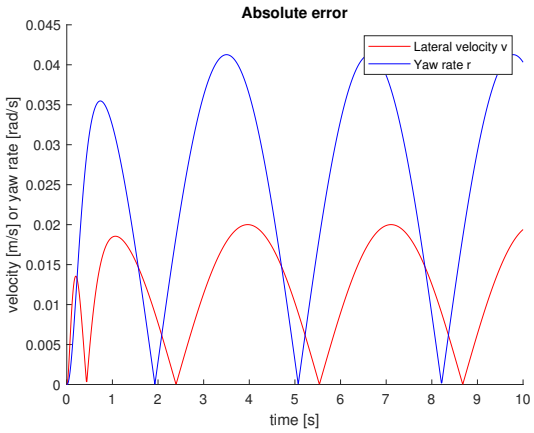

Figure 5-2: Absolute error between both methods

	mean	variance
Lateral velocity v error $[m/s]$	$1.22 \cdot 10^{-2}$	$3.67 \cdot 10^{-5}$
Yaw rate r error $[rad/s]$	$2.56 \cdot 10^{-2}$	$1.62 \cdot 10^{-4}$

Table 5-2: Error between Euler and Runge-Kutta 4

5-3-2 Consideration

As discussed previously, before the simulation can run the algorithm matrices F_t until Z_t have to be calculated in symbolics. When trying to calculate these matrices from discretized vehicle dynamics, these matrices become significantly large. The largest matrices have the size n^4, where n represents the amount of states in the state vector. To counter this effect, a small discretization method is required, more on this in chapter 9. The suggested method, Runge-Kutta 4, is too large to run *sim_matrix_preparation.m* on the hardware used. Euler however is significantly smaller. Therefore the error between Euler and RK4 prediction is tested.
Passenger vehicles have built-in sensors to estimate the lateral velocity and yaw rate. These sensors have a certain error, when trying to predict and/or measure the velocity of the system. Using GPS or integrating acceleration sensor output gives an error of 0.05 m/s [17]. Table 5-2 shows a lateral acceleration error of 0.01 m/s. Assuming the prediction error of RK4 << the prediction error of Euler, means the total prediction error of Euler $\equiv$ the error between Euler and RK4. From this we can conclude that using Euler instead of Runge-Kutta 4 for discretization of the bicycle model is acceptable under these conditions. Runge-Kutta 4 is more accurate, but the error between both methods is in the same order of magnitude as sensor noise. Of course, when the calculation time would be less substantial, Runge-Kutta 4 would be the preferred method.

5-4 Tune-able parameters

As discussed previously, the preparation of the simulation is a computationally demanding task. In chapter 9 there is a more in-depth look into the root of this problem. The matrices F_t until Z_t have to be calculated in symbolics in MATLAB, to be used in the simulation. To assist in decreasing the computation time, as much symbolics have been eliminated as possible. All unused parameters have been made "0" in diagonal matrices and the constant parameters of the vehicle model have been substituted by its value beforehand. Even the parameter estimation has been limited to a single varying parameter, the tyre stiffness of the front tyre. The result is a full simulation taking less than 5 minutes and a preparation taking around 2 hours. The downside however, is not being able to adjust every parameter in between simulations quickly. The noise matrices and cost function parameters are tune-able between simulations though.

Chapter 6

Testing performance

In this chapter we will look into testing the algorithm. The goal is to find an answer to the third sub-question: *Does the algorithm perform on a vehicle model?*. To answer this, we need to try the POMDP algorithm on realistic driving scenarios. The algorithm starts off with an initial trajectory and optimizes it using the cost function. The performance of the algorithm is measured by the parameter estimate accurateness and variance over the simulation time. The target is finding a policy that makes the EKF infer the true value of the parameter quicker than with the initial trajectory. For the experiments the parameter values from table 5-1 are used. The simulation time is 10 seconds.

6-1 Sine input experiment

To answer the third sub-question, we develop a driving scenario and analyze how the algorithm improves upon it. It is a scenario where the steering input of the initial trajectory is a sine with an amplitude of 30°. The initial guess of parameter C_1 is $1.94 \cdot 10^5$ N/rad, while the true value is $1.14 \cdot 10^5$ N/rad. Performing a slalom steering input will result in an improved parameter estimation in the initial trajectory since it already gains insight into the handling dynamics. The repeating pattern is also interesting for testing the influence of the initial trajectory on the optimized one. The results of this experiment are on the next page.

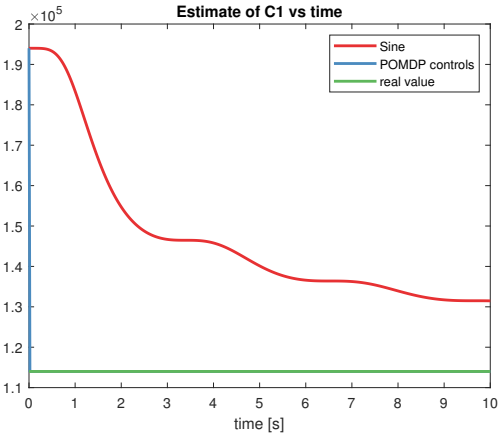

Figure 6-1: Estimate of C_1 $[N/rad]$

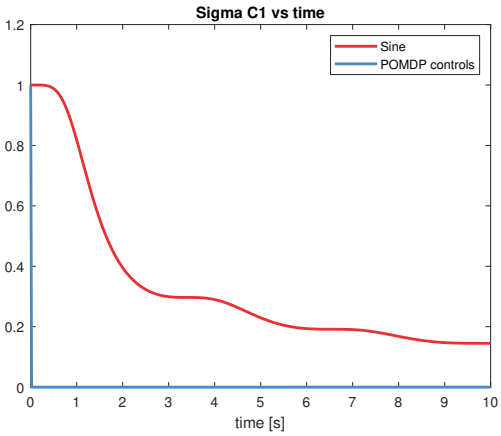

Figure 6-2: Variance of C_1 estimate

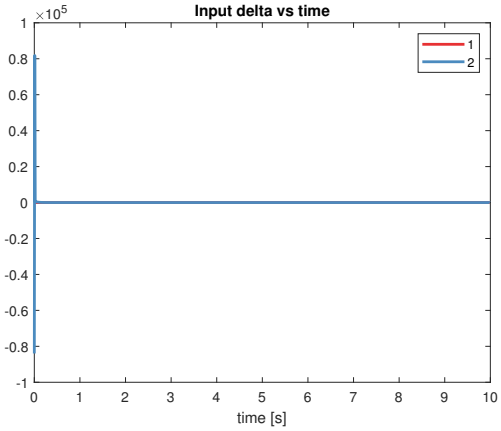

Figure 6-3: Steering input δ $[rad]$

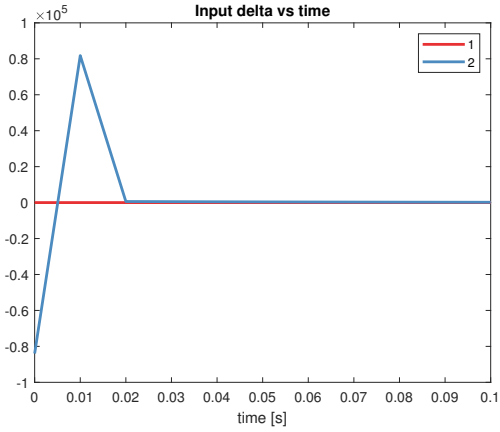

Figure 6-4: Zoomed in steering input δ $[rad]$

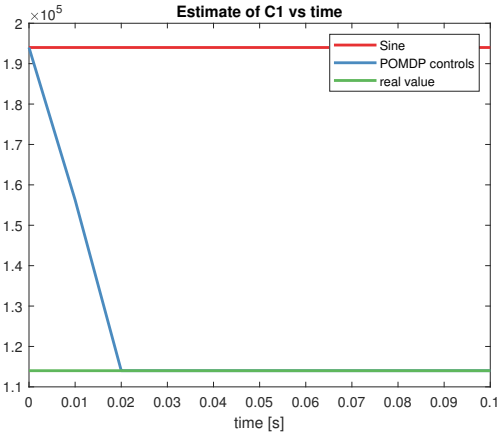

Figure 6-5: Zoomed in estimate of C_1 $[N/rad]$

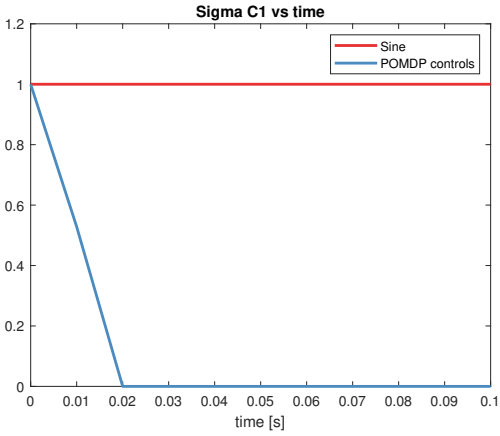

Figure 6-6: Zoomed in variance of C_1 estimate

The first three plots show the estimate of the parameter C_1, the variance of C_1 and the steering input δ for the whole simulation time. The other plots show the same results, but then only for the first 0.1 seconds of the simulation. The red line represents the original trajectory and the blue line shows the improved, POMDP based trajectory. Looking at the results, performance is almost perfect. Within 2 timesteps, or 0.02 seconds, the parameter estimate has converged to the correct value. The algorithm gives the input a peak upwards and downwards and then settles towards zero. The only problem with this improved steering input is the unrealistically high steering angle of $8 \cdot 10^4$ *rad*. Due to the linearity of the tyre model, it is accompanied by extremely high lateral acceleration and yaw rate. This means that this kind of dynamic behaviour would not be possible in real life. Steering beyond 90 degrees doesn't make any sense then, but here it is proportionally connected. In this simulation increasing steering simply results in increased lateral and rotational acceleration. It shows that the algorithm can guide the EKF to perform parameter inference very quickly, however the EKF probably needs unrealistically high values for it. Returning to the sub-question, the results of this experiment show that the algorithm does perform on a vehicle model. Knowing this, we can look into how it performs within realistic steering bounds.

Chapter 7

Applying steering bounds

In this chapter we will continue testing the algorithm. The goal is to find an answer to the fourth sub-question: *Does the algorithm perform within realistic bounds?*. In the previous chapter we saw that the algorithm applied an unrealistically high steering input on the system. Now we would like to see how performance is within realistic steering input bounds. First, we will introduce a bound to the steering input and then we will test the algorithm again on the same experiment.

7-1 Saturating the steering input

The algorithm will calculate a new input for each timestep of the simulation. This input should result in a lower cost and thus be better performance. The downside of this method, as can be seen in the previous chapter, is that the magnitude of the required input is not within the boundaries of reality. In other experiments as well, it has corrects the steering input to values way beyond feasible in reality. To counter this effect, Webb et al. [3] suggests saturating the maximum value of the input. Even though there is no direct feedback to the algorithm that it has been saturated, it will still convergence. This is also what we have applied here. The saturation limit has been set to a 50° steering angle of the wheels, which is already quite optimistic considering an average passenger vehicle. Note that there is no steering ratio between the steering input and the angle of the front wheels.

7-2 Sine input experiment

To answer the fourth sub-question we use the same initial trajectory as before and analyze how the algorithm improves upon it. Again, the initial guess of parameter C_1 is $1.94 \cdot 10^5$ N/rad, while the true value is $1.14 \cdot 10^5$ N/rad. The results of the experiment are below.

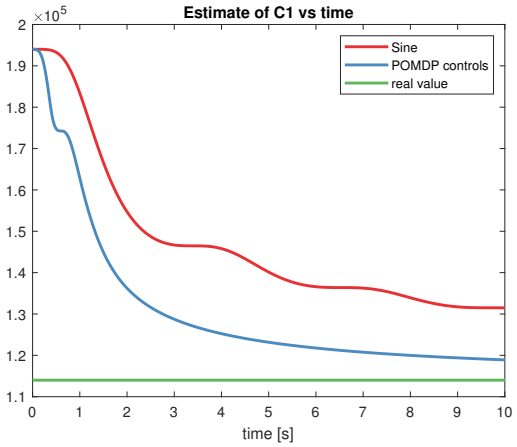

Figure 7-1: Estimate of C_1 $[N/rad]$

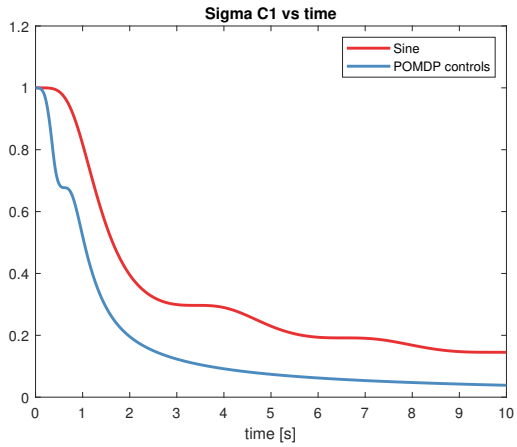

Figure 7-2: Variance of C_1 estimate

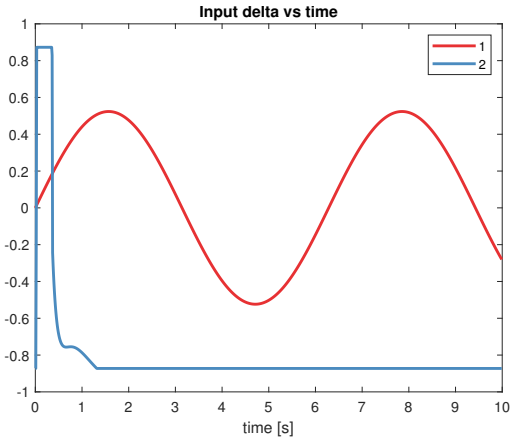

Figure 7-3: Steering input δ $[rad]$

Looking at the results, performance is a lot different than without steering saturation. The new trajectory still leads to improved parameter inference performance. The parameter estimate goes a lot steeper down towards the true value than with the initial trajectory. Due to the heavily changed steering input, the estimate keeps improving until the end of the simulation. As can be seen, it does not manage to reach convergence within the 10 seconds of simulation time. The shape of the variance is very similar to the parameter estimate itself, which is logical. An improved estimate should lead to a decreased variance in that estimate. Interestingly, the steering input fluctuates for little over a second and then just stays at the saturation limit. The results of the experiment show that the algorithm does perform and is able to improve parameter estimation performance. However, bounding the steering input leads to a significantly retarded convergence of the estimate towards the true value of the parameter. It strengthens the suspicion from the previous experiment, that the EKF needs high input values to be excited into fast converging parameter inference. Returning to the sub-question, the results show that the algorithm performs within realistic bounds, but it is not great. Comparing them to the previous results shows that saturating the input severely limits the parameter inference performance.

Chapter 8

Testing noise influence

In this chapter we will test the algorithm on sensor noise. The goal is to find an answer to the fifth sub-question: *Does the algorithm suffer from high noise?*. For real life driving the algorithm has to handle noise well, especially noise in the same order of magnitude as real sensor noise on lateral velocity and yaw rate. We will test the algorithm using the same experiment yet applying different levels of noise.

8-1 Sine input experiment

To answer this sub-question we use the same initial trajectory as before and vary the sensor noise. Then we can analyze how the algorithms performance is affected by it. Again, the initial guess of parameter C_1 is $1.94 \cdot 10^5$ N/rad, while the true value is $1.14 \cdot 10^5$ N/rad. As discussed before, all noise is considered Gaussian. The diagonal of the noise matrix N has been altered for this experiment. For all three states v, r and C_1 the same Gaussian noise has been applied to the system and the EKF. First a variance of 10^{-6} was tested, then a variance of 10^{-4} and lastly a variance of 10^{-2}. The variance of v is in m/s, the variance of r is in rad/s and the variance of C_1 is in N/rad. The results of the experiments are below.

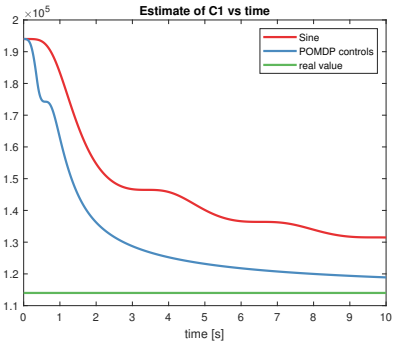

Figure 8-1: Sensor noise variance of 10^{-6}

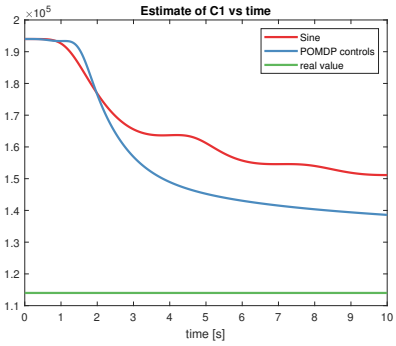

Figure 8-2: Sensor noise variance of 10^{-4}

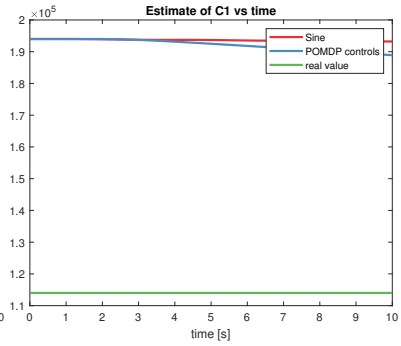

Figure 8-3: Sensor noise variance of 10^{-2}

Looking at the results, performance is affected by increasing the measurement/sensor noise of the system. The plot on the left shows the same performance as in the previous experiment. The noise is increased in the plot in the middle (10^{-4}) and even more increased in the plot on the right (10^{-2}). For the algorithm to work properly with the vehicle model, the noise matrices for measurement noise in the simulation and the algorithm, have to be very small in magnitude (10^{-6}). The Extended Kalman Filter has to infer the parameter and it is clearly visible from the results that the initial trajectory's inference performance is affected by increasing sensor noise. The initial trajectory's inference performance is significantly decreased and the POMDP controls follow this trend. This is not surprising, since they both use the same EKF. Built-in sensors for measuring lateral velocity typically have a magnitude of noise of around $5.0 \cdot 10^{-2}\ m/s$ [17]. Looking at the performance in figure 8-3 shows that the algorithm does not handle real life sensor noise well. Returning to the sub-question, we can conclude that the algorithm suffers severely from high noise.

Chapter 9

Increasing model complexity

In this chapter we will look into increasing the model complexity. The goal is to find an answer to the last sub-question: *Does the algorithm suit increasing model complexity?*. In the experiments performed in the previous chapters only one parameter, the front tyre stiffness, has been estimated. For a real passenger vehicle it would be preferred to estimate the parameters of all four tyres. We would like to see how increasing the amount of parameters, or increasing the model complexity would affect the algorithm. First, we will go deeper into how the algorithm is affected by the complexity of a system and then we will explain why only one parameter was estimated in the experiments.

9-1 Algorithm size

The algorithm is based on taking the derivative with respect to the steering input and equating this to zero. This is a very simple and effective method, but, as discussed previously, it does result in big matrices that have to be calculated, because it is an analytic solution. The algorithm contains multiple linearized matrices that need to be calculated each iteration. The largest among them have the size n^4, where n is the amount of states the model contains, i.e. the size of the state vector. The parameter that has to be estimated becomes part of the so called "augmented state vector". This means that every uncertain parameter increases the state size by 1, and then exponentially increasing the whole algorithm in size.

9-2 Root of the problem

To get a clear look into why a small increase of the model complexity has a big impact on the computational performance, we need to look into the biggest matrices of the algorithm. One of the big algorithm matrices is U_t, which can be found in appendix A. We will examine the calculation of this matrix, beginning at the system dynamics. Let's say we have the following

simple system:

$$x_{t+1} = f[x_t, u_t] + m, \quad m \sim \mathcal{N}[0, M[x_t, u_t]] \tag{9-1}$$
$$z_t = h[x_t] + n, \quad n \sim \mathcal{N}[0, N[x_t]] \tag{9-2}$$

Where f represents the discretized dynamics. Using Euler this becomes:

$$f[x_t, u_t] = x_t + dt \cdot (A_{cont} \cdot x_t + B_{cont} \cdot u_t) \tag{9-3}$$

Then, it is used for the EKF equations:

$$A_t = \frac{\partial f[x_t, u_t]}{\partial x}, \quad H_t = \frac{\partial h}{\partial x}[f[\hat{x}_t, u_t]] \tag{9-4}$$
$$\Gamma_t = A_t \Sigma_t A_t^T + M[\hat{x}_t, u_t] \tag{9-5}$$
$$K_t = \Gamma_t H_t^T (H_t \Gamma_t H_t^T + N[\hat{x}_t])^{-1} \tag{9-6}$$

Here the computational problems begin. Multiplying all these matrices is not a problem, but dividing them leads to a big symbolic answer. Following these calculations:

$$\Phi[\hat{x}_t, \Sigma_t, u_t] = \Gamma_t - K_t H_t \Gamma_t \tag{9-7}$$

The matrix Φ now has to be turned into a vector, this is done by putting all of the columns under each other, creating a column vector. Doing this for Σ as well leads to the final calculation:

$$U_t = \frac{\partial \operatorname{vec}[\Phi]}{\partial \operatorname{vec}[\Sigma]} \left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right] \tag{9-8}$$

This final step also significantly increases the algorithms size. Both matrices are of size $n \times n$, meaning the vector versions become size $n^2 \times 1$. Taking the jacobian of one with respect to the other leads to a matrix of size $n^2 \times n^2$, hence the algorithm matrices scale with n^4. The size of the matrix is important, but even more important is the fact that each element is loaded with calculations. The dynamics are inputted as symbolics, meaning that every calculation step each element becomes larger and larger. Recalling the discretization of the system dynamics in equation 9-3, it is clear to see the importance of the chosen method. This calculation is done in the beginning, before the matrix grows with symbolics each calculation step. Replacing Euler with RK4 makes the discretized dynamics f considerably larger. Combining this with the rest of the calculation steps is a recipe for slow performance and a heavy algorithm. This computationally expensiveness is what makes the algorithm unsuitable for increasing model complexity.

9-3 Amount of parameters estimated

In practise, increasing the amount of parameters to be estimated lead to computational shortage very easily. For the experiments containing only one augmented parameter it takes roughly 2 hours to calculate the preparation matrices in symbolics on an i7 laptop with 8GB of RAM. Each 10 second simulation can then be run within approximately 5 minutes. Augmenting the state with a second parameter, lead to a depletion of RAM and MATLAB running out

of it before finishing the preparation. This meant only one parameter could be estimated in this setup. Optimizing the algorithm script and implementing it into C++ can decrease calculation time immensely, but the size of the algorithm is to be considered. Using a simple and computationally inexpensive dynamical model does not result in a small algorithm.

Returning to the sub-question, it becomes clear that the algorithm seems to be unsuitable to increase model complexity. Adding an uncertain parameter causes the total algorithm to scale with n^4, which is significant. Naturally, there is a considerable amount of performance to be gained by improving efficiency of the script and using C++, but more linear scaling algorithms are far more suitable for large, complex systems.

Chapter 10

Conclusion and recommendations

The goal of this research is to test the POMDP algorithm for autonomous driving passenger vehicles. Looking at the experiments performed and the corresponding results, it becomes clear that the selected algorithm [3] is able to increase parameter inference performance. The algorithm will update the trajectory in such a way that more useful information is obtained for the Extended Kalman Filter. The EKF can then improve the parameter estimate accordingly. In all of the experiments, the algorithm is able to improve the parameter inference performance.

The main disadvantage to the algorithm, when used with the vehicle model within realistic bounds, is the relatively improved performance. As can be seen in the results, an initial trajectory containing steering input already gives information about the parameter estimate. The EKF uses this to update the parameter estimate continuously and also uses this parameter estimate in its own prediction model. The updated trajectory does improve inference performance, but it is not exceptionally better. It only seems capable of marginal improvement, unless the steering input is not restricted by saturation. As soon as the saturation is removed and the system is allowed to perform unrealistic movements, the estimation of the parameter is perfect within two timesteps. This means that the saturated results have a lot of room for improvement. Looking at the way the algorithm works, it has to do with the combination of the EKF and the vehicle model. The vehicle model is responsible for the measurements and the EKF is responsible for the interpretation of these measurements with respect to the internal dynamical model. If the parameter estimation could be done by something other than the EKF or the measurements could be manipulated to excite the EKF more, it could be possible to reach faster convergence on the parameter estimate. In this setup the vehicle cannot provide the insane accelerations the EKF needs to update the parameter quickly and adequately.

Returning to the sub-questions, the first one was: *What is the most suitable vehicle model?*. From the reviewed papers it became clear that the bicycle model is the most suitable one for this research. For the tyre model, the linearized one was chosen. They both were effective in being accurate under normal driving (low slip conditions), while being simple and computationally inexpensive. The next question was: *What is the most promising algorithm?*.

From the literature it became clear that the most promising algorithm was the one used in Webb et al. [3]. The exceeding factors being the amount of parameters being inferred and the similar dynamical systems in the experiments. The third question was: *Does the algorithm perform on a vehicle model?*. To answer this question, the vehicle model has been implemented together with the algorithm in MATLAB and the setup has been tested. From the results it became clear that parameter inference performance was significantly increased by changing the trajectory. The updated trajectory caused the parameter estimate to converge to a correct value within 2 timesteps, or 0.02 seconds. The only downside being the incredibly high steering input of $8 \cdot 10^4$ *rad*, resulting in unrealistically high lateral acceleration and yaw rate. The fourth question was: *Does the algorithm perform within realistic bounds?*. Using the same simulation as before, but now with saturation on the steering input results were gathered. Compared to the unbounded results parameter estimation performance is decreased, but compared to the original trajectory it is still a significant improvement. Within the total simulation time of $10s$ an accurate estimate was not reached, but the estimate was within 10% of the correct value. The fifth question was: *Does the algorithm suffer from high noise?*. Again using the same simulation as before, but now with increasing levels of sensor noise results were gathered. From the results it became clear that the algorithms parameter estimation performance suffers from increasing noise. The noise was increased until real-life sensor noise levels for lateral velocity and performance was significantly decreased compared to the original level of noise. The final sub-question was: *Does the algorithm suit increasing model complexity?*. Again using the same simulation as before, but now increasing the amount of parameters to estimate. From the results it became clear that it does not suit increasing model complexity. The algorithm became so big, that more than one parameter was not achieved on the used hardware. Looking into the big matrices that cause computational problems, it became clear that these matrices scale with a factor of n^4. Combining this with multiple matrices that need to be calculated and it becomes clear why adding only 1 parameter is this computationally expensive.

All of the sub-questions have been explored and the research question can now be answered. The research question being: *Is it worthwhile to regard the problem of estimating parameters online for an autonomous passenger vehicle as a POMDP?*. The tested algorithm can improve the trajectory, such that lateral tyre stiffness parameter estimation performance is significantly increased. However, applying realistic bounds to the steering angle severely limits inference performance. Taking into account the effect of high noise and increasing of model complexity on performance and calculation speed decreases appeal. Based on these findings, we can conclude that in current form it is not worthwhile to regard this problem as a POMDP, certainly not solve it using this algorithm. The potential is visible, but the drawbacks outweigh the improved performance.

This research has brought interesting results and conclusions. The algorithm shows high potential and excellent performance without saturation on the steering input. Interesting for future research would be, to look into a different way of inferring the parameter value. The Extended Kalman Filter seems to infer quickly only when the input is very high, so perhaps an alternative to the current EKF could boost performance. This does not necessarily mean replacing the EKF, it could be done by modifying it to be more sensitive to smaller steering inputs. This could be highly beneficial.

Appendix A

Equations of the POMDP algorithm

In this appendix the derivation and the equations of the algorithm are displayed. The original algorithm is thoroughly explained in Van den Berg et al. [14] and Van den Berg et al. [15]. Here only this original one will be clarified. The modifications done in Webb et al. [3] that assist in parameter estimation have already been described in chapter 4.

A-0-1 System and Kalman Filter

The algorithm starts off with the dynamics of the system. The system is defined in the following matter.

$$x_{t+1} = f[x_t, u_t] + m, \quad m \sim \mathcal{N}[0, M[x_t, u_t]] \tag{A-1}$$

$$z_t = h[x_t] + n, \quad n \sim \mathcal{N}[0, N[x_t]] \tag{A-2}$$

Where x_{t+1} represents the discretized system dynamics and z_t represents the sensor data, which is based on the current state. Note that both the system and the sensors are influenced by noise, which is assumed to be of Gaussian nature.

On top of the system, there is a Kalman Filter present. The job of the Kalman Filter is to maintain a belief over the current state, based on the sensor data z_t and an internal model of the system. The equations of the Extended Kalman Filter are stated below.

$$\hat{x}_{t+1} = f[\hat{x}_t, u_t] + K_t(z_{t+1} - h[f[\hat{x}_t, u_t]]) \tag{A-3}$$

$$\Sigma_{t+1} = \Gamma_t - K_t H_t \Gamma_t \tag{A-4}$$

where

$$\Gamma_t = A_t \Sigma_t A_t^T + M[\hat{x}_t, u_t] \tag{A-5}$$

$$K_t = \Gamma_t H_t^T (H_t \Gamma_t H_t^T + N[\hat{x}_t])^{-1} \tag{A-6}$$

$$A_t = \frac{\partial f}{\partial x}, \quad H_t = \frac{\partial h}{\partial x}[f[\hat{x}_t, u_t]] \tag{A-7}$$

Where $\hat{x}$ and Σ represent the state estimation and it's variance respectively. Note that there is assumed to be no noise on the control input u when passed on to the Extended Kalman Filter.

A-0-2 Value iteration

The goal of solving the POMDP is finding the optimal policy. In our case case this policy will lead to the fastest tyre parameter inference performance. The effectiveness of a policy is defined by the predefined cost function, c_t. The designer chooses the shape of the cost function and tunes the parameters, in order to accommodate for the desired goals. The optimal policy is the one that has the least cost over the whole trajectory. This means minimizing the following equation.

$$E_{z_1,\dots,z_l}[c_l[\mathcal{X}_l] + \sum_{t=0}^{l-1} c_t[\mathcal{X}_t, u_t]] \tag{A-8}$$

A general approach is using value iteration. A value functions is different than a cost function. A value function not only accounts for the current cost, but also for (a part of) the expected future cost. The value is calculated for each step of the trajectory using backwards recursion.

$$v_l[\mathcal{X}] = c_l[\mathcal{X}] \tag{A-9}$$

$$v_t[\mathcal{X}] = \min_u(c_t[\mathcal{X}, u] + \underset{z}{E}[v_{t+1}[\beta[\mathcal{X}, u, z]]]) \tag{A-10}$$

$$\pi_t[\mathcal{X}] = \arg\min_u(c_t[\mathcal{X}, u] + \underset{z}{E}[v_{t+1}[\beta[\mathcal{X}, u, z]]]) \tag{A-11}$$

Where β represents the discrete belief dynamics. By working backwards, the value function of the next time is already available for the current time step. This allows the POMDP to be solved.

A-0-3 Algorithm

To apply the method of value iteration to the system, there has to be a model for the belief dynamics. The belief is updated via the Extended Kalman Filter, as discussed above. The belief is stochastic, due to the fact that the measurement z cannot be known in advance. Assuming the stochastic belief dynamics can modelled as a Gaussian leads to the following simplified equations.

$$\hat{x}_{t+1} = f[\hat{x}, u] + w, \quad w \sim \mathcal{N}[0, W[\hat{x}_t, \Sigma_t, u_t]] \tag{A-12}$$

$$\Sigma_{t+1} = \Phi[\hat{x}_t, \Sigma_t, u_t] \tag{A-13}$$

where

$$W[\hat{x}_t, \Sigma_t, u_t] = K_t H_t \Gamma_t \tag{A-14}$$

$$\Phi[\hat{x}_t, \Sigma_t, u_t] = \Gamma_t - K_t H_t \Gamma_t \tag{A-15}$$

Note that these are based on the belief the Kalman Filter holds over the time. The nominal trajectory defined as a series of beliefs and control inputs.

$$traj_{nominal} = (\bar{x}_0, \bar{\Sigma}_0, \bar{u}_0, ..., \bar{x}_l, \bar{\Sigma}_l, \bar{u}_l) \tag{A-16}$$

The value function is defined in the following manner.

$$v_t[\hat{x}, \Sigma] \approx s_t + 0.5(\hat{x} - \bar{x}_t)^T S_t(\hat{x} - \bar{x}_t) + \mathbf{s}_t^T(\hat{x} - \bar{x}_t) + \mathbf{t}_t^T vec[\Sigma - \bar{\Sigma}_t] \tag{A-17}$$

Where $vec[\Sigma]$ represents a vector where all the columns of matrix Σ are placed under each other. Looking back at equation A-9, the parameters of the value function can be set for $t = l$ using Taylor expansion.

$$s_l = c_l[\bar{x}_l, \bar{\Sigma}_l] \tag{A-18}$$

$$S_t = \frac{\partial^2 c_l}{\partial \bar{x} \partial \bar{x}}[\bar{x}_l, \bar{\Sigma}_l] \tag{A-19}$$

$$\mathbf{s}_l = \frac{\partial c_l}{\partial \bar{x}}[\bar{x}_l, \bar{\Sigma}_l] \tag{A-20}$$

$$\mathbf{t}_l^T = \frac{\partial c_l}{\partial vec[\Sigma]}[\bar{x}_l, \bar{\Sigma}_l] \tag{A-21}$$

Applying these equations to equation A-10 gives:

$$\begin{aligned} v_t[\hat{x}, \Sigma] = \min_u (c_t[\hat{x}, \Sigma, u] + E_w[s_{t+1} + 0.5(f[\hat{x}, u] + w - \bar{x}_{t+1})^T S_{t+1}(f[\hat{x}, u] + w - \bar{x}_{t+1}) \\ + \mathbf{s}_{t+1}^T (f[\hat{x}, u] + w - \bar{x}_{t+1}) + \mathbf{t}_{t+1}^T vec[\Phi[\hat{x}, \Sigma, u] - \bar{\Sigma}_{t+1}]]) \end{aligned}$$

$$\begin{aligned} v_t[\hat{x}, \Sigma] = \min_u (c_t[\hat{x}, \Sigma, u] + s_{t+1} + 0.5(f[\hat{x}, u] - \bar{x}_{t+1})^T S_{t+1}(f[\hat{x}, u] - \bar{x}_{t+1}) \\ + \mathbf{s}_{t+1}^T (f[\hat{x}, u] - \bar{x}_{t+1}) + \mathbf{t}_{t+1}^T vec[\Phi[\hat{x}, \Sigma, u] - \bar{\Sigma}_{t+1}] + 0.5 vec[S_{t+1}]^T vec[W[\hat{x}, \Sigma, u]]) \end{aligned} \tag{A-22}$$

Given that

$$\bar{x}_{t+1} = f[\bar{x}_t, \bar{u}_t] \tag{A-23}$$

$$\bar{\Sigma}_{t+1} = \Phi[\bar{x}_t, \bar{\Sigma}_t, \bar{u}_t] \tag{A-24}$$

There can be assumed:

$$f[\bar{x}, \bar{u}] - \bar{x}_{t+1} \approx F_t(\hat{x} - \bar{x}_t) + G_t(u - \bar{u}_t) \tag{A-25}$$

$$vec[\Phi[\hat{x}, \Sigma, u] - \bar{\Sigma}_{t+1}] \approx T_t(\hat{x} - \bar{x}_t) + U_t vec[\Sigma - \bar{\Sigma}_t] + V_t(u - \bar{u}_t) \tag{A-26}$$

$$vec[W[\hat{x}, \Sigma, u]] \approx y_t + X_t(\hat{x} - \bar{x}_t) + Y_t vec[\Sigma - \bar{\Sigma}_t] + Z_t(u - \bar{u}_t) \tag{A-27}$$

$$c_t[\hat{x}, \Sigma, u] \approx q_t + 0.5 \begin{bmatrix} \hat{x} - \bar{x}_t \\ u - \bar{u}_t \end{bmatrix}^T \begin{bmatrix} Q_t & P_t^T \\ P_t & R_t \end{bmatrix} \begin{bmatrix} \hat{x} - \bar{x}_t \\ u - \bar{u}_t \end{bmatrix} + \begin{bmatrix} q_t \\ r_t \end{bmatrix}^T \begin{bmatrix} \hat{x} - \bar{x}_t \\ u - \bar{u}_t \end{bmatrix} + p_t^T vec[\Sigma - \bar{\Sigma}_t] \tag{A-28}$$

where

$$F_t = \frac{\partial \mathbf{f}}{\partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\mathbf{u}}_t\right], \quad G_t = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\left[\overline{\mathbf{x}}_t, \overline{\mathbf{u}}_t\right]$$
$$T_t = \frac{\partial \operatorname{vec}[\Phi]}{\partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad U_t = \frac{\partial \operatorname{vec}[\Phi]}{\partial \operatorname{vec}[\Sigma]}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$V_t = \frac{\partial \operatorname{vec}[\Phi]}{\partial \mathbf{u}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad X_t = \frac{\partial \operatorname{vec}[W]}{\partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$Y_t = \frac{\partial \operatorname{vec}[W]}{\partial \operatorname{vec}[\Sigma]}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad Z_t = \frac{\partial \operatorname{vec}[W]}{\partial \mathbf{u}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$\mathbf{y}_t = \operatorname{vec}\left[W\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]\right], \quad q_t = c_t\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$Q_t = \frac{\partial^2 c_t}{\partial \hat{\mathbf{x}} \partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad \mathbf{q}_t^T = \frac{\partial c_t}{\partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$R_t = \frac{\partial^2 c_t}{\partial \mathbf{u} \partial \mathbf{u}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad \mathbf{r}_t^T = \frac{\partial c_t}{\partial \mathbf{u}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$
$$P_t = \frac{\partial^2 c_t}{\partial \mathbf{u} \partial \hat{\mathbf{x}}}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right], \quad \mathbf{p}_t^T = \frac{\partial c_t}{\partial \operatorname{vec}[\Sigma]}\left[\overline{\mathbf{x}}_t, \overline{\Sigma}_t, \overline{\mathbf{u}}_t\right]$$

Using these assumptions on equation A-22 gives:

$$v_t[\hat{\mathbf{x}}, \Sigma] \approx \min_{\mathbf{u}} \left(e_t + \frac{1}{2} \begin{bmatrix} \hat{\mathbf{x}} - \overline{\mathbf{x}}_t \\ \mathbf{u} - \overline{\mathbf{u}}_t \end{bmatrix}^T \begin{bmatrix} C_t & E_t^T \\ E_t & D_t \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}} - \overline{\mathbf{x}}_t \\ \mathbf{u} - \overline{\mathbf{u}}_t \end{bmatrix} + \begin{bmatrix} \mathbf{c}_t \\ \mathbf{d}_t \end{bmatrix}^T \begin{bmatrix} \hat{\mathbf{x}} - \overline{\mathbf{x}}_t \\ \mathbf{u} - \overline{\mathbf{u}}_t \end{bmatrix} + \mathbf{e}_t^T \operatorname{vec}\left[\Sigma - \overline{\Sigma}_t\right] \right) \tag{A-29}$$

where

$$\begin{aligned} C_t &= Q_t + F_t^T S_{t+1} F_t, & D_t &= R_t + G_t^T S_{t+1} G_t \\ E_t &= P_t + G_t^T S_{t+1} F_t, & e_t &= q_t + s_{t+1} + \tfrac{1}{2} \operatorname{vec}\left[S_{t+1}\right]^T \mathbf{y}_t \\ \mathbf{c}_t^T &= \mathbf{q}_t^T + \mathbf{s}_{t+1}^T F_t + \mathbf{t}_{t+1}^T T_t + \tfrac{1}{2} \operatorname{vec}\left[S_{t+1}\right]^T X_t \\ \mathbf{d}_t^T &= \mathbf{r}_t^T + \mathbf{s}_{t+1}^T G_t + \mathbf{t}_{t+1}^T V_t + \tfrac{1}{2} \operatorname{vec}\left[S_{t+1}\right]^T Z_t \\ \mathbf{e}_t^T &= \mathbf{p}_t^T + \mathbf{t}_{t+1}^T U_t + \tfrac{1}{2} \operatorname{vec}\left[S_{t+1}\right]^T Y_t \end{aligned} \tag{A-30}$$

Then simply taking the derivative with respect to u and set equal to zero leads to an updated control input.

$$\mathbf{u} = L_t(\hat{\mathbf{x}} - \bar{\mathbf{x}}_t) + \mathbf{l_t} + \bar{\mathbf{u}}_\mathbf{t} \tag{A-31}$$

where

$$L_t = -D_t^{-1} E_t \tag{A-32}$$
$$\mathbf{l_t} = -D_t^{-1} \mathbf{d_t} \tag{A-33}$$

Filling it back in gives:

$$\begin{aligned} s_t &= e_t + \tfrac{1}{2} \mathbf{d}_t^T \mathbf{1}_t, & S_t &= C_t + L_t^T E_t \\ \mathbf{s}_t^T &= \mathbf{c}_t^T + \mathbf{l}_t^T E_t, & \mathbf{t}_t^T &= \mathbf{e}_t^T \end{aligned} \tag{A-34}$$

Which in turn can be used for the next iteration, working all the way backwards.

Bibliography

[1] Tesla, "Dual motor model s and autopilot," *www.tesla.com/blog/dual-motor-model-s-and-autopilot*, 2014.

[2] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1, pp. 35–40, IEEE, 1999.

[3] D. J. Webb, K. L. Crandall, and J. van den Berg, "Online parameter estimation via real-time replanning of continuous gaussian pomdps," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5998–6005, 2014.

[4] V. Krishnamurthy, *Partially Observed Markov Decision Process*. Cambridge University Press, 2016.

[5] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[6] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1099, 2015.

[7] C. J. Taylor, J. Košecká, R. Blasi, and J. Malik, "A comparative study of vision-based lateral control strategies for autonomous highway driving," *The International Journal of Robotics Research*, vol. 18, no. 5, pp. 442–453, 1999.

[8] J. Ryu, "State and parameter estimation for vehicle dynamics control using gps.," *PhD thesis, Stanford*, 2005.

[9] H. Pacejka, *Tire and vehicle dynamics*. Elsevier, 2005.

[10] D. E. Smith and J. M. Starkey, "Effects of model complexity on the performance of automated vehicle steering controllers: Model development, validation and comparison," *Vehicle System Dynamics*, vol. 24, no. 2, pp. 163–181, 1995.

[11] Q. Vroom, "Pomdp based autonomous driving passenger vehicles," *Literature Thesis, TU Delft*, 2018.

[12] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A pomdp approach," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1288–1302, 2014.

[13] P. Slade, P. Culbertson, Z. Sunberg, and M. Kochenderfer, "Simultaneous active parameter estimation and control using sampling-based bayesian reinforcement learning," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 804–810, 2017.

[14] J. Van Den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, 2012.

[15] J. Van Den Berg, S. Patil, and R. Alterovitz, "Efficient approximate value iteration for continuous gaussian pomdps," *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[16] B. Shyrokau, *Vehicle Dynamics A [lecture slides]*. Technical University of Delft, 2016-2017.

[17] D. M. Bevly, "Global positioning system (gps): A low-cost velocity sensor for correcting inertial sensor errors on ground vehicles," *Journal of dynamic systems, measurement, and control*, vol. 126, no. 2, pp. 255–264, 2004.