Learning Model Predictive Control for Autonomous Racing

Improvements and Model Variation in Model Based Controller

Shuqi Xu
Authors
Shuqi Xu <shuqixu@kth.se>
Electric Engineering and Computer Science
KTH Royal Institute of Technology

Place for Project
Berkeley, California, US
Model Predictive Control Lab
Department of Mechanical Engineering
University of California, Berkeley

Examiner
Mikael Johansson
Professor
Department of Automatic Control, EECS
KTH Royal Institute of Technology

Supervisor
Ugo Rosolia
Graduate Student Researcher
Model Predictive Control Lab
Department of Mechanical Engineering
University of California, Berkeley
Acknowledgements

This is the thesis work of my master study in KTH Royal Institute of Technology and UC Berkeley. I am sincerely thankful to Prof. Borrelli for inviting me and hosting me doing this 6-month thesis project in Model Predictive Control Lab. I would like to thank Ugo Rosolia for supporting in both theoretical and practical questions, moreover I really appreciate the team work with Ugo Rosolia and his responsibility when we were working together day and night as a team. I would also like to thank Jon Gonzales for his work on building Berkeley Autonomous Racing Car (BARC) platform and practical suggestions for my experiment.

I would like to thank my examiner in KTH, Mikael Johansson for guiding me through my master’s program. He taught me Model Predictive Control (MPC) and gave me the chance of developing MPC toolbox in Julia, which helped me to gain the necessary theoretical knowledge and practical skills for my master thesis.

I would like to thank my parents for giving me the opportunity to follow my study in KTH and write thesis in UC Berkeley.

Last but not least, I would like to thank my roommate in Berkeley for helping me adapt to the new life and accommodate me in the comfortable apartment. I would also like to thank my brothers and sisters in Channel Church for all the fellowship time and your encouragement during my staying. I was also thankful to my friends in KTH, who gave me a lot of help on my study and master thesis application. Special thanks to Peiyan Li and Motoya Ohnishi for the "seminar time" during our dinner, when we were discussing from Artificial Intelligent to mathematics and philosophy.
Abstract

In this work, an improved Learning Model Predictive Control (LMPC) architecture for autonomous racing is presented. The controller is reference free and is able to improve lap time by learning from history data of previous laps. A terminal cost and a sampled safe set are learned from history data to guarantee recursive feasibility and non-decreasing performance at each lap. Improvements have been proposed to implement LMPC on autonomous racing in a more efficient and reliable way. Improvements have been done on three aspects. Firstly, system identification has been improved to be run in a more efficient way by collecting feature data in subspace, so that the size of feature data set is reduced and time needed to run sorting algorithm can be reduced. Secondly, different strategies have been proposed to improve model accuracy, such as least mean square with/without lifting and Gaussian process regression. Thirdly, for reducing algorithm complexity, methods combining different model construction strategies were proposed. Also, running controller in a multi-rate way has also been proposed to reduced algorithm complexity when increment of controller frequency is necessary. Besides, the performance of different system identification strategies have been compared, which include strategy from newton’s law, strategy from classical system identification and strategy from machine learning. Factors that can possibly influence converged result of LMPC were also investigated, such as prediction horizon, controller frequency. Experiment results on a 1:10 scaled RC car illustrates the effectiveness of proposed improvements and the difference of different system identification strategies.

Keywords

Model Predictive Control, Autonomous Racing, Learning, System Identification
Abstract

I detta arbete, presenteras en förbättrad inlärning baserad modell prediktiv kontroll (LMPC) för autonom racing, styralgoritm är referens fritt och har visat sig att kunna förbättra varvtid genom att lära sig ifrån historiska data från tidigare varv. En terminal kostnad och en samplad säker mängd är lärde ifrån historisk data för att garantera rekursiv genomförbarhet och icke-avtagande prestanda vid varje varv.


Prestanda av olika systemidentifiering har jämförts, bland annat, Newtons lag, klassisk systemidentifierings metoder och strategier från maskininlärning. Faktorer som eventuellt kan påverka konvergens av LMPC resultat har också undersökts. Såsom, prediktions horisont, styrfrekvensen.

Experimentresultat på en 1:10 skalad RC-bilen visar effektiviteten hos föreslagna förbättringarna och skillnaderna i olika systemidentifierings strategier.

Nyckelord

modell prediktiv kontroll, autonom racing, inlärning, systemidentifierings
Abbreviations and Notations

Abbreviation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>LMPC</td>
<td>Learning Model Predictive Control</td>
</tr>
<tr>
<td>BARC</td>
<td>Berkeley Autonomous Racing Car</td>
</tr>
<tr>
<td>QP</td>
<td>Quadratic Programming</td>
</tr>
<tr>
<td>GPR</td>
<td>Gaussian Process Regression</td>
</tr>
<tr>
<td>LMS</td>
<td>Least Mean Square</td>
</tr>
</tbody>
</table>

Notations

\[ s \quad \text{curvilinear abscissa} \]
\[ e_y \quad \text{lateral distance to the center line, lateral position error} \]
\[ e_\psi \quad \text{heading angle error} \]
\[ v \quad \text{car speed} \]
\[ v_x \quad \text{car longitudinal speed} \]
\[ v_y \quad \text{car lateral speed} \]
\[ \dot{\psi} \quad \text{car yaw rate} \]
\[ \beta \quad \text{car body slip angle} \]
\[ a \quad \text{longitudinal acceleration} \]
\[ \delta_f \quad \text{front steering angle} \]
# Contents

1 Introduction 1

2 Theory and Formula 4

2.1 Model Predictive Control (MPC) 4
2.2 Learning Model Predictive Control (LMPC) 5
2.3 Least Mean Square (LMS) 8
2.4 Gaussian Process Regression (GPR) 8

3 State Estimation: Multi-rate Kalman Filter 11

3.1 Measurement Signals and Multi-rate Implementation 11
3.2 Coordinate Transformation 12
3.3 $V_y$ Estimation 13

4 Controller: Proposed Models and Comparison 14

4.1 From Newton’s Law 15
4.2 From Classical System Identification, Least Mean Square 19
4.3 From Machine Learning, Gaussian Process Regression (GPR) 24

5 Experiment Setup 26

5.1 Racing Track Design and Track Data Saving 26
5.2 Steering and Acceleration Low Level Mapping 28
5.3 LMPC Initialization: Path Following Controller Design 28
5.4 LMPC Controller 29
5.5 LMPC Code Architecture for Autonomous Racing 32

6 Implementation Strategies 33

6.1 Efficient Feature Data Collecting 33
6.2 Local Feature Data Selection 35
6.3 Multi-rate Controller Design 35

7 Experiment Results 37

7.1 LMPC Learning Comparison 37
7.2 CPU Time Comparison 40
7.3 Nonlinear Kinematic Bicycle Model and Linearized Dynamic Bicycle Model .......................... 40
7.4 Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS 41
7.5 Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS with Lifting ......................... 42
7.6 Time Variant Nonlinear Dynamic Bicycle Model by LMS+GPR ........................................ 45
7.7 Biased Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS+GPR ......................... 46
7.8 Time Invariant Nonlinear Dynamic Bicycle Model (10Hz/20Hz) ....................................... 48

8 Conclusions and Future Work 50
8.1 Conclusions ........................................... 50
8.2 Future Work ........................................... 52
1 Introduction

Automation has made manufacturing more efficient. In production lines, to improve production efficiency, many robotics arms were programmed so that they can finish a specific task in a short time. To program those robotics arm in an optimal way, engineers need to know the optimal way to finish tasks. Solutions to those tasks usually use knowledges from optimal control theory [1].

In recent years, model based control gained a great development in both theory and practice in industry. Model predictive control (MPC, [2]) was known to control community as one of the best practice of model based control and many successful applications [3] [4] of model predictive control has been seen in industries. Using the knowledge of dynamic programming [5], MPC can give a prediction of what will happen in the future. Using the knowledge of optimization [6], MPC can solve the problem and get an sequence of optimal solution for the prediction. In opposite to model based control was error based control, where only inputs and outputs of the system can be seen. A typical error based control was proportional-integral-derivative (PID) control, which has dominated the practice for over 80 years. Either error based control or model based control has its own advantages and disadvantages.

Being benefited from rapid progress in computer science, more and more computational power was available so that complex problems that cannot be solved before can now be solved and even in a shorter time. For example, model predictive control can be applied to chemistry industry [7] 30 years ago, which had sampling time up to several minutes. But now, model predictive control could be applied to autonomous driving [8], which usually has sampling time less than 0.1 second.

Model predictive control has many advantages. For example, it can handle input constraints and state constraints quite well. Moreover, in MPC, solving multi-input-multi-output (MIMO) control problem only requires a small amount of extra work than solving single-input-single-output (SISO) control problem, which requires much more work in other control methods.

Model predictive control is a promising approach to solve many control problems.
However, it is only true if we are able to get an accurate model for the real system and have enough computational power to solve the optimization problem. It is difficult to solve MPC problems when the prediction horizon is long, which will increase the number of constraints in optimization. It is also difficult to solve MPC problems when the model is nonlinear, which will increase the number of steps to find the optimal solution for the optimization problem. Because of those difficulties, there are still many technical challenges in applying MPC to solve complex control problems.

Autonomous racing was a challenging task. If we want to solve autonomous racing task using MPC, it will be challenging. Firstly, it is because the vehicle model was highly nonlinear due to nonlinearity of tire. Secondly, it is a long prediction horizon if we let the horizon cover the full lap. There has been researches on finding the optimal time trajectories through a given track [9]. Interesting model based predictive control method has also be proposed and tested on real system [8].

In model predictive control, the model used in the controller is important. The technical challenge here is to find a model that is not only accurate to represent the real system but also not too complex so that it did not take many steps to solve the optimization problem. To find a suitable model, there are many strategies we can use from the field of system identification [10]. For example, we can apply newton’s law to get dynamic equations to represent the real system. From classical system identification, method of least mean square can also be applied to identify the model equations. Method like Gaussian process regression [11] from machine learning can also be applied to identify the model.

A novel method called Learning Model Predictive Control (LMPC, [12]) was proposed to solve repetitive tasks. The controller is reference free and is able to improve its performance by learning from previous iterations. In a repetitive task, the end of the task is also the beginning of the task. In LMPC, input sequence and state sequence to finish the task in previous iterations were used to guarantee the recursive feasibility and non-decreasing performance.

Those different system identification strategies will be used in this work to construct model for the controller and performance of those controllers will
be compared. Experiment result on a 1:10 scaled car illustrates the difference between different system identification strategies.
2 Theory and Formula

In this chapter, theory of model predictive control (MPC) and learning model predictive control (LMPC) are presented. Besides, theory of least mean square and Gaussian process regression, which are needed for proposed system identification strategies are also presented in the second half of this chapter.

2.1 Model Predictive Control (MPC)

This section gives a brief introduction of the theory of Model Predictive Control (MPC) [2], based on which the theory of Learning Model Predictive Control (LMPC) is built.

Model predictive control is an optimal control method, originally developed from Chemical Process Industry [13], which can be briefly described as optimizing (minimizing) the cost \( J(x) \) on optimizer \( u \). And the MPC problem can be formulated as following.

Given discrete time system,

\[
x_{t+1} = f(x_t, u_t),
\]

where \( x_t \in \mathbb{R}^{n_x} \), \( u_t \in \mathbb{R}^{n_u} \), are the system states and inputs vector, subjected to constraints \( x_t \in \mathcal{X}, u_t \in \mathcal{U} \) and \( n_x, n_u \) are the number of the state dimensions and input dimensions respectively. The MPC controller tries to minimize the cost defined as

\[
J(x) = \sum_{i=1}^{N+1} (x_i - x_{ref})^T Q (x_i - x_{ref}) + \sum_{i=1}^{N} (u_i - u_{ref})^T R (u_i - u_{ref}) + \sum_{i=2}^{N+1} (x_i - x_{i-1})^T Q_{deri} (x_i - x_{i-1}) + \sum_{i=2}^{N} (u_i - u_{i-1})^T R_{deri} (u_i - u_{i-1})
\]

\[
x_{i+1} = f(x_i, u_i), i \in \{1, 2, \cdots, N\}
\]

\[
x_i \in \mathcal{X}, i \in \{1, 2, \cdots, N + 1\}
\]

\[
u_i \in \mathcal{U}, i \in \{1, 2, \cdots, N\}
\]

where \( Q \) is the state scaling cost matrix, \( R \) is the the input scaling cost matrix,
used to tune the behavior of the MPC controller. Also, cost terms such as the cost for state derivative and input derivative can also be defined by $Q_{\text{deri}}, R_{\text{deri}}$, which is to prevent some abrupt change in input and the achieved state sequence. Moreover, in the constraints part, firstly, states on the prediction horizon need to satisfy state transition function $f$, which can be time invariant or time variant, $f_t(x_t, u_t)$. Besides, we might also have some constraints on states and input, which can be modeled by constraints $x_t \in \mathcal{X}, u_t \in \mathcal{U}$. In this way, we can write down the standard MPC formulation, which is also a standard optimization problem formulation. Using standard optimization solvers, we can solve the optimization problem and get the values of the optimizer, which is the optimal solution of the problem.

Since this is a finite horizon optimal control problem, in practice, we can design a terminal cost matrix $Q_f$ such that in a way to account for the future cost to achieve a "better" control performance.

MPC is a great algorithm for reference tracking by taking account model information into optimization problem, it can calculate the corresponding optimal solution for the parameter setting we put in the problem and at the same time satisfies the constraints. Nowadays, with more and more computational power available and better optimization solvers available, MPC is being applied to many other applications, such as active steering \cite{14} in the field of autonomous driving, which has the requirement that the optimization needs to be done in a short sampling time.

2.2 Learning Model Predictive Control (LMPC)

The theory of learning MPC will be briefly introduced in this section, for detailed proof and original work, we refer the interested readers to the original work \cite{12}. Compared to standard MPC controller, the biggest advantage of Learning MPC and the main difference to standard MPC is that it is reference free. Thus, no precomputed reference trajectory is needed. Instead, with such a freedom provided, the controller is now able to "learn" the "best" racing trajectory and the corresponding optimal control sequence, which is the meaning of the name
Before introducing the theory of LMPC, we need to give a definition for sampled safe set, which is the key concept for constructing the terminal state constraint in LMPC.

Sampled safe set are used to construct the terminal state constraint for recursive feasibility.

**Definition: Sampled Safe Set**

$x^i_t$ is the system state at time $t$ of iteration $i$. The repetitive task is finished once in each iteration.

We introduce the notation for previous iterations. At the $j^{th}$ iteration, the matrix $x^j$ and $u^j$ collect the state and input history at $j^{th}$ iteration.

$$u^j = [u^j_0, u^j_1, \ldots, u^j_t, \ldots]$$
$$x^j = [x^j_0, x^j_1, \ldots, x^j_t, \ldots]$$

$$SS^j = \left\{ \bigcup_{i \in M^j} \bigcup_{t=0}^{\infty} x^i_t \right\}$$

where

$$M^j = \left\{ k \in [0, j] : \lim_{t \to \infty} x^k_t = x_F \right\}$$ (2.6)

$SS^j$ is the collection of all the states the agent reached in iteration $i$, for $i \in M^j$. And $M^j$ is the set of the index of those successful iterations $k$ for $k \leq j$. And from Eq.(2.6), we can know that $M^i \subset M^j, \forall i \leq j$, which also implies

$$SS^i \subset SS^j, \forall i \leq j$$ (2.7)

The LMPC algorithm tries to solve an infinite horizon optimal control problem in
a finite horizon optimal control way with the help of the sampled safe set and the corresponding cost on each element belonging to this Sample Safe Set. The LMPC formulation [12] is shown in Eq. (2.8).

\[
J_{t \rightarrow t+N}^{\text{LMPC}, j} = \min_{u_{t|t}, \ldots, u_{t+N-1|t}} \left[ \sum_{k=t}^{t+N-1} h(x_{k|t}, u_{k|t}) + Q^{j-1}(x_{t+N|t}) \right] 
\]

(2.8a)

\[
x_{k+1|t} = f(x_{k|t}, u_{k|t}), \forall k \in [t, \ldots, t + N - 1]
\]

(2.8b)

\[
x_{t|t} = x^j_t
\]

(2.8c)

\[
x_{k|t} \in \mathcal{X}, u_{k|t} \in \mathcal{U}, \forall k \in [t, \ldots, t + N - 1]
\]

(2.8d)

\[
x_{t+N|t} \in S S^{j-1}
\]

(2.8e)

**Terminal state cost:**

Here, terminal cost \(Q^{j-1}(x_{t+N|t})\) gives the performance improvement and guarantees the optimality in general applications. This terminal cost is explained as "cost to go" as the number of steps to finish the current iteration.

**Terminal constraint relaxation**

Eq. (2.8) is the original LMPC formulation [12], terminal constraint is a set of discrete states of previous iterations, which makes the optimization problem an computationally expensive mix-integer programming problem. To make the optimization tractable, a convex-hull relaxation can be done for this terminal constraint [15] [16]. In this way, the terminal constraint Eq. (2.8e) can be reformulated as in Eq. (2.9)

\[
\tilde{S} S^{j-1} = \left\{ x_i \in SS^{j-1}, \alpha_i \in [0,1] : \sum_{i=1}^{\left| SS^{j-1} \right|} \alpha_i x_i \right\}
\]

(2.9)

where \(\left| SS^{j-1} \right|\) is the cardinality of the Sample Safe Set. In this way, the mixed-integer programming problem is reformulated as a nonlinear programming problem which requires less computational power.

If the system is linear, all properties of LMPC is preserved. If the system is
nonlinear, all LMPC properties are lost, but with smooth dynamics and close points in the sampled safe set, this relaxation is fine, which can be shown by experiments.

Hence, the difference between MPC and LMPC is that in LMPC, it has a terminal state constraint and terminal cost constructed from history data.

### 2.3 Least Mean Square (LMS)

Consider a function

\[ y = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \]  
\[ (2.10) \]

With feature data we got for this function

\[
\begin{bmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_m
\end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{n,m} \\
                      x_{1,2} & x_{2,2} & \cdots & x_{n,m} \\
                      \vdots      & \vdots      & \vdots      & \vdots      \\
                      x_{1,m} & x_{2,m} & \cdots & x_{n,m}
\end{bmatrix} \begin{bmatrix} \theta_1 \\
                        \theta_2 \\
                        \vdots      \\
                        \theta_n
\end{bmatrix}
\]
\[ (2.11) \]

We need to solve the problem

\[
\theta^* = \arg\min_{\theta} \| y - X\theta \|_2
\]
\[ (2.12) \]

### 2.4 Gaussian Process Regression (GPR)

In this section, we will give a brief introduction of Gaussian process regression (GPR, [11], [17]).

Consider \( M \) inputs and the corresponding measurement

\[
\{ z_1^T, z_2^T, \cdots, z_M^T \}, \{ y_1^T, y_2^T, \cdots, y_M^T \}
\]
\[ (2.13) \]
which are from unknown function \( g(z) : \mathbb{R}^{n_z} \rightarrow \mathbb{R}^{n_d} \) under the following statistical model

\[
y_j = g(z_j) + w_j
\]  

(2.14)

where \( w_j \) is i.i.d. Gaussian noise with zeros mean and diagonal variance \( \Sigma_w = \text{diag}([\sigma_1^2, \sigma_2^2, \cdots, \sigma_{n_d}^2]) \). Assuming a GP prior on \( g \) in each output dimension \( a \in \{1, 2, \cdots, n_d\} \), the measurement data is normally distributed with

\[
y_{.,a} \sim \mathcal{N}(0, K^a_{zz} + \sigma^2_a)
\]  

(2.15)

where \( K^a_{zz} \) is the Gram matrix of the points using the kernel function \( k^a(\cdot, \cdot) \), i.e. \( [K^a_{zz}]_{ij} = k^a(z_i, z_j) \). We choose squared exponential kernel function

\[
k(z, z) = \sigma_f^2 \exp \left( -\frac{1}{2} (z - z)^T L^{-1} (z - z) \right)
\]  

(2.16)

Squared exponential kernel function is a good candidate to be used here because the continuity and smoothness of Gaussian distribution. When two inputs are close, the output of squared exponential kernel function is close to 1 otherwise, the output will be closer to 0, which is a quantification of the correlation of two different inputs.

The joint distribution of the training data and an arbitrary test point \( z \) in output dimension \( a \) is given by

\[
p([y]_a, [\mathbf{y}]_{.,a}) \sim \mathcal{N} \left( 0, \begin{bmatrix} K^a_{zz} & K^a_{z\mathbf{y}} \\ K^a_{z\mathbf{y}}^T & K^a_{\mathbf{y}\mathbf{y}} \end{bmatrix} \right)
\]  

(2.17)

where \( [K^a_{zz}]_{ij} = k^a(z_j, z) \), \( K^a_{z\mathbf{y}} = (K^a_{zz})^T \) and \( K^a_{\mathbf{y}\mathbf{y}} = k^a(z, z) \). And the resulting conditional distribution is Gaussian with

\[
p([y]_a || [\mathbf{y}]_{.,a}) \sim \mathcal{N}(\mu^d_a(z), \Sigma^d_a(z))
\]  

(2.18a)

\[
\mu^d_a(z) = K^a_{z\mathbf{y}} (K^a_{zz} + I \sigma_a^2)^{-1} [\mathbf{y}]_{.,a}
\]

(2.18b)

\[
\sigma^d_a(z) = K^a_{zz} - K^a_{z\mathbf{y}} (K^a_{zz} + I \sigma_a^2)^{-1} K^a_{z\mathbf{y}}
\]
And we call the resulting GP approximation of the unknown function $g(z)$

$$d(z) \sim \mathcal{N}(\mu^d(z), \Sigma^d(z)) \quad (2.19)$$

with $\mu^d = [\mu_1^d, \mu_2^d, \ldots, \mu_n^d]$ and $\Sigma^d = \text{diag}([\Sigma_1^d, \Sigma_2^d, \ldots, \Sigma_n^d])$. Evaluating Eq.(2.19) has cost $O(n_d n_z M)$ and $O(n_d n_z M^2)$ for mean and variance. And in our application, since we can pre-calculate some of the matrix, the complexity is linear to $M$. 
3 State Estimation: Multi-rate Kalman Filter

Kalman filter [18], is an essential contribution in system control, which is the optimal state estimation method. In this section, we will present how we construct the Kalman filter for BARC and propose the best way to do state estimation for BARC according to our knowledge.

3.1 Measurement Signals and Multi-rate Implementation

From on-board sensors, we can get measurement signal from three sensors, indoor GPS, inertial measurement unit, and encoder. From indoor GPS, we can get \( x, y, z \) measurement at up to \( 16 \) Hz. From inertial measurement unit, we can get linear accelerations \( a_x, a_y, a_z \) on three mutual perpendicular axes and angular velocity \( \omega_x, \omega_y, \omega_z \) on three mutual perpendicular axes. All signals from inertial measurement unit can be at up to \( 100 \) Hz. From encoder, we can get the wheel angular speed at up to \( 20 \) Hz.

For the details of those sensors and how to purchase and install them, please go to BARC-project\(^1\) for more details of hardware. The original purpose of BARC platform was for student education. So it should be low cost and easy to build. All the sensors used on BARC are low cost, which makes the state estimation a challenging task for this thesis. For example, indoor GPS system gives accuracy to be \( 2 \) cm, besides, we also experienced GPS signal jump and package loss problems.

Three sensors are running at different frequencies and we also need to choose a frequency for the Kalman filter. To make the state estimation run in a correct way, we need to change the matrix \( H(k), R(k), z(k) \), which are denoted as color red in the second step, measurement update step. The change of those matrix or vectors depends on whether there is a newly available measurement signal. If the measurement is not updated, the corresponding element in \( z(k) \) and the corresponding columns and rows in matrix \( H(k), R(k) \) will be deleted.

\(^1\)http://www.barc-project.com
Model:

\[ x(k) = A(k - 1)x(k - 1) + u(k - 1) + v(k - 1) \]  
\[ z(k) = H(k)x(k) + w(k) \]  

(3.1a)  
(3.1b)

Prior update/Prediction step:

\[ \hat{x}_p(k) = A(k - 1)\hat{x}_m(k - 1) + u(k - 1) \]  
\[ P_p(k) = A(k - 1)P_m(k - 1)A^T(k - 1) + Q(k - 1) \]  

(3.2a)  
(3.2b)

Posterior update/Measurement update step:

\[ P_m(k) = \left( (P_p^{-1}(k) + H^T(k)R^{-1}(k)H(k)) \right)^{-1} \]  
\[ \hat{x}_m(k) = \hat{x}_p(k) + P_m(k)H^T(k)R^{-1}(k)(z(k) - H(k)\hat{x}_p(k)) \]  

(3.3a)  
(3.3b)

3.2 Coordinate Transformation

To do state estimation correctly, coordinate transformation needs to be done for signals from inertial measurement unit, which is because of vehicle body’s roll and pitch. It is not difficult to do coordinate transformation, but it is really important to do coordinate transformation because this coordinate transformation will introduce a bias into the state estimation. In the end, we need to do coordinate transformation for three measurement signals, \(a_x, a_y, \psi\).

If we use \(\theta\) to denote roll angle, \(\phi\) to denote pitch angle and \(\psi\) to denote yaw angle, then we can get the coordinate transformation [19].

\[ a_{x,\text{car}} = a_{x,\text{imu}} \cos(\phi) + a_{y,\text{imu}} \sin(\phi) \sin(\theta) + a_{z,\text{imu}} \sin(\phi) \cos(\theta) \]  
\[ a_{y,\text{car}} = a_{y,\text{imu}} \cos(\theta) - a_{z,\text{imu}} \sin(\theta) \]  
\[ \psi_{\text{car}} = \dot{\phi}_{\text{imu}} \sin(\theta)/\cos(\phi) + \psi_{\text{imu}} \cos(\theta)/\cos(\phi) \]  

(3.4a)  
(3.4b)  
(3.4c)
3.3 $V_y$ Estimation

Vehicle lateral velocity is not directly measurable, so it can only be estimated from Kalman filter. However, when the vehicle is at low speed or the vehicle is going straight, small $\dot{\psi}$, $V_y$ is not observable. Thus, to have a better estimation for state $V_y$, we propose to use an artificial measurement for $V_y$ at low speed or small yaw rate, $\dot{\psi}$.

**Artificial $V_y$ Measurement Switching Conditions:**

1. $V_y = \arctan(\tan(\delta_f))$ at low longitudinal speed or small yaw rate where, $\delta_f$ is front steering angle.

2. Remove this artificial $V_y$ measurement when the longitudinal speed is high and yaw rate is big.

The longitudinal speed threshold and yaw rate threshold are tuning parameters, which brings more engineering work to estimator tuning, since threshold tuning here is coupled with the measurement covariance matrix tuning.
4 Controller: Proposed Models and Comparison

Since the controller we are discussing here is model-based controller. Model used in the controller is crucial to the performance. The requirement is that the model used in the controller should have a high accuracy to represent the real physical system, but the model should also bring a low complexity. The reason for high accuracy is to have a better prediction. And the reason for low complexity is for real time implementation.

such that the algorithm meets the requirement of real-time implementation, which is calculating the desired control signal within the sampling time.

In this section, with the theories and formula introduced in section 2, we will introduce different model candidates constructed from different methods.

Before introducing different vehicle models, we need to introduce the coordinate system we used, we are using Frenet’s coordinate system [20] as shown in figure 4.1. All the models introduced in this section is constructed based on this coordinate system.

![Figure 4.1: Frenet coordinate](image)

In Frenet coordinate system, curvature \( c(s) \) is used to represent the shape of the race track, the curvature is defined as the inverse of the curve radius, \( c(s) = \frac{1}{r(s)} \).

For a race track given in global coordinate \((x, y)\), the curvature can be calculated...
as
\[ c = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}} \]

Models introduced in the following part of this chapter are constructed in Frenet coordinate system.

4.1 From Newton’s Law

From Newton’s second law, we know that \( F = m \cdot a \), in this section, we will introduce the method of constructing the model from vehicle dynamics equations.

4.1.1 Nonlinear Kinematic Bicycle Model

One thing to be noted, this model, nonlinear kinematic bicycle model has only four states, thus vector \( v_x \) and vector \( v_y \) are combined into one vector \( v \). And state yaw rate \( \dot{\psi} \) is not included into this model, which is actually limiting the accuracy of this model to represent the real physical system.

\[
\begin{align*}
\dot{s} &= v \cdot \frac{\cos(e_\psi + \beta)}{1 - e_y \cdot c(s)} \\
\dot{e}_y &= v \cdot \sin(e_\psi + \beta) \\
\dot{e}_\psi &= \frac{v}{L_r} \cdot \sin(\beta) - v \cdot c(s) \frac{\cos(e_\psi + \beta)}{1 - e_y \cdot c(s)} \\
\dot{v} &= a - c_f \cdot v
\end{align*}
\] (4.1)

Here, \( c(s) \) is the curvature dependent on state \( s \), which is the state representing where the car is on the track. \( \beta \) is the vehicle body slip angle, which is approximated by \( \beta = \arctan(\tan(\delta_f)) \). And \( a \) is the acceleration input into the vehicle, \( c_f \) is a coefficient representing the effect of friction and aerodynamic drag.
4.1.2 Linearized Kinematic Bicycle Model

By applying linearization on nonlinear kinematic bicycle model, we can get the linearized kinematic bicycle model, which has a lower complexity. The original nonlinear model is linearized on points, \((x_{lin}, u_{lin})\).

\[
\begin{pmatrix}
\dot{s} \\
\dot{e}_y \\
\dot{e}_\psi \\
\dot{v}
\end{pmatrix} = A_{lin}(x_{lin}, u_{lin}) \begin{pmatrix} s \\ e_y \\ e_\psi \\ v \end{pmatrix} + B_{lin}(x_{lin}, u_{lin}) \begin{pmatrix} a \\ \delta_f \end{pmatrix}
\] (4.2)

4.1.3 Nonlinear Dynamic Bicycle Model

Dynamic bicycle model is a more complex and gives a more accurate description of the state transition especially for transient state transition. It includes two more states, \(v_y, \dot{\psi}\), which is described based on Pacejka tire model [21].

\[
\begin{align*}
\dot{s} &= \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_y \cdot c(s)} \\
\dot{e}_y &= v_x \cos(e_\psi) + v_y \sin(e_\psi) \\
e_\psi &= \dot{\psi} - c(s) \cdot \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_y \cdot c(s)} \\
\dot{v}_x &= \dot{\psi}v_y - c_f \cdot v_x + a \\
\dot{v}_y &= \frac{1}{m} (F_f \cdot \cos(\delta_f) + F_r) - \dot{\psi}v_x \\
\ddot{\psi} &= \frac{1}{I_z} (L_f \cdot F_f \cdot \cos(\delta_f) - L_r \cdot F_r)
\end{align*}
\] (4.3a-f)

where \(F_f\) and \(F_r\) are front and rear tire force generated from tire slip angle.

\[
\begin{align*}
F_f &= D \cdot \sin(C \cdot \arctan(B \cdot \alpha_f)) \\
\alpha_f &= \arctan\left(\frac{v_y + L_f \dot{\psi}}{|v_x|}\right) - \delta_f \\
F_r &= D \cdot \sin(C \cdot \arctan(B \cdot \alpha_r)) \\
\alpha_r &= \arctan\left(\frac{v_y - L_r \dot{\psi}}{|v_x|}\right)
\end{align*}
\]
\( \alpha_f \) and \( \alpha_r \) are front tire slip angle and rear tire slip angle. \( F_f \) and \( F_r \) are front and rear tire lateral force respectively, which are calculated based on \textit{Pacejka} tire model function [21]. \( D, C, B \) are constant need to be identified for different cases, which are mainly determined by road surface and the type of tire. A plot of the \textit{Pacejka} tire function depending on tire slip angle is plotted in figure 4.2.

![Figure 4.2: Pacejka tire model with \( D = 1, C = 1.6, B = 6 \)](image)

Either using this model or not depends mainly on two facts, accuracy of coefficients, \( B, C, D \) in tire model and computational power. Only if when we have accurate tire coefficients and enough computational power, this model will be recommended. Another possible limitation of this model is that the \textit{Pacejka tire model} only models lateral tire force. The tire force coupling effect in some critical case is not modeled. So even if we identify the \textit{Pacejka tire model} accurately, it might not be an accurate tire model in some cases. Besides, load transfer and other vehicle dynamics factors are not included in this dynamics bicycle model. In our case, we are using optimization modeling package \texttt{JuMP.jl} and nonlinear solver, \texttt{Ipopt.jl} in \texttt{Julia} [22], which did not satisfy the second condition. Thus, we are not able to put this model into the controller and test its performance.
4.1.4 Fully linearized dynamic bicycle model

Due to the fact that we did not have enough computational power, another thing we can try is to linearize the nonlinear dynamic bicycle model, which make the optimization into a quadratic programming (QP) problem.

Similar to the method used in linearized kinematic bicycle model, we can also linearize the nonlinear dynamic bicycle model. The benefit we can get from linearizing dynamic bicycle model can be more than the linearization of kinematic model since the complexity reduction gained here is believed more than the complexity reduction gained from kinematic bicycle model linearization. And in the end, we get a convex problem. Linearized dynamic bicycle model can be described by Eq. (4.4), where \( A_{\text{lin}}(x_{\text{lin}}, u_{\text{lin}}) \), \( B_{\text{lin}}(x_{\text{lin}}, u_{\text{lin}}) \) are matrix dependent on state, \( x_{\text{lin}} \) and input \( u_{\text{lin}} \). And \( x_{\text{lin}} = (s, e_y, e_\psi, v_x, v_y, \dot{\psi})^T \) is the state to be linearized at and \( u_{\text{lin}} = (a, \delta_f)^T \) is the input to be linearized at.

\[
\begin{pmatrix}
\dot{s} \\
\dot{e}_y \\
\dot{e}_\psi \\
\dot{v}_x \\
\dot{v}_y \\
\dot{\psi}
\end{pmatrix}
= A_{\text{lin}}(x_{\text{lin}}, u_{\text{lin}})
\begin{pmatrix}
s \\
e_y \\
e_\psi \\
v_x \\
v_y \\
\dot{\psi}
\end{pmatrix}
+ B_{\text{lin}}(x_{\text{lin}}, u_{\text{lin}})
\begin{pmatrix}
a \\
\delta_f
\end{pmatrix}
\tag{4.4}
\]

4.1.5 Partially linearized dynamic bicycle model

We can also do the linearization only on the last three states, \( v_x, v_y, \dot{\psi} \), and keep the first three states, \( s, e_y, e_\psi \) nonlinear, which is the same to those three equations in nonlinear dynamic bicycle model. This partially linearized model can be described in Eq. 4.5.
The reason to do partial linearization is that the solver can solve problem with nonlinearity in the first three states. And keeping the first three states nonlinear, the model is more accurate.

Since in this racing problem, the longitudinal velocity, $v_x$ can be assumed to be always positive, the absolute operator $|\cdot|$ in tire force calculation, $F_f, F_r$ can be removed so that the dynamic bicycle model is differentiable.

Possible tools can be used for doing linearization are, for example, package SymEngine.jl in Julia or symbolic toolbox in Matlab. Some practical tips to do this linearization and to use this linearization model in the algorithm is that don’t use function subs() in either Julia or Matlab, but hard code the linearized system model, since calling subs() is really expensive. And we are desired to reduce the sampling time, so take some time to hard code the really long state transition matrix.

4.2 From Classical System Identification, Least Mean Square

Although, nonlinear kinematic bicycle model and linearized dynamic bicycle model are two possible candidates, they have their disadvantages. Nonlinear kinematic bicycle model has only four states, which has low accuracy to represent the real physical system. The accuracy of linearized dynamic bicycle model is limited by the accuracy of tire model coefficients. In our case, we did not have
a validated accurate tire model, which might bring us an inaccurate model. This is the reason why we also propose to do classical system identification and use the identified model in the controller.

Using the theory of least mean square (LMS) introduced in section 2.3, we can identify a model to represent the real system by using feature data collected from experiments. Besides, in MPC, we can also choose to do LMS once and fix the model for the horizon or do LMS for every point on the horizon, which will bring us time invariant and time variant model for MPC controller.

### 4.2.1 Time Invariant (variant) Nonlinear Dynamic Bicycle Model by LMS

As explained in section 2.3, we can apply LMS here to do system identification. The model remained to be identified is

\[
\dot{s} = \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_y \cdot c(s)} \tag{4.6a}
\]

\[
\dot{e}_y = v_x \cos(e_\psi) + v_y \sin(e_\psi) \tag{4.6b}
\]

\[
e_\psi = \dot{\psi} - c(s) \cdot \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_y \cdot c(s)} \tag{4.6c}
\]

\[
v_x = \theta_{x,1} v_x + \theta_{x,2} v_y + \theta_{x,3} \dot{\psi} + \theta_{x,4} a + \theta_{x,5} \delta_f \tag{4.6d}
\]

\[
v_y = \theta_{y,1} v_x + \theta_{y,2} v_y + \theta_{y,3} \dot{\psi} + \theta_{y,4} \delta_f \tag{4.6e}
\]

\[
\ddot{\psi} = \theta_{\psi,1} v_x + \theta_{\psi,2} v_y + \theta_{\psi,3} \dot{\psi} + \theta_{\psi,4} \delta_f \tag{4.6f}
\]

And we need to solve the problem of

\[
\theta^* = \arg \min_{\theta} \|y - X\theta\|_2 \tag{4.7}
\]

where \(y\) and \(X\) are selected from pre-collected feature data set. For now, we assume that we have this pre-selected dataset. The way to efficiently construct this data set will be explained in section 6.1. For example, to determine \(\theta_{x,\cdot}\), we
have

\[
\mathbf{y} = \begin{pmatrix}
v_{x,i_1+1} - v_{x,i_1} \\
v_{x,i_2+1} - v_{x,i_1} \\
\vdots \\
v_{x,i_n+1} - v_{x,i_1}
\end{pmatrix} \quad \mathbf{X} = \begin{pmatrix}
v_{y,i_1} & v_{x,i_1} & \dot{\psi}_{i_1} & a_1 & \delta_1 \\
v_{y,i_2} & v_{x,i_2} & \dot{\psi}_{i_2} & a_2 & \delta_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
v_{y,i_n} & v_{x,i_n} & \dot{\psi}_{i_n} & a_n & \delta_n
\end{pmatrix} \quad \mathbf{\theta} = \begin{pmatrix}
\theta_{x,1} \\
\theta_{x,2} \\
\theta_{x,3} \\
\theta_{x,4} \\
\theta_{x,5}
\end{pmatrix}
\]

(4.8)

where \(i_1, i_2, \ldots, i_n\) are index of the points we selected from the pre-selected feature point data set. The number of points selected from the data set is a tuning parameter. The selection criterion is 2-norm distance between the current state and the feature points on dimensions \(v_x, v_y, \dot{\psi}, a\) and \(\delta_f\).

\[
d_i = \| (v_x, v_y, \dot{\psi}, a, \delta_f)^T - (v_{x,i}, v_{y,i}, \dot{\psi}_i, a_i, \delta_{f,i})^T \|
\]

(4.9)

After calculating the 2-norm distance from the current state to all feature points, we will selected \(n\) feature points with the minimum 2-norm distance, which will be put into Eq. (4.13) for \(\theta_{v_x}\). This can also be done for \(\theta_y\) and \(\theta_{\dot{\psi}}\) in the same way.

Besides, 2-norm might needs to be weighted on different dimensions since some dimensions might be more important and different dimensions have different units.

\[
d_i = \left\| w_{v_x}, w_{v_y}, w_{\dot{\psi}}, w_a, w_{\delta_f} \right\| \left( (v_x, v_y, \dot{\psi}, a, \delta_f) - (v_{x,i}, v_{y,i}, \dot{\psi}_i, a_i, \delta_{f,i}) \right)^T \right\|
\]

(4.10)

4.2.2 Time Invariant (variant) Nonlinear Dynamic Bicycle Model by LMS with Lifting

The dynamic bicycle model in Eq. (4.3) is used to derive the following system identification model, Eq. (4.11). We chose to do feature lifting in this way because
therms like $\dot{v}_x, \dot{v}_y$, $\frac{\dot{v}_x}{v_x}, \frac{\dot{v}_y}{v_y}$ appears in Pacejka tire model.

\[
\dot{s} = \frac{v_x \cos(\psi) - v_y \sin(\psi)}{1 - e_y \cdot c(s)} \tag{4.11a}
\]

\[
\dot{e}_y = v_x \cos(\psi) + v_y \sin(\psi) \tag{4.11b}
\]

\[
\dot{\psi} = \psi - \frac{v_x \cos(\psi) - v_y \sin(\psi)}{1 - e_y \cdot c(s)} \tag{4.11c}
\]

\[
\dot{v}_x = \theta_{x,1} \dot{\psi} v_y + \theta_{x,2} v_x + \theta_{x,3} a + \theta_{x,4} \delta_f \tag{4.11d}
\]

\[
\dot{v}_y = \theta_{y,1} \frac{\dot{v}_y}{v_x} + \theta_{y,2} \dot{\psi} v_x + \theta_{y,3} \frac{\dot{\psi}}{v_x} + \theta_{y,4} \delta_f \tag{4.11e}
\]

\[
\ddot{\psi} = \theta_{\psi,1} \dot{\psi} + \theta_{\psi,2} \frac{\dot{v}_y}{v_x} + \theta_{\psi,3} \delta_f \tag{4.11f}
\]

$\theta_{\cdot \cdot}$ can be determined online from pre-collected dataset by solving

\[
\theta^* = \arg \min_{\theta} ||y - X \theta||_2 \tag{4.12}
\]

where $y$ and $X$ are selected from pre-collected data set. For now, we assume that we have this pre-selected dataset, of which the way to efficiently construct this data set will be explained in section 6.1. For example, to determine $\theta_{x,\cdot}$, we have

\[
y = \begin{pmatrix}
    v_{x,i1+1} - v_{x,i1} \\
v_{x,i2+1} - v_{x,i1} \\
    \vdots \\
v_{x,in+1} - v_{x,in}
\end{pmatrix}
\quad
X = \begin{pmatrix}
\dot{\psi}_{i1} & v_{y,i1} & v_{x,i1} & a_1 & \delta_1 \\
\dot{\psi}_{i1} & v_{y,i1} & v_{x,i2} & a_2 & \delta_2 \\
    \vdots & \vdots & \vdots & \vdots & \vdots \\
\dot{\psi}_{in} & v_{y,in} & v_{x,in} & a_n & \delta_n
\end{pmatrix}
\quad
\theta = \begin{pmatrix}
\theta_{x,1} \\
\theta_{x,2} \\
\theta_{x,3} \\
\theta_{x,4}
\end{pmatrix} \tag{4.13}
\]

Feature data is selected in the same way as LMS without lifting.

### 4.2.3 Time Invariant (variant) Linearized Kinematic Bicycle Model

This model combines two system identification strategies, nonlinear kinematic bicycle model and nonlinear dynamic bicycle model by LMS (with lifting), so that the advantages of both model are utilized.
Nonlinear dynamic bicycle model by LMS can help to provide accurate linearization points, and thus to identify a more accurate model. By using linearized kinematic bicycle model, we can reduce the computational complexity. The idea here is to use the nonlinear dynamic bicycle model by LMS and previous MPC solution to forecast a "better" trajectory and use the states on this "better" trajectory to do linearization instead of the previous MPC solutions.

We believe that the reason why this is working in experiment as we will show later in section 7 is that linearization points will be dominant in the optimal solution calculated from MPC controller.

The idea to get the linearization point is simple. Firstly, use the same method in section 4.2.2, we can get the model by LMS. Apply shifted MPC input solution from previous time step, \((u_{2|k-1}, u_{3|k-1}, \cdots, u_{N|k-1}, u_{N|k-1})\) on the model identified by LMS. In this way, we can get a sequence of forecast states, which will be used in the linearization points.

For every time step, we have previous solution

\[
u_{k-1} = \begin{pmatrix} u_{1|k-1} \\ u_{2|k-1} \\ \vdots \\ u_{N|k-1} \end{pmatrix} = \begin{pmatrix} a_{1|k-1} & \delta f_{1|k-1} \\ a_{2|k-1} & \delta f_{2|k-1} \\ \vdots & \vdots \\ a_{N|k-1} & \delta f_{N|k-1} \end{pmatrix}
\] (4.14)

from LMPC controller, and we know our current state \(x_k\) from the measurement data. By applying least mean square (LMS), Eq. (4.12) for system identification, we can identify the current model, \(\tilde{f}\), and inputs from previous solution \(u_{2|k-1}\) will be applied for current state \(x_{1|k}\). This is done in a series way to get a more accurate time variant model and a more accurate trajectory. Those points in figure 4.3, \(x_{1|k}, \tilde{x}_{2|k}, \cdots, \tilde{x}_{N-1|k}\) and \(u_{2|k-1}, u_{3|k-1}, \cdots, u_{N|k-1}, u_{N|k-1}\) will be used as the linearization points to linearize the kinematic bicycle model. Thus, we will get a time variant linearized kinematic bicycle model, which will be used in LMPC controller.

Linearization points,
will be used in linearized kinematic bicycle model,

\[
\begin{bmatrix}
    u_{2|k-1} & x_{1|k} \\
    u_{3|k-1} & \tilde{x}_{2|k} \\
    \vdots & \vdots \\
    u_{N|k-1} & \tilde{x}_{N-1|k} \\
    u_{N|k-1} & \tilde{x}_{N|k}
\end{bmatrix}
\]

4.3 From Machine Learning, Gaussian Process Regression (GPR)

We did not construct the model purely by GPR, we use GPR to identify part of the model. So, we need to have a model, and use GPR to identify the remaining part of the model to compensate the model mismatch part.

From the collected data, by comparing the solution from MPC controller and measurement data, we can know the model mismatch quantitatively by calculating one step prediction error.

**Definition: one step prediction error:**
state difference between the model in MPC controller and the true model by applying the same control input.

\[
x_{k+1} = \tilde{f}_k(x_k, u_k) + w = \begin{pmatrix} \dot{s}_k \\ \dot{e}_{y,k} \\ \dot{\psi}_{x,k} \\ \dot{\psi}_{y,k} \\ \psi_{k} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ w_{v_x,k} \\ w_{v_y,k} \end{pmatrix}
\] (4.15)

GPR is used to estimate \(w_{v_x,k}, w_{v_y,k}, w_{\psi,k}\) in equation 4.15 The feature input data for GPR can be got by calculating the one step prediction error.

The model \(\tilde{f}_k\) in Eq. 4.15 can be any model candidates introduced before. And GPR is compensating the remaining model mismatch part.

Another thing to be noted in the experiment is that, the one step prediction error we can collect is not the true one step prediction error since we can not get the ground truth, instead we calculate one step prediction error based on state values estimated by state estimator.
5  Experiment Setup

After introducing the knowledge needed to design the state estimator and the controller, we will introduce the other practical parts needed in the experiments, which are the other construction components to successfully do the experiment. Experiment hardware setup is illustrated in figure 5.1. We need to design the track and save the track data. We also need to assemble the vehicle and do hardware low level mapping.

![Racing track design and curvature profile](image)

Figure 5.1: Racing track design and curvature profile

5.1  Racing Track Design and Track Data Saving

There are two requirements for track design. One is that the racing track needs to fit the size of the experiment room. The other one is that the maximum curvature on the track needs to be within the vehicle cornering limit, because every vehicle has its smallest turning radius. The requirement to design a racing track is that the track can push the car to its working limit.

With those racing track design requirements, we design the track as shown in figure 5.2.

This track has width 0.9m, track length is 18.2m and the discretization is 0.01m.

Another important thing that should be born in mind while designing the track is the kinematic turning limit from the car steering system, which can be described
by figure 5.3. And by geometric calculation, the calculation of the minimum turning radius can be described by Eq. (5.1).

\[ R = \sqrt{L_r^2 + (L_f + L_r) \cdot \cot(\delta_f)} \]  

(5.1)

where, \( \delta_f = \frac{\delta_{f_0} + \delta_t}{2} \) due the Ackerman Steering Geometry [23]. If we substitute the value of the maximum steering angle into the equation, we can get the minimum turning radius according to which we can design a racing track. If we know the maximum lateral acceleration the vehicle can achieve, then we can also do another simple calculation and get an idea of the fastest speed the vehicle can go at this minimum turning radius by \( v_{\text{max}} = \sqrt{R^2 \cdot a_y} \).

Track Data Saving

Track data is saved as two one dimensional array. One array is for \( s \), for vehicle position localization. The other array is an one-to-one mapping for the array of \( s \),
which saved the curvature information for the track.

Firstly, from sensors and state estimator, we can know the \((x, y)\) position for the car on the track. By simple geometric calculation, we can get the corresponding \(s\) for the vehicle current position. After knowing where the car is on the track, we can use the second array to know the curvature of the vehicle current position and also the curvature for those points on the prediction horizon.

### 5.2 Steering and Acceleration Low Level Mapping

An accurate low level mapping is really crucial for a successful experiment. Even with a really advanced controller, an inaccurate low level mapping might deteriorate the experiment results. For readers who are interested in the details of how to do low level mapping, please refer to this [24] master thesis. There are two low level mappings need to be done, low level mapping for steering and low level mapping for acceleration.

### 5.3 LMPC Initialization: Path Following Controller Design

We will keep using Frenet’s coordinate system introduced in the beginning of chapter 4.

So if kinematic bicycle model or linearized kinematic bicycle model is used, our system state and input vector are

\[ [s, e_y, e_{\psi}, v], [a, \delta_f] \]

and if nonlinear dynamic bicycle model, linearized dynamic bicycle model or nonlinear dynamic bicycle model by LMS is used, our system state and input vector are

\[ [s, e_y, e_{\psi}, v_x, v_y, \dot{\psi}], [a, \delta_f] \]

In our solution to this autonomous racing problem, we need to construct sampled
safe set, so that we can initialize the Learning Model Predictive Control (LMPC) algorithm. Thus we need a standard MPC controller to do path following task to construct the initial sampled safe set.

We will use nonlinear kinematic bicycle model in this MPC controller, since when the car is going at a low longitudinal speed, nonlinear kinematic bicycle model can give an accurate modeling for the car. The path following controller can be described as in Eq. (5.2).

\[
J(x_1) = \sum_{i=1}^{N+1} (x_i - x_{ref})^T Q(x_i - x_{ref}) + \sum_{i=1}^{N} (u_i - u_{ref})^T R(u_i - u_{ref}) \\
+ \sum_{i=2}^{N+1} (x_i - x_{i-1})^T Q_{deri}(x_i - x_{i-1}) + \sum_{i=2}^{N} (u_i - u_{i-1})^T R_{deri}(u_i - u_{i-1})
\]

\[
x_{i+1} = f(x_i, u_i), i \in \{1, 2, \cdots, N\}
\]

\[
x_{lb} < x_i < x_{ub}, i \in \{1, 2, \cdots, N + 1\}
\]

\[
u_{lb} < u_i < u_{ub}, i \in \{1, 2, \cdots, N\}
\]

where \( f(\cdot, \cdot) \) is the nonlinear kinematic bicycle model and if we let \( x = (s, e_y, e_\psi, \dot{\psi})^T \), the cost matrix will be \( Q[i, i] \neq 0, i \in 2, 3, 4 \) because for path following, it is enough to give a reference to \( e_y, e_\psi, v \). Reference value for \( e_\psi \) and \( e_y \) can be chosen to 0. Reference value for \( v \) can be chosen freely with a reasonable value, such as 1 m/s. The derivative cost for state and input are added for the purpose of gentle behavior.

We can also do path following initialization not only on the centerline, but also on both the inner side and outer side of the track. We can simply change \( e_{y,ref} = 0.9 \cdot w/2 \) or \( e_{y,ref} = -0.9 \cdot w/2 \), where \( w \) is the track width. In this way, we have richer sampled safe set initialization for LMPC.

### 5.4 LMPC Controller

When we apply LMPC to a racing problem, firstly, coordinate system needs to be defined. Here, we will keep using Frenet coordinate system as being consistent.
to path following controller. Secondly, model used in LMPC needs to chosen. Thirdly, the way how to design constraints and cost functions will be introduced in this section, because there are specialized terminal state constraint and terminal state cost for LMPC controller.

In this controller, *Frenet Coordinate* [20] is used describe the state transition, the advantage of this coordinate is that the relative position from the car to the track can be easily described and thus the stage cost function can be formulated conveniently. This *Frenet Coordinate* has been illustrated in figure 4.1.

The LMPC controller can be formulated as

\[
J_{t \rightarrow t+N}^{\text{LMPC},j} = \min_{u_t, \ldots, u_{t+N-1}} \left[ \sum_{k=t}^{t+N-1} h(x_k|t, u_k|t) + Q^{j-1}(x_{t+N}|t) \right] \tag{5.3a}
\]

\[
x_{k+1|t} = f(x_k|t, u_k|t), \forall k \in [t, \ldots, t + N - 1] \tag{5.3b}
\]

\[
x_{t|t} = x^j_t \tag{5.3c}
\]

\[
x_k|t \in \mathcal{X}, u_k|t \in \mathcal{U}, \forall k \in [t, \ldots, t + N - 1] \tag{5.3d}
\]

\[
x_{t+N|t} \in \tilde{S}^{j-1} \tag{5.3e}
\]

**Constraints**

State transition constraint, Eq. (5.3b) and initial state constraint, Eq. (5.3c) are standard. As for state/input constraints, Eq. (5.3d), we have state/input boundary constraints, which are \(x_{\text{lb}} < x < x_{\text{ub}}\), \(u_{\text{lb}} < u < u_{\text{ub}}\), where \(x_{\text{lb}}, x_{\text{ub}}, u_{\text{lb}}, u_{\text{ub}}\) are state lower bound and upper bound.

**Terminal state constraint**

Eq. (5.3e) is the terminal state constraint, in which part of the history data in the previous iteration are selected to construct the sampled safe set, which can be illustrated in figure 5.4, and the convex hull of the sampled safe set will be finally used in the LMPC controller, as explained in section 2.2. Both the way to select and construct the sampled safe set and the idea of convex hull are shown in figure 5.4. Grey points and blue points represent all states the system has reached before, but only blue points are selected into the sampled safe set for terminal state constraint.
The reason is that in this autonomous racing application, the other states reached before are impossible for the terminal state to reach. To reduce the algorithm complexity, we partially select points to construct the sampled safe set.

![Sampled safe set construction](image)

**Figure 5.4: Sampled safe set construction**

**Cost function**

In this racing problem, since LMPC is reference free, so no state cost $x^T Q x$ or input cost $u^T R u$ are included.

1. State and input derivative cost is included

\[ \sum_k (x_k - x_{k-1})^T Q_{deri} (x_k - x_{k-1}) \]

2. Slack cost on soft constraint on $e_y$

3. Terminal state cost

4. Slack cost on soft constraint on terminal state

Terminal cost can be illustrated in figure 5.5 for this autonomous racing application. If the terminal state is far away from the end of finishing the task, it has a higher terminal cost.

The reason of using slack cost on both $e_y$ and terminal state is to guarantee feasibility for every time step in the experiment.
5.5 LMPC Code Architecture for Autonomous Racing

On the software side, we wrote the code for experiment in Robot Operating System (ROS, [25]). Figure 5.6 shows the software architecture, each rectangle in figure 5.6 represents a "node" in ROS. Each node is independent and can be written in different languages. They can also be run at its own frequency. ROS will take care of the communication between different nodes and assign them required computational resource. Figure 5.6 shows the hardware architecture. Communication network was built between BARC and laptop through wifi router.

If it is in simulations, then we don’t need nodes for those sensors, we just need a node for simulator and we can keep the other nodes the same, which is also an advantage of ROS, code modularization.

Figure 5.6: Software architecture
6 Implementation Strategies

Besides the implementation strategies introduced in the previous chapters, in this chapter, we would like to highlight some other implementation strategies that can improve the performance.

6.1 Efficient Feature Data Collecting

In least mean square for system identification, either with lifting or without lifting, we need feature data to do system identification. We have two problems, one is how to construct the data set to do system identification. The other one is how to select “good” data from this data set to do system identification. So in this section and the next section, we will explain how to do feature data set construction and feature data selection in an efficient and accurate way.

In this implementation strategy, we will introduce an efficient way to collect feature data and thus reduce the feature data set size, so that time needed to run sorting algorithm (algorithm complexity is $N \log(N)$) can be reduced.

Since the theory of LMPC guarantees the recursive feasibility and non-decreasing performance. In this application, LMPC for autonomous racing, lap time will be reduced lap by lap and vehicle speed will increase lap by lap. One way to construct feature data set is to use the data from previous several laps since those states and input data are locally closer to states and inputs of current lap, which can reduce the feature data set size. However, the disadvantage is that if there is package loss or any hardware problems, it will be difficult to recover from the failure since the
car can not find feature data that are close to its current state to do LMS for system identification. But to fully construct the feature data set requires a lot of feature data points since it is a 5-dimensional space for $[v_x, v_y, \dot{\psi}, a, \delta_f]$. If we only collect 10 points for each dimension, we need at least $10^5$ feature points. However, this won’t satisfy the requirement of real time implementation.

Because of this, we would like to introduce the following strategy for feature data set construction. It is collecting feature data in a subspace of the 5-dimensional space. We can collect the feature data in this way because some dimensions are correlated, for example, dimension $v_y$, $\dot{\psi}$ and $\delta_f$ are correlated. When the vehicle is steered right, $\dot{\psi}$ is negative if the car is not drifting.

The proposed maneuver for feature data collecting is illustrated in figure 6.1. An 8-figure maneuver, the car go from low speed to high speed. Besides, the curvature is different on this 8-figure, at the two ends on the left and right, the track has the highest curvature. The controller for this maneuver can be simply a path following controller. Moreover, randomness on acceleration control signal is introduced in the maneuver so that it can explore on dimension $a$.

![Figure 6.1: Maneuver to collect feature data efficiently](image)

And the feature data we will collect looks like

$$
\begin{align*}
[s_1, e_{y,1}, e_{\psi,1}, v_{x,1}, v_{y,1}, \dot{\psi}_1], [a_1, \delta_{f,1}] \\
[s_2, e_{y,2}, e_{\psi,2}, v_{x,2}, v_{y,2}, \dot{\psi}_2], [a_2, \delta_{f,2}] \\
\vdots \\
[s_M, e_{y,M}, e_{\psi,M}, v_{x,M}, v_{y,M}, \dot{\psi}_M], [a_M, \delta_{f,M}]
\end{align*}
$$
In this way, we can collect and construct the feature data set efficiently, and this feature data set will be used in the next section for feature data selection.

6.2 Local Feature Data Selection

As introduced in the previous section, we can efficiently construct the feature data set. In this section, we will explain the way of selecting feature data more effectively.

If we assume that there are $M$ feature data points in the feature dataset, to have a better LMS, we need some ”good” data points which are locally close to the current state. We will select the feature data locally by 2-norm criterion, which can be explained by the following equations.

Current state and input:
$$[s, e_y, e_{\psi}, v_x, v_y, \dot{\psi}, \dot{\psi}, a, \delta_f]$$

sort$$\left(\left\{\|v_x - v_{x,i}, v_y - v_{y,i}, \dot{\psi} - \dot{\psi}_i, a - a_i, \delta_f - \delta_{f,i}\|^2 \right| i = 1, 2, \cdots, M\right)$$

We firstly calculate the two norm between the current state and all feature points in the feature data set. After which, we need to do a sorting for this array and select the first $n$ points with the smallest 2-norm distance. The number of point we select to do LMS is $n$.

Although do LMS for system identification in this way will increase the algorithm complexity because the sorting algorithm needs to be run, especially for the case of time variant model, it is still within the computational capacity of a personal laptop, which can be seen in section 7, CPU time for different models.

6.3 Multi-rate Controller Design

In this section, we will introduce an implementation strategy that can be done for any MPC controllers if we want to increase the controller frequency but not to increase the algorithm complexity.
The idea to do this is simple, which can be illustrated in figure 6.2. In figure 6.2, blue rectangles represent cars. The reason why they are always on the second prediction horizon is that the controller needs time to calculate the required control input. During the time of calculating the control input, the car has already moved one step forward, thus, the input needs to be given to the car is the second control input signal from the MPC controller.

The normal way to increase the controller frequency is to increase the discretization time for every step. However, to keep the predicted distance the same as before, prediction horizon needs to also be increased, thus increase the algorithm complexity a lot.

However, we noticed that we only need the second input signals. We only need to decrease the discretization time for the first time step and keep the discretization time the same for prediction horizon after that. In this way, we increase the controller frequency but did not increase the controller complexity.

![Figure 6.2: Multi-rate controller design](image)
7 Experiment Results

There are two main contributions from this master thesis, one is proposing an efficient and reliable architecture to implement LMPC on autonomous racing, which has been introduced in the previous sections. The other contribution of this master thesis is using system identification strategies from different fields to construct the model used in LMPC controller.

In this section, we will compare the experiment results for LMPC controllers with different models. Besides, we also investigated the influence of different controller setting, such as different prediction horizon and different controller frequency.

Experiment results presented in this chapter were collected from controllers without tuning. The same controller parameters used in simulations were used in experiments. The same parameter setting is used for all controllers.

7.1 LMPC Learning Comparison

In this section, we will compare the learning process of controllers with different models and comment on the result plot.

From figure 7.1, we can clearly see that the controller with nonlinear kinematic bicycle model did not learned from the history. The reason for this is that nonlinear kinematic bicycle model has big model mismatch when the car is going at high speed.

The second observation we can see is that linearized dynamic bicycle model’s lap time even goes higher than path following. This is simply because we did not do any tire model identification. Thus a complex model without accurate parameter is still a bad model.

All the other models can at least learn from the history and the lap time will converge to a smaller value.

For the learning convergence, we can notice that models we identify by LMS with or without lifting are converging to the same lap time. The controller with
linearized kinematic bicycle model with linearization points forecast by nonlinear dynamic bicycle model by LMS converge to another lap time.

The most interesting result is that linearized kinematic bicycle model achieves a better lap time than nonlinear kinematic bicycle model. The reason is that we get linearization points from nonlinear dynamic bicycle model by LMS. Although we did not achieve a lap time as good as the nonlinear dynamic bicycle model by LMS, we still see the potential of this combined model method since it reduce the algorithm complexity and achieves a better lap time than nonlinear kinematic bicycle model. Hence we believe that by other hardware update and improvements, this model has the potential to achieve a better lap time.

From the learning convergence comparison in figure 7.2, we can notice that there are three lap time sudden increment. This was because of package loss and the car went out of the track. However, another interesting observation we can see is that LMPC can recover from hardware failure during the learning process. This shows that LMPC algorithm can work quite robustly to disturbance ot even temporary failure.

If we zoom in the highlighted rectangular area in figure 7.2, we can get figure 7.3. If we compare nonlinear dynamic bicycle model by LMS with lifting and nonlinear
dynamic bicycle model without lifting, we can notice that model identified by LMS with lifting has a more regularized learning process, which shows that with feature lifting, a more accurate model is used in LMPC controller.
7.2 CPU Time Comparison

In previous sections, we have compared the learning results from different models. However, for a real-time implementation, time needed for each time step to solve the problem is another important factor. In this section, CPU time comparison for different models and the reasons for CPU time difference will be presented. The CPU time needed for different models have been summarized in table ???. Different models brings different algorithm complexity to the controller. Some model needs to do run sorting algorithm and online system identification. Some model needs to run Gaussian process regression. Some model are time variant, thus requires more solving time.

**CPU time difference explanations:**

1. Linearized dynamic bicycle model has the lowest CPU time since it is a QP problem.
2. From the comparison of TI/TV, we can see that each LMS takes around 0.0007s because the MPC prediction horizon is 10.
3. Model by LMS with feature lifting is more difficult to solve than model by LMS without lifting because of more nonlinear equations.
4. Gaussian Process Regression is cheap because many matrix can be pre-computed. There is no matrix inversion but only matrix multiplication. This is the reason why GPR did not take much time. However, one thing needs to be noted that, if we need to do GPR for 3 states and prediction horizon, $N = 10$, time for GPR is $3 \cdot (10 - 1) \cdot 0.0003 = 0.0081s$

7.3 Nonlinear Kinematic Bicycle Model and Linearized Dynamic Bicycle Model

In this section, we will present the close loop trajectory of LMPC results with nonlinear kinematic bicycle model and linearized dynamic bicycle model. The close loop trajectory is consistent to lap time comparison in section 7.1. The lap time is high for those two models because the close loop trajectory from those two
<table>
<thead>
<tr>
<th>Model</th>
<th>Iteration time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linearized dynamic</td>
<td>0.01</td>
</tr>
<tr>
<td>Nonlinear kinematic</td>
<td>0.02</td>
</tr>
<tr>
<td>TI linearized kinematic</td>
<td>0.013</td>
</tr>
<tr>
<td>TV linearized kinematic</td>
<td>0.018</td>
</tr>
<tr>
<td>TI nonlinear dynamic by LMS</td>
<td>0.025</td>
</tr>
<tr>
<td>TV nonlinear dynamic by LMS</td>
<td>0.031</td>
</tr>
<tr>
<td>TI nonlinear dynamic by LMS (lifting)</td>
<td>0.03</td>
</tr>
<tr>
<td>TV nonlinear dynamic by LMS (lifting)</td>
<td>0.037</td>
</tr>
<tr>
<td>Gaussian process regression time</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Table 1: CPU time comparison for controller with prediction horizon, $N = 10$

models are really bad due to high model mismatch.

![Image](K1.png)

Figure 7.4: Learning comparison: nonlinear kinematic bicycle model and linearized dynamic bicycle model

For nonlinear kinematic bicycle model, model mismatch increases as the velocity increases. For linearized dynamic bicycle model, model mismatch is high because tire model parameters are inaccurate.

### 7.4 Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS

In this section, we will present the close loop trajectory results from time invariant (variant) nonlinear dynamic bicycle model by LMS. The comparison is focused on
the difference of time invariant model and time variant model.

From lap time comparison in section 7.1, we did not see much difference from the lap time, and from the close loop trajectory comparison, time invariant model and time variant model learned the same close loop trajectory.

Figure 7.5: LMPC: time invariant (variant) nonlinear dynamic bicycle model by LMS

7.5 Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS with Lifting

In this section, we also compare the close loop trajectory learned by time invariant model and time variant model, but the model is identified by LMS with lifting.

The interesting result is that, there is still not much difference between time invariant model and time variant model. But if we compare the result in this
section and the previous section, we can see the difference because of feature lifting.

Figure 7.6: LMPC: time invariant nonlinear dynamic bicycle model by LMS with lifting

Figure 7.7: LMPC: time variant nonlinear dynamic bicycle model by LMS with lifting

7.5.1 Time Invariant (Variant) Linearized Kinematic Bicycle Model

As shown in the previous two sections, we did not see much difference between time invariant model and time variant model. In this section, we compare the learned close loop trajectory by linearized kinematic bicycle model with linearization points forecast by nonlinear dynamic bicycle model by LMS.

Still, we did not see much difference between time invariant model and time variant model.
If we compare experiment results in this section and experiment results in section 7.3 for nonlinear kinematic bicycle model, we see that here with linearized kinematic bicycle model in LMPC controller, we can achieve better lap time and also a more reasonable close loop trajectory than nonlinear kinematic bicycle model. This result shows that, in this way, we can both reduce algorithm complexity and increase model accuracy.

However, the close loop trajectory and the lap time is still not good enough. Possible explanation for this can be state estimation is not good enough, which will influence the quality of LMS by providing a set of inaccurate feature data. So we believe that in some application, we can achieve desirable result with this method.

Figure 7.8: LMPC: time invariant (variant) linearized kinematic bicycle model
7.5.2 Time Invariant Nonlinear Dynamic Bicycle Model by LMS (N=10/15)

LMPC paper, [12] shows that for linear system, different prediction horizons will not influence the LMPC result. In this section, we show that for a nonlinear system, we get the same learned close loop trajectory and thus show by experiment results that different prediction horizon will not influence the learned close loop trajectory in this nonlinear autonomous racing problem.

![Time invariant nonlinear dynamic bicycle model by LMS](image)

Figure 7.9: LMPC: time invariant nonlinear dynamic bicycle model, N=10/15

7.6 Time Variant Nonlinear Dynamic Bicycle Model by LMS+GPR

In this section and the next section, we will show the experiment results when we implemented Gaussian Process Regression (GPR).
We chose time variant nonlinear dynamic bicycle model by LMS as the car model. We use GPR to identify and compensate the model mismatch part. Experiment result is shown in figure 7.10.

![Figure 7.10: LMPC: time variant nonlinear dynamic bicycle model by LMS + GPR](image)

In figure 7.10, we see that when we use GPR to compensate the model mismatch part. Even for the last 5 laps, it did not converge. The reason is that LMS is done online for every time step and the corresponding model identified by LMS is an unbiased model. So the model mismatch part has zero mean and GPR can not help compensate the model mismatch, but instead, introduce more noise, which is what we see in figure 7.10.

### 7.7 Biased Time Invariant (Variant) Nonlinear Dynamic Bicycle Model by LMS+GPR

In last section, we see that GPR did not help when the model is unbiased. In this section, we introduce some bias into the system on purpose, and we want to see if GPR can help when the model is biased. The experiment results are shown in figure 7.11 and figure 7.13. We tried to use GPR on both biased time invariant model and biased time variant model, from which we see the same effect from GPR.

We introduce one more step delay in the feature data used to do system identification (LMS). In this way, feature data is shifted by one step and thus the model identified by LMS is a biased model.
If we compare figure 7.11 and figure 7.6, we can see the difference of introducing this one more step delay. Biased model and unbiased model converge to two different close loop trajectories.

After implementing GPR on this biased model, we can see that the close loop trajectory is corrected back to a close loop trajectory similar to the one we see in figure 7.5.

As for the lap time comparison, results for biased time invariant nonlinear dynamic bicycle model by LMS with/without GPR are shown in figure 7.12. From lap time comparison, GPR gives lower lap time after the convergence and correct the bias we introduced into the model.

In figure 7.13, we show that for time variant model, we see the same result as in time invariant model. It is consistent to the result we have in section 7.1. Result from time variant model also help to enhance the conclusion we got in time
Lap time comparison result for biased time variant nonlinear dynamic bicycle model by LMS with/without GPR is shown in figure 7.14. The same as time invariant model, GPR gives lower lap time and correct the bias we introduced into the model.

7.8 Time Invariant Nonlinear Dynamic Bicycle Model (10Hz/20Hz)

A controller with 10Hz frequency is actually a low frequency controller. It is interesting to see the result when we double the controller frequency. In figure 7.15, we actually see different converged close loop trajectory from controllers with different frequency.

Controller with higher frequency can get feedback in a higher frequency and thus can correct the vehicle states in a higher frequency. In figure 7.15, we see difference in close loop trajectory between the controllers with different frequencies. We believe those two close loop trajectories are two local minimum
solutions for this autonomous racing application because of different controller frequencies. For lap time comparison from 7.16, we did not see much difference, either. The lap time jump was due to communication package loss.
8 Conclusions and Future Work

8.1 Conclusions

We can summarize and conclude some main points from the experiment results we presented in section 7.

1. Time invariant model and time variant model converge to the same lap time and the same close loop trajectory. Experiment results tell us not to use time variant model to do LMPC racing, since it only increase algorithm complexity but not lap time and converged close loop trajectory.

2. LMS with feature lifting gives more regularized learning process and converge to a close loop trajectory more similar to human driver’s racing trajectory.

3. The idea of using linearized kinematic bicycle model in the controller and use model identified by LMS to forecast linearization points can reduce algorithm complexity and increase model accuracy. Thus compared to nonlinear kinematic bicycle model, a better lap time and more reasonable
Figure 7.15: LMPC: time invariant nonlinear dynamic bicycle model by LMS with lifting (10Hz/20Hz)

4. Prediction horizon will not change the converged close loop trajectory. But experiment results of extreme short or extreme long prediction horizon are remaining to be shown.

5. Gaussian process regression (GPR) will help when the model is biased and has a non-zero mean model mismatch. And GPR will make the learning worse if the model is unbiased and has a zero mean model mismatch part.

6. Higher controller frequency can also help. But if there is not so much computational power available, lower frequency can also give a satisfied learning result.
8.2 Future Work

Based on the experiment result we presented in section 7, there are still some points that can be improved.

1. In the work of this master thesis, we show that linearized dynamic bicycle model did not work well with wrong tire parameters. But we did not spend time on tire parameters identification. So those tire model parameters are still worthy to be identified and see the performance of that model.

2. In the multi-rate Kalman filter, we use kinematic bicycle model in the filter. However, better model such as dynamic bicycle model can be used to improve the state estimation. So with tire parameters available, this can be tried and to compare the state estimation.

3. For state estimation, higher frequency and higher resolution sensors can be tried to see how much we can improve by buying more expensive sensors. It would be better if an experiment can be run to know the ground truth for those states we want to estimate.
References


