Model Identification and Control of Autonomous Ground Vehicles

by

Sina Aghli

B.A., Azad University, Iran, 2009

M.S., University of Tabriz, Iran, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2018

This thesis entitled:
Model Identification and Control of Autonomous Ground Vehicles
written by Sina Aghli
has been approved for the Department of Computer Science

_____

Prof. Christoffer Heckman

_____

Prof. John Hauser

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Aghli, Sina (Ph.D., Computer Science)

Model Identification and Control of Autonomous Ground Vehicles

Thesis directed by Prof. Christoffer Heckman

Autonomous Ground Vehicles (AGV) are mobile robotic platforms used in variety of applications to execute tasks which could be dangerous for humans to operate. Recently, autonomous cars are discussed in carrying passengers from point to point without human interaction. Sophisticated controllers are required to operate autonomous vehicles while responding to both normal and hazardous driving conditions. Dangerous conditions which might be easily perceivable by sensors in the system require controllers that can readily benefit from the new sensory information. In this thesis, we address this problem by asserting that the design of controllers and corresponding calibration and local planning methods are required to quickly adapt to changes in both the dynamic model of vehicle as well as changes in environment. A full pipeline of calibration, local planner and model predictive controller has been developed and tested in simulation and on physical platform. Properties of a high fidelity model and a simpler model has been studied and their pros and cons has been discussed. Also, a calibration algorithm has been developed to calibrate parameters of dynamic models based on informativeness of robot's motion. Next, A local planning algorithms has been developed to plan vehicle's reference path between consecutive waypoints and finally a model predictive controller has been designed to stabilizes the vehicle to the reference path. A theoretical proof for stability of proposed controller is given. One of the goals behind this work has been design of an adaptive method in a sense that system can quickly adapt to changes in robot's model or environment.

Dedication

To love of my life Mahsa and best of parents Mahdi and Rana

Acknowledgements

I express my sincere gratitude to my advisor Prof.Christoffer Heckman for the continuous support of my Ph.D study, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research. I am especially indebted to Dr. John Hauser for thought-provoking and insightful discussions about this work.

My sincere thanks also go to my undergraduate mentor Mehdi Imani, who taught me the art of critical thinking which has always helped me in my research.

Most of all, I would like to thank my wife and family for supporting me spiritually throughout hard times during my PhD study.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

1.1 Autonomous Vehicle Control

Sophisticated controllers are required to operate autonomous vehicles while responding to both normal and hazardous driving conditions. Dangerous conditions which might be easily perceivable by sensors in the system require controllers that can readily benefit from the new sensory information. **In this thesis, we address this problem by asserting that the design of controllers and corresponding calibration and local planning methods are required to quickly adapt to changes in both the dynamic model of vehicle as well as changes in environment.** Furthermore, these dynamic changes can be directly perceived via unique and well-tailored sensors on-board the vehicle.

Controlling a vehicle is about answering the question of "Given the state of vehicle what actions it needs to take in order to reach a desired goal?". Answer to this question depends on many parameters in the vehicle itself as well as the environment it interacts with. Different methods has been developed in order to solve this problem where each can have its own benefits depending on the performance measures that are expected from the system. As a human we deal with this task in a daily basis, when we learn to drive a car our brain learns a complicated function which can perceive the environment such as where we are located in the map, where we are located with respect to road, road surface condition, signs and lights around the road and also vehicles and pedestrians around us. Once we have a good perception of the environment we make a decision on how to press the gas/brake pedals or how much to steer the vehicle such that we reach the goal

successfully. Having a good perception of the environment always helps us take better actions but if we don't have a good understanding of the environment (lets say driving in a foggy/snowy day) then we adjust our driving behavior to the condition. In another perspective, our driving skills improve as we practice more, which means that our perception improves by being more aware of our environment and also we learn how vehicle responds to braking or steering changes. When it comes to rally motorsports, drivers train themselves to use their vehicle in the edge of its performance. They learn how much they can steer the vehicle before it slides and gets out of their control. An example which shows importance of learning vehicle dynamics can be seen in a video of the professional rally driver Ken Block where he is asked to drive a different vehicle [**?**] with a welded back differential (not common in rally vehicles) where he doesn't have a good control of the vehicle and hits most of the path cones on the track.

When it comes to automatic control of autonomous vehicles, we have the advantage to study and understand the vehicle dynamics and design algorithms which can control the vehicle even in its edge of performance. However, if uncertainties in the perception method are not considered, then resulted actions can have catastrophic events.

In majority of proposed solutions for this problem, vehicles dynamics is considered on its own in order to control the vehicle and environmental effects are assumed as disturbances to the model but this assumption can put the vehicle in a state which might not be able to recover from due to actuator saturations or other vehicle dynamics limitations. A vehicle's dynamics not only depend on the vehicle itself but also condition of its environment it interacts with. For this purpose a solution which can take in the environment measurements when available is necessary.

Perception methods in the first stage of an autonomous vehicle are concerned with finding state of the vehicle and its environment based on the measurements from set of sensors on the platform. Simultaneous localization and mapping (SLAM) techniques are capable of finding location of robotics platform based on measurements from laser scanners, cameras and IMUs. These methods can also output a geometric map of the environment. Data from multiple sensors can be fused in order to improve accuracy of the perception pipeline. In the vehicle model level, different sensors

can be used to measure/estimate some internal states of the dynamics. Methods like Kalman filter and its extensions, extended kalman filter (EKF) and unscented kalman filter (UKF) has been widely used for linear and nonlinear systems in order to get an estimate of internal state of dynamic model. All these methods help with having a reasonably accurate understanding of current state of model and its environment which can result in better performance of future stages in controlling an autonomous vehicle.

1.2 Modeling

Both the perception and control methods depend on model of the vehicle in order to perform their tasks. A model is basically a mathematical function which represents behavior of the actual platform for some input signal. The depth in which one might want to model a physical system depends on the questions that one might wanna ask from the system. For example if a vehicle is supposed to drive in slow speeds where there is no sliding of wheels then a simple kinematic model of the system is accurate enough to show how vehicle moves based on some steering and acceleration inputs. On the other hand, if vehicle is supposed to drive in snowy roads and higher speeds then vehicle's tire model has more effect on how the system behaves and it should be included in the model. Since autonomous vehicles are designed to operate in outdoors, their model needs to be capable of showing how the physical vehicle would behave in different road conditions and using simple models for both perception and control methods can result in taking wrong actions.

1.3 Calibration

Once a model is designed, It contains some parameters which need to be calibrated. Calibration is a major part of modeling effort since it can result in miss-prediction of system's behavior. Depending on the model, this task can be very cumbersome and might require doing extensive tests in a lab environment. Even if one is capable of achieving a good calibration, parameter values can change as the vehicle operates. Environmental effects like air temperature and vibrations applied from road can drastically change the model parameters. This raises the requirement for a

calibration pipeline which can continuously run on the vehicle and estimate the parameters as they change.

1.4 Layout

The primary contribution of this thesis is sketched by the discussions above and in following chapters, detailed discussions of each part is given.

Chapter 2 give an overview of available methods which have been previously used in approaching the problems above and makes a comparison between different methods and my personal judgment on how the solution needs to be approached. This conclusion has shaped the research direction of this thesis and following chapters.

Chapter 3 introduces the mathematical framework for rest of the thesis and some of the algorithms which are frequently used through rest of the chapters.

Chapter 4 discusses details of a robotic platform (Parkour Car) which has been designed to be used in experiments of this research and multiple other researches. Parkour Car is frequently used in experiments as the test platform in Autonomous Robotics And Perception Lab of University of Colorado Boulder.

Chapter 5 gives a detailed explanation of a calibration pipeline which calibrates parameters of the model based on motion of the vehicle and can be used with any dynamic model. This method doesn't suffer from calibration result getting biased toward a group of parameters and runs in constant time.

Chapter 6 discusses a new approach of constructing a models predictive controller which can use a physics engine as the underlying model which can take in effect of contact forces applied from environment map to the vehicle model. This chapter also give a theoretical convergence proof for the proposed method.

Chapter 7 introduces a new analytical model for a four wheel drive vehicle and discusses achievable performance gains when it is used with MPC method of chapter 6.

Chapter 8 concludes this thesis by summarizing the contributions and proposing a future

research direction.

Chapter 2

Background and Literature Review

2.1 History

Ground vehicles has been used for transportation since 18th century but advancement of technology in transmitting wireless signals brought the idea of controlling different mechanical platforms from distance. The first unmanned remotely controlled vehicle dates back to 1926 where Houdina Radio Control demonstrated an unmanned vehicle equipped with electric motors attached to vehicle controls and a person controlling the car with radio signals transmitted from a car following the driverless vehicle [127]. At that point driver less cars seemed to be more of an advertisement tool for car companies and other companies working in tech fields. They would mostly be controlled by a remote operator or by adding electric lines to the road bed which would vehicle the vehicle by following the magnetic field. It wasn't until 1960s which universities and car making companies started thinking of developing self driving vehicles as commercial products [127]. Solutions like embedding electronic devices in roadway were proposed but soon these solutions turned to be very expensive if not impossible. By 1980 silicon based sensors and processors had enough advancement to be used in sensing the environment and making decisions. Mercedes-Benz developed its first vision-guided robotic van and achieved a speed of 39 miles per hour on streets without traffic. This vehicle used four cameras to perceive the environment. Analog video feeds were routed to A/D s and then to 60-70 processors to do the road lane detection, obstacle detection, signal light detection and also calculating controls for keeping the vehicle in the road. This attracted attention of more companies and universities to join the competition.

At the mean time AGV's were designed with similar technologies to carry equipment or heavy loads in factory environments which helped reducing space required for maneuvering [126]. Different types of sensors and stronger processors helped accelerate development of this technology. Technology of self driving vehicles got very hight attention with introduction of DARPA Grand Challenge in 2004 where competing teams were expected to design autonomous vehicle capable of driving a 150 mile route without human intervention. None of the robot vehicles made it to the finish line at the time. This contest was held for another year, but this time multiple teams finished the contest. In 2007 Urban Challenge was hold where self-driving vehicles had to drive in an urban environment while obeying street rules. Last challenge was a success and that jump started self-driving cars. AGV's had also been used in exploring volcanic sites [119] or explore other planet surfaces [110].

Autonomous part of all these platforms depend on two major parts. First, robots need to be aware of their environment. This could be the road condition for a self-driving car, position of a magnetic stripe on the floor of a factory environment or stiffness of a hill for a mars rover to ride on without getting stuck. Second part is, given measurements from environment what actions are needed to be taken such that the robot achieves a predefined goal. In this paper, literature review of the later part is discussed and assumptions under which the proposed techniques work has been represented. In section two, different modeling techniques which have been used for AGVs is discussed. In section three, different approaches of controlling AGVs inputs and their pros and cons is presented. Finally, a comparison of Models and also Controllers is made to show advantages of some class of underlying techniques.

2.2 Models

Modeling different dynamical platforms depends on questions one might want to ask about the system. Meaning that, what are our expectations from the system under study? for example, behavior of a car operating in nice weather condition in slow speeds on flat streets might not be very different than behavior of a unicycle but if the same car is supposed to operate on frozen street beds

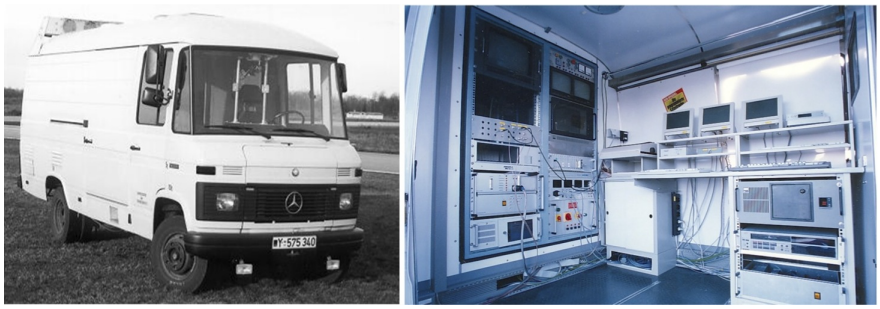

Figure 2.1: VAMORS [74] was one of the first self driving vehicles designed by Ernst Dickmanns and his team in late 1980. This van used two front looking and two back looking cameras and about sixty processors to drive the vehicle on traffic less street.

it would show significant difference compared to a unicycle. In this section different approaches in modeling AGVs under proposed conditions of operation is discussed. Models are generally divided into two categories, kinematic models and dynamic models. first category is generally used in applications where AGV is expected to move in low speeds and kinematic constraints of the system are good representation of its general behavior, but in cases where forces applied to the body of AGV has significant affect on its behavior then the second category is considered.

2.2.1 Unicycle Model

A simple way to model behavior of an AGV is to represent it as a single non-holonomic wheel with a velocity and torque and a steering signal directly applied to the wheel [73]. In this simple kinematic model as well as other kinematic models its assumed that wheel will roll without slipping. kinematic unicycle model is presented as follows

$$\begin{aligned} \dot{x} &= v sin(\theta) \\ \dot{y} &= v cos(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{2.1}$$

where x and y are coordinates of center of AGV in two dimensional plane, θ is heading angle, ω is steering rate input and v is linear velocity input. Becker *et al.* [10] use a unicycle model in designing an ensemble controller which globally stabilizes the robotic platform to given coordinates.

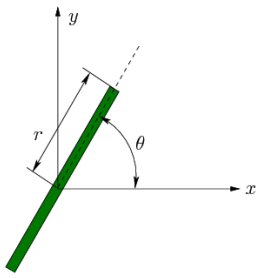

Figure 2.2: Unicycle kinematic model. Forward linear velocity v is normally modeled as average of left and right wheels rotational speed multiplied by wheel radius r. figure from [66].

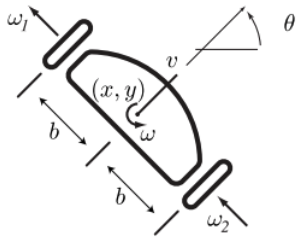

Figure 2.3: Differential drive model assumes two wheels on sides of AGV with speed inputs on each wheel. figure from [9].

Aicardi *et al.* [4] give a "polar-like" description of this model in order to prevent brockett's [17] negative result and design a time invariant control law for asymptotically stabilizing the model. Kanayama *et al.* [54] finds a parameterized Liapunov function and a control law accordingly which shows global stability of system which also is capable of assessing if system can be under-damped with some set of tuning parameters. The controller is also experimentally tested on Yamabico-11 robot. lee *et al.* [69] show that this model can be used in problems with saturation constraints and they propose a control law which globally stabilizes the AGV to any path specified with strait line, circle or a path approaching the origin using a single controller. Their experiments also show that this can be used in applications like parallel parking. Benefits of using unicycle kinematic model could be its low number of state variables and less complications in the model which allow design of asymptotically stable controllers but this is under the assumption that system avoids any side slips and is not affected with forces generated from uneven surfaces which vehicle can drive on.

Kinematics of robot can also be modeled as a differential drive system (Figure.2.3). In this

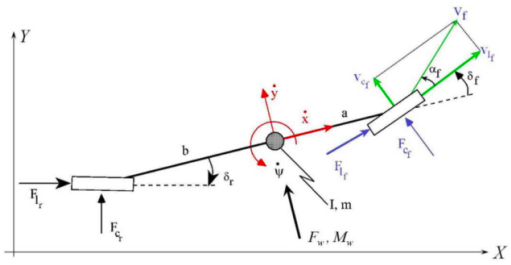

Figure 2.4: Single track (bicycle) model schematic. Red axes shows the center of mass AGV frame axes. Figure from [35]

model its assumed that two wheels on two sides of chassis apply linear and rotational velocities to the chassis. Chassis is assumed to be constrained (maybe with passive wheels) such that it is always parallel to ground plane. In most cases differential drive models are transfered to unicycle model by averaging rotational speeds of both wheels and to get a single wheel speed and a rotational speed of $\omega = \frac{1}{2b}(v_r - v_l)$ and the model is again simplified to the Equation.2.1 form. Chwa [23] uses this method to model AGV and designing tracking control using a feedback linearization method, Indiveri *et al.* [49] uses this model with bounds on wheel speeds to design a path following controller. This models also has been used in path planning approaches which require a simplified model due to computational complexity of planning algorithms [9].

2.2.2 Single Track (Bicycle) Model

Single track or bicycle model is one of the favorite models for both controls and planning applications. Single track models make assumption of driving on a plane (2D) and are used in both kinematic and dynamic form. Figure 2.4 shows a schematic of this model where a back wheel is mostly responsible of applying forward acceleration without steering while front wheel is capable of steering but it is a passive wheel. [20, 65, 39, 68, 48, 35]. Similar to unicycle case kinematic bicycle model can be used in kinematic form with neglecting AGV's mass but this model would not be able to account for wheel drifts. Peppy *et al.* [94] make a comparison of kinematic vs dynamic bicycle models for planning purposes. Kinematic model is also referred as non-holonomic constraint since wheels are not allowed to move side ways (drift), however adding mass to the body

of vehicle and wheels, requires the model to have side forces applied to the wheels. Kritayakirana *et al.* [65] show a bicycle model in which tire dynamics discover the amount of forces that wheels can generate. These forces can result in calculation of maximum amount of accelerations that a vehicle can create based on tire properties. Rucco *et al.* [106] uses a single track model with both a simplified linear (relation between lateral force and side slip is linear) tire model [79, 41, 37] and a Pacejka's magic tire model [90, 8] to extract vehicle's equilibrium manifolds. Other tire models also has been proposed at [28, 18, 89]. Falcone *et al.* [35] design a MPC controller based on model shown at Figure 2.4 which is capable of following trajectories on slippery roads at the highest possible entry speed. This model has also been used in linearized parameter varying form under stochastic disturbances to design robust trajectory tracking controllers [20].

The dynamic single track models has also been used in planning algorithms. Hwan Jeon *et al.* [48] use a dynamic version of this model in order to implement a RRT* based optimal motion planning algorithm to be used in high-speed driving. Lee *et al.* [68] use this model in design of a planning algorithm for parking assist of a vehicle.

Mostly when modeling dynamics of an AGV, geometric dimensions of the chassis of AGV are not considered. This problem is not studied very well but in some researches collision avoidance has been included in the system by considering bounding boxes around the model and adding constraints to the problem which avoids collisions between convex shapes [20].

Single track model seems to be the most popular model when it comes to modeling behavior of an AGV since it has a good balance between model complexity and accuracy.

2.2.3 Tricycle Model

This model assumes two back wheels where each wheel could have different velocity and torque and a single virtual steering wheel in front which is assumed to have the average steering angle of each left and right front wheels (Figure 2.5). Kodagoda *et al.* [62] use a dynamic model of tricycle to design a fuzzy PD-PI controller to control the AGV on a reference trajectory. Bom *et al.* [13] propose a controller in order to move a set of vehicles in platoon fashion and experimental

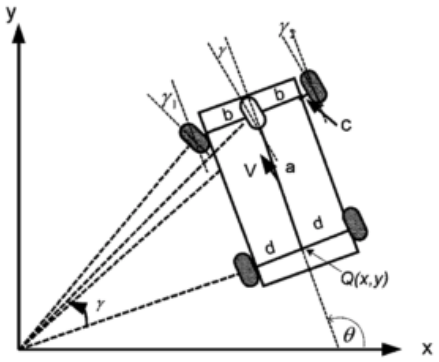

Figure 2.5: Tricycle model of an AGV. Figure from [62]

platforms are modeled as tricycles. They also show that this non-linear model can be converted into a so called chained form [109] which has a decoupling effect on state variables and hence two global separate lateral and longitudinal controllers can be designed for the vehicle platoon.

Tricycle models in general does not seem to have any more benefit compared to bicycle model explained in section 2.2.2 which explains non popularity of this model.

2.2.4 Two Track Model

Double track model is one of the most complete models of a four wheel AGV where all four wheels are considered in the model. Center of gravity of the vehicle chassis is normally considered in equations of motion of the vehicle as well as tire dynamics. Rucco *et al.* [105] propose a reduced order two track model of a car vehicle includes lateral and longitudinal load transfer which captures key features of vehicle dynamics for aggressive maneuvering of complex cars. Even though their model doesn't include suspension dynamics but this effect is modeled by applying normal forces from all four contact points with the road to the chassis in the model. In another paper from Rucco *et al.* [107], a two track model with tire dynamics and lat/long load transfer has been used to find a trajectory of the car minimizing the traveling time subject to steering and tire limits. in addition a constraint has been applied so that tires always keep contact with road surface.

Gao *et al.* [39] uses a two track model in order to formulate an obstacle avoidance problem

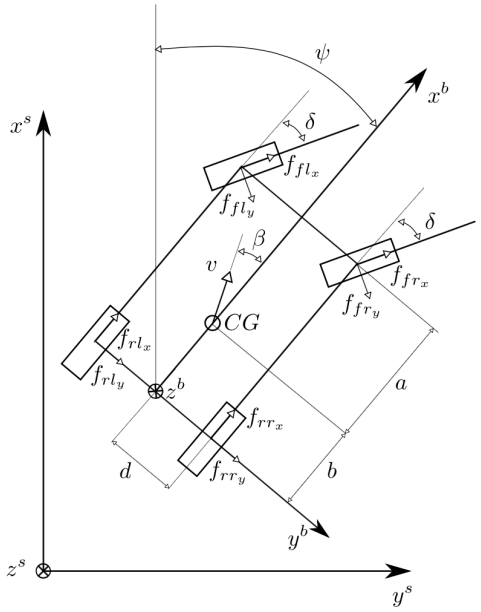

Figure 2.6: Two Track model of an AGV. Figure from [105]

for semi-autonomous ground vehicles. Tire models are formulated as in [89] and calculated tire forces are applied to the chassis as shown in Figure 2.6. This model is later used in design of a controller in MPC fashion to avoid obstacles on the path of vehicle.

When it comes to planning and control of cars close to limits of its dynamics it seems that two track model has the closest behavior to the actual system. However these models are still based on the assumptions that vehicle is driving on a flat road and normal forces from road surface do not change very frequently.

2.2.5 Lateral/Longitudinal Dynamics

In some cases lateral and longitudinal dynamics of the AGV are considered separately in order to achieve specific goals. Longitudinal dynamics of the car can be used in designing a cruise controller [95] and lateral dynamics of the vehicle can be used in design of controllers to do automatic lane keeping [95, 96] . Sierra *et al.* [114] uses lateral dynamics of the of the vehicle in order to estimate cornering stiffness of the vehicle. . In some cases vehicle might be modeled as single track

model as explained in section 2.2.2 but this model can be linearized around its longitudinal speed in order to be able to take benefit of linear controllers to stabilize the system [20].

System dynamics need to be decoupled in order to be able to treat lateral dynamics separate from longitudinal dynamics but as discussed above in cases where one part of the model (ex. lateral dynamics) is not affecting the behavior of the other part much (ex. longitudinal dynamics) then they could be modeled separately. This method seems to be effective only in designing controllers which are linear and locally stable in most cases.

2.2.6 Indirect Modeling

In this section we discuss more modern methods which do not require writing newtonian equations of motion in the sense described in previous sections. These approaches are more investigated by computer science robotic community.

2.2.6.1 Data Driven Approaches

Accurate system modeling might not be easily possible and data driven approaches are concerned with creating models by capturing input/output data of the platform and regressing the data to an specific function. Moon *et al.* [81] design a adaptive cruise control system which includes human behavior in the model. To achieve this model they capture human inputs for speed adjustments and some states from vehicle and they fit a first order and a second order regression model to the captured data. This model is later used to adjust vehicle's cruise controller response such that it feels natural for the human driver.

Markov decision processes (MDPs) have also been used in order to model stochastic system dynamics using their input/output data. Bethke *et al.* [11] proposes an online method for modeling behavior of the system and adjusting the system control policy and shows its benefits over static/offline approaches. Even though the approach by Bethke has been tested on a surveillance airplane platform but it is applicable to AGV systems too.

Reinforcement learning as well as deep learning methods has been one of the choices for

modeling and control of different robotic platforms in last couple of years among AI and robotics researchers. Riedmiller *et al.* [104] shows a Q-learning based approach in controlling a vehicle. Michels *et al.* [78] show that a reinforcement learning method can be used in order to control a model car at high speeds through unstructured outdoor environments by proposing a method in which, supervised learning is used to estimate depth from a monocular camera and later directly learns controls policies using policy search method in a simulated environment. In this method control signals can be directly calculated from vision system output. In some methods input/output model and corresponding control signal can be learned to a gaussian process (GP) function [97].

In one effort, Bojarski *et al.* [12] show that raw image pixels from single fron-facing camera can be mapped directly to steering commands using a convolutional neural network (CNN). An autonomous car can drive using this end-to-end method with minimum data from humans in traffic on local roads with or without lane markings and on highways. Similar approach has been used in [92].

Other methods with combination of deep learning and reinforcement learning has also been proposed [70, 32], in which some even show human level performance [80].

Methods explained here are mostly capable of including some more information from the environment and in end-to-end control case the perception and local planning (or maneuver regulation) task is abstracted in the process. Almost all of these methods would not have any prior information about structure of the model under study and they get better at modeling or controlling the robotic platform as they gather more input/output data pair. Another problem with these approaches is that non of the parameters of actual system directly correlate to the model parameters in the underlying model. That means that if there are any changes in system dynamics, then the previous model needs to be scratched and a new model needs to be learned. One other concern with these methods is lack of theoretical guarantees. Even if a large amount of data is used in order to learn a model, if state of the system falls in a region which there hasn't been enough data to train the model then model might show a completely different output compared to actual system.

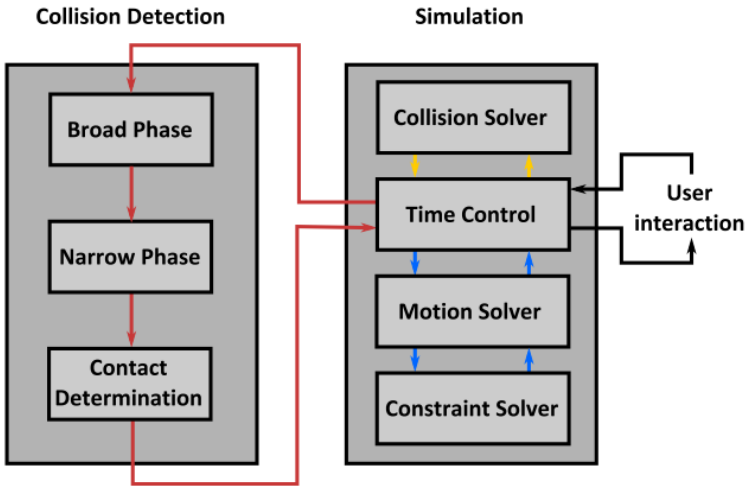

Figure 2.7: Universal architecture of a physics engine according to modular design described in [34, 116]

2.2.6.2 Physics Engines

Physics engine is a name mostly used in computer game industry which refers to simulating behaviors of objects in computer games. They normally use simplified mathematical models and algorithms to mimic behavior of an actual objects in a 3D world. These algorithms are mostly optimized to run in very high speed on consumer computers and have a realistic simulation results.

The actual design of different physics engines would obviously differ but a generic architecture of a physics engine is given in figure 2.7. There are generally two main modules in this model, A collision detection module and a simulation module. Collision detection (CD) module is concerned with geometries of objects and simulation module is concerned with mathematical modeling of motion.

Bullet physics engine [25, 24] is an open source library which does contact modeling as well as other newtonian dynamics calculations and is used both in games and robotic applications. Keivan *et al.* [57] samples steering control signal at a given state and uses the model designed by this simulation library in order to generate trajectories of vehicle and later chooses the closest control candidate based on some heuristic. Using this method they are capable of driving an actual model car with agile motion trajectories through a rough terrain. Mujoco [120] is another example of

physics engines which has been used for modeling high speed ground vehicles. Williams *et al.* [130] uses a physics engine in order to sample control signals and simulate thousands of motion samples in short horizon to adjust vehicles behavior accordingly. Other researches use this physics engine as underlying dynamical model aswell [129, 128, 131].

Stkepien [116] has detailed explanation of underlying techniques for most of these methods in his thesis which interested reader could refer to. Also a detailed review of simulation tools for model-based robotics between four different physics engines has been done in [33].

2.3 Controllers

In this section different control methods which have been used in stabilizing autonomous ground vehicles is discussed.

2.3.1 Linear Controllers

Linear control techniques are mostly used methods in industry in order to stabilize dynamical systems [16, 52] but they are used on systems with linear dynamics or in linearized models around an specific point in state space of system. In the context of autonomous ground vehicles, there aren't many applications of using this method since dynamics of a ground vehicle can be highly non-linear however in a work by Divelbiss *et al.* [31] they design a linear quadratic regulator (LQR) controller by linearizing vehicle dynamics around a given trajectory and in another research, Normey *et al.* [87] designs a PID controller for a mobile robot for path tracking application.

Linear controllers like [31] are normally stabilizing in a small neighborhood of the reference trajectory and if there are large perturbations of state then there is no guarantee on converging back to the state. Raksincharoensak *et al.* [96] uses a two wheeled linear vehicle dynamics in order to design a lane keeping controller. The controller is designed to to keep the vehicle in between lanes while there are no stability guarantees.

Linear controllers with local stabilizing properties can be useful in some cases. Tedrake *et al.* [118, 117, 102] designed LQR trees algorithm in which it synthesizes many locally stabilizing LQR

controllers to cover some bounded region of the state space. These local controllers are stored in a table and depending on the state of the system a local control gain is used from the table. This method despite being powerful and generic has some disadvantages. System requires high amount of processing in order to synthesize the controllers and model changes would require the algorithm to recalculate all local controllers from scratch.

2.3.2 Non-Linear Controllers

Non-Linear controllers are concerned with stabilizing non-linear dynamics of the system without any linearization. These techniques can result in a very powerful feedback law in the sense that they stabilize the system in whole space. But this comes with some drawbacks. It is very difficult in general to find a stabilizing control law for non linear systems. for this purpose some specific techniques has been developed for some class of non-linear systems [60]. Direct feedback control laws for autonomous ground vehicles has been designed for very simplified models. Kanayama *et al.* [54] uses a kinematic model of the vehicle similar to the one in section 2.2.1 and defines an error function between the vehicle and the path and comes up with a tracking control law using a Liapunov function [123].This controller will be stabilizing as long as the vehicle stays in the limits of the model used to design the controller.

2.3.2.1 Feedback Linearization

Feedback linearization is another techniques used in nonlinear control design. This approach involves design of a function which transforms non-linear system into an equivalent linear system through a change of variables and maybe choosing different set of control inputs. Once a control input transformation function is developed then a linear control strategy can be used to stabilize the underlying model. This method has mostly used in SISO systems but MIMO versions of it also exist. Chwa [23] uses this method on a differential drive model (similar to Figure 2.3) to design a non-linear controller for a non-holonomic vehicle. Feedback linearization method tries to stabilize the system by canceling non-linear dynamics of the system and a controller designed by this method

might fail if the underlying controlled system doesn't match the assumed model. To eliminate this method there has been more advancement in developing robust feedback linearization techniques [43, 6] but they haven't been used in context of autonomous ground vehicles.

2.3.2.2 Optimal Control Methods

Optimal control theory has techniques for both linear and non-linear dynamics. One of the major non-linear optimal control methods is the receding horizon approach. In this approach at every given state, a set of future states are considered to have some convergence to the goal state. This horizon can be limited or infinite size. finding an infinite horizon control law which results in a feedback law is not an easy task (opposite of LQR controller for linear dynamics). For a limited horizon however an optimization problem can be constructed where control signals for all future steps up to horizon, get calculated and only the first value is actually applied to the system at current time and rest of the controls are discarded. In next time step a new cost function for the optimization problem needs to be constructed and optimized. It should be mentioned that another name for this method is model predictive control (MPC). Model predictive controllers in general has higher processing cost since they need to solve an optimization problem at every time step. Due to this problem this method was normally used in SISO systems which did not require high bandwidth on control loop. Chemical reactions were one of the major systems which took benefit of these controllers and other systems could not use this method due to low speed of processors in the past. A complete review of using receding horizon approaches before year 2000 is given at [76]. However, with the development of faster processors, this method could be used in robotic platforms in realtime.

Different formulations of MPC are proposed for autonomous ground vehicles. Gao [39] formulates a MPC problem which uses non-linear dynamics in section 2.2.2 and shows that the optimization problem can be solved with a sequential quadratic programming approach. Their proposed solution also includes obstacle avoidance which is very unique. In another similar work, Carvalho *et al.* [19] designs an MPC controller which can track a center line in a lane while avoiding collisions

with obstacles. In this method nonlinear dynamics of the system are linearized around states and inputs at each iteration and a solution for the optimization problem is calculated. Falcone *et al.* [35] proposes two MPC formulations in which one uses the nonlinear model and other one uses a local linearized LTV model in order to calculate control signals. This paper shows effectiveness of both method but high computational costs of nonlinear method has been concerning.

In order to have a robust controller, some stochastic predictive control techniques has also been developed. Carvalho *et al.* in [20] develop a chance-constrained MPC problem where the trade-off between risk and conservativeness is managed by a risk factor. In this method a linear time varying model of vehicle is formulated and calculated online. Proposed method also includes a probabilistic model of the environment. In order to avoid collisions a signed distance function is considered which reports existence of collisions between rigid bodies. This method avoids collisions similar to [39, 19] but no collision forces are calculated in the process.

2.3.3 Learning to Control

Learning based approaches rely on systems input/output data in order to learn behavior of the system and perform actions. simple neural networks has been one of the favorite systems to learn a control action up to year 2000s [42, 71]. But with progress in machine learning techniques new class of network based controllers were designed.

Machine learning approaches are amongst this group of techniques. Reinforcement learning has been one of the methods in which the algorithm tries to learn control actions from a human controller while maximizing a reward function [88, 78, 97] . Another approach is to use a deep neural network to learn a function which generates desired control signals base on current state of AGV. Pan *et al.* [92] represent a network which can learn a CNN by training the network over large amount of input/output data and system is capable of controlling a vehicle directly from camera images.

A mixture of these methods also exist in which a deep network has been used with a reinforcement learning approach where the network replaces the value function in the reinforcement

learning pipeline [70, 80] .

These methods do very well when it comes to making control decisions based on measurements from environment and they are capable of controlling very non-linear systems. One of the downsides for these systems might be that they require a large amount of data to train themselves. When it comes to processing cost, these methods require stronger processors but they are also highly parallelizable and can be used in realtime setting if implemented on multi-core computers.

2.3.3.1 Fuzzy Controller

Fuzzy control logic seems to be one of the control methods used in stabilizing autonomous ground vehicles around early 2000s. Kodagoda *et al.* [62] designed a PD-PI controller for steering and speed control. Kodagoda, claims that using fuzzy logic to synthesize the controller, facilitates implementation of a controller while guranateeing stability and uncoupling steering control from speed control. Speed controller itself consists of acceleration controller and braking controller. Even though a Quasi-Lyapunove stability prove is given for every mode of the controller this system eventually implements many local PD controllers which many gain parameters which needs to be tuned. In another work Naranjo *et al.* [85] design a fuzzy logic steering controller for a car to execute an overtaking action in the road.

In some cases, combination of different methods in this section has been researched. For example, Dai *et al.* [26] design a fuzzy controller for a vehicle but the tuning task is done through a reinforcement learning. Or Lin *et al.* [71] design a controller by combining neural networks and fuzzy controller.

Although alternative approaches such as genetic algorithms and neural networks can perform just as well as fuzzy logic in many cases, fuzzy logic has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. This makes it easier to mechanize tasks that are already successfully performed by humans [93].

2.4 Conclusions

Each of the techniques discussed in this paper can be the best choice depending on the conditions and expectations which one might have from the system performance. Some prose and cons of each part is given at each section but in order to conclude ask some questions where answering those has helped me choose a research direction.

In table 2.1 a comparison between models discussed in section 2.2 is given. In this table we try to assign a score for each model based on following questions (lower scores are better).

- Question 1: Ease of modeling? (0:very easy, 6:very hard)
- Question 2: Possibility of including environment map in model? (0:possible with fine details, 6:not possible)
- Question 3: Ease of debugging model behavior? (0:very easy, 6:very hard)
- Question 4: Processing power required for trajectory calculations? (0:very low, 6:very hight)
- Question 5: How static the model is? meaning if something changes in the structure of actual robot how hard is it to adapt the model (0:easily adaptable, 6:very static)
- Question 6: Ability to model contacts and contact forces? (0:very capable, 6:not capable)

Table 2.1: Modeling method comparison table

Model / Question	Unicycle	Single Track	Two Track	Lat/Long	Data Driven	Physics Engine
Q1	4	4	6	6	0	0
Q2	6	6	6	6	2	0
Q3	3	4	5	5	6	1
Q4	0	0	1	1	5	6
Q5	5	5	5	5	6	0
Q6	5	5	5	5	6	2

Scores in Table 2.1 are my personal judgment based on hundreds of papers that I have studied through past four years of which some has been referenced in this paper. Also one should keep in mind that questions 1-5 are my personal concerns when it comes to modeling and control of robotic platforms. Looking at the table 2.1 we can see that Physics engines can do better in general compared to classical modeling and machine learning based approaches in the scope of questions 1-5, However it might suffer from not being able to use it in a real-time pipeline.

Now we do a similar comparison for the control methods discussed in section 2.3. The comparison scope is based on following questions and scores are given based on this review and my past experience with each system (lower scores are better).

- Question 7: Runtime complexity of method? (0:very low, 6: very high)
- Question 8: Capability in handling nonlinear dynamics of system? (0:very capable, 6: not capable)
- Question 9: Ease of designing the controller for a given model? (0:very easy, 6: very hard)
- Question 10: How static the controller is? meaning if there are changes in the model how hard is it to adapt the controller to new condition? (0:easily adaptable, 6: very static)
- Question 11: Possibility of growing basin of attraction of controller? (0:easily, 6: very hard)
- Question 12: Any theoretical stability guarantees? (0: Exists, 6: no stability guarantees)
- Question 13: Actuator saturations can be included? (0: easily, 6: not possible)

Table 2.2: Controller comparison table

Question \ Controller	Linear	Synthesized	Non-Linear	MPC	Learning Based
Q7	0	1	0	6	5
Q8	6	0	0	0	2
Q9	3	5	6	1	4
Q10	6	6	6	0	6
Q11	6	1	6	1	3
Q12	0	0	0	0	6
Q13	3	3	3	0	6

Looking at Table 2.2 we can conclude that for the scope of questions which were asked MPC controller is a better option for control of autonomous ground vehicles (based on question 7-13). MPC approaches suffer from high computational cost.

As it appears, combination of physics engine based modeling and MPC can result in a very strong control method but since both of them suffer from high computational cost, It might not be practical to use them in realtime fashion.

In chapter 6 we show an implementation of MPC method where a physics engine is used as underlying model of vehicle and we discuss challenges related to using this method and later in chapter 7 we design an analytical model of the vehicle which shows effectiveness of proposed MPC method and can run in realtime.

Chapter 3

Mathematical Framework

3.1 Rigid Transformations

A rigid transformation between two 3Dof frames can be represented by a 4×4 matrix operating on homogeneous vectors

$$\begin{pmatrix} \overline{x} \\ \overline{y} \\ \overline{z} \end{pmatrix} = \left(\begin{array}{ccc|c} & \boldsymbol{R} & & \boldsymbol{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3.1}$$

where $\boldsymbol{R}$ is a 3×3 rotation matrix and $\boldsymbol{t}$ is a translation vector in $\mathbb{R}^3$. In this thesis in addition to the representation above a tangent space representation $se(3)$ in vector form is also used

$$P_i = \begin{pmatrix} x & y & z & \omega_1 & \omega_2 & \omega_3 \end{pmatrix}^T \tag{3.2}$$

w_i are components of a 3×3 skew-symmetric matrix $[\omega]_x \in so(3)$

$$[\omega]_x = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \tag{3.3}$$

when ever required, tangent space form of Eq. (3.2) can be shown in $SE(3)$ via an exponential map $exp : se(3) \rightarrow SE(3)$ which has a closed form expression. This map is defined using Rodrigues formula [38] :

$$A \equiv \frac{\sin(\theta)}{\theta}$$
$$B \equiv \frac{1-\cos(\theta)}{\theta^2}$$
$$\theta \equiv \sqrt{\omega^T \omega}$$
$$R \equiv I + A[\omega]_x + B[\omega]_x[\omega]_x$$
$$V \equiv I + B[\omega]_x + (\frac{1-A}{\theta^2})[\omega]_x[\omega]_x,$$
$$exp(P_i) = \left(\begin{array}{c|c} \boldsymbol{R} & V_u \\ \hline 0 & 1 \end{array}\right), \tag{3.4}$$

There is also a similar closed form expression of logarithm map $log : SE(3) \rightarrow se(3)$ which takes a group element to a tangent vector. Details of Lie group algebra can be found in [121, 38].

Error between two nearby poses are calculated frequently in this thesis, this error can be calculated with first mapping both poses to the tangent space with log map, calculating difference between two poses and mapping the result back to $SE(3)$ using exponential map.

3.2 Bézier Curve Parametrization

In order to represent a curve there exist many method of which the simplest can be shown by points sampled from it, set of functions can also be used in order to define the curve such as polynomials. We are interested in use of a class of Parameterized curves known as Bézier designed by Dr. Pierre Bézier. Bézier curves of various degrees can be shown as

$$\boldsymbol{P}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{(n-i)} t^i \boldsymbol{P_i}, \tag{3.5}$$

given control points $\boldsymbol{P_i}$, points of the curve can be found by varying time parameter $t \in [0 \dots 1]$ in formulation above. An arbitrary parameter interval for t can also be formulated as well [112]. This parameterization is used in future chapters to reduce dimension of control signal and largely improve performance of algorithms. If required these curves can be glued together or split as required. Further operations on Bézier curves can be found in [112, 53].

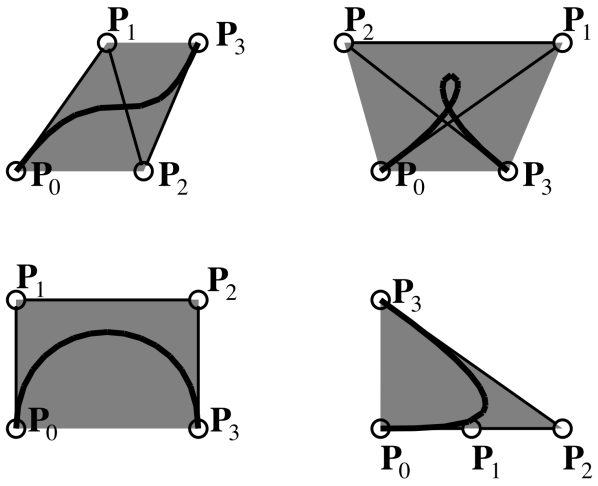

Figure 3.1: Convex hull of a bézier curve

3.2.1 Convex Hull Property

An important property of Bézier curves that is used in future chapters is that the resulted curve from a set of Bézier curve, always lies within the convex hull of its control points. The convex hull can be extracted by connecting control points via lines and the area contained in resulted polygon would represent convex hull. Since all control points of curve lie on one side of an edge of convex hull, it is impossible for the center of mass (Section 2.1 of [112]) of those control points to lie on the other side of line. A graphical representation of convex hull is given in Figure. 3.1

3.3 Gauss-Newton Nonlinear Optimization

In this thesis Gausss-Newton optimization method or some other variations of it are used in order to solve non-linear least squares problems. This method can take advantage of second order derivatives of cost function to achieve higher convergence rates. An m dimensional cost function with n variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ where $m \geq n$ and multiple residual functions $r(x)$ is given as

$$C(x) = \sum_{i=1}^{m} r_i(x)^2, \tag{3.6}$$

residual function is normally defined in an error function form

$$r_i(x) = z_i - h(x_i), \tag{3.7}$$

where z_i is the expected out come of the function $h(\cdot)$. Cost value of $C(x)$ can be minimized iteratively and a minimum value can be found using following update formulation from iteration s to $s+1$

$$x^{(s+1)} = x^{(s)} - (\boldsymbol{J_r}^T \Sigma^{-1}\ \boldsymbol{J_r})^{-1} \boldsymbol{J_r}^T \Sigma^{-1}\ r(x^{(s)}), \tag{3.8}$$

$$\boldsymbol{J_r} = \frac{\partial r_i(x^{(s)})}{\partial x}, \tag{3.9}$$

where $\boldsymbol{J_r}$ is the jacobian matrix of $r(x)$ with respect to vector of parameters x and Σ is diagonal matrix of uncertainties of measurements z_i. If calculation of an analytical jacobian is not possible then a numerical version can be calculated by finite differencing approach. In Eq. 3.8 the term $\boldsymbol{J_r}^T \boldsymbol{J_r}$ is known as hessian matrix.

3.4 Bond Graph Modeling

One of my interests in developing models is to design the model of systems from geometric descriptions and parameters of system. For this purpose two approaches are studied in this thesis of which one is by use of Bond Graph modeling. Using Bond Graphs it is possible to describe the structure of the system in a graphical model and then through an automated algorithm all differential equations of system can be extracted. Here we give a simple example of this approach where more details can be found in [55].

Bond graph models are designed based on energy transfer between components. For example consider the spring, damper, mass system in Figure. 3.2. Every component of this model can be represented with bond graph component. Mass can be shown by an inertial component I, spring can be shown by capacitive component C and damper can be shown by a resistive component R. Bond graph components are connected to each other with bonds where each bond represents the path which that component exchanges energy with rest of the system. Arrows on the bonds show

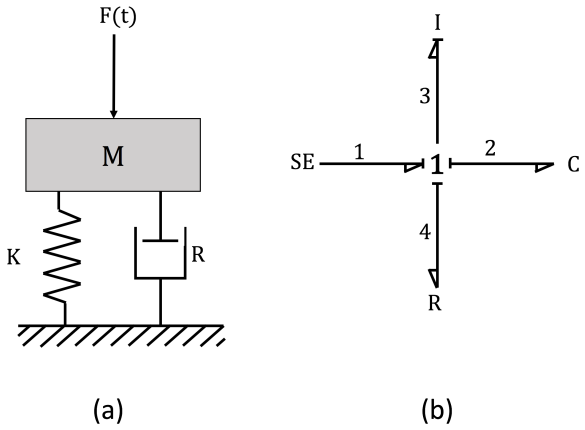

Figure 3.2: Bond Graph modeling of dynamic systems. (a) spring-damper-mass system with force input. (b) corresponding Bond Graph

direction of power distribution where as perpendicular lines on each bound define direction of flow in the system. A **1** component in Figure. 3.2 is a flow (velocity in mechanical systems) equalizing component which means that all the components have same velocity (spring and damper have same velocity as mass in their connection point to the mass). A source of energy is shown with SE component which corresponds to force input. Bond Graph in Figure. 3.2 can be used in a very well studied algorithm to write analytical equations of motion. Resulted formulation might require simplification by substitution to be in a more understandable equation for humans. For the system in Figure. 3.2 equations of motion can be extracted as

$$\begin{aligned} \dot{q}_2 &= \frac{p_3}{M_3} \\ \dot{p}_3 &= SE1 - K_2 * q_2 - R_4 \cdot \frac{p_3}{M_3}, \end{aligned} \tag{3.10}$$

where p_3 is momenta of mass and q_2 is displacement of mass with respect to ground, $SE1$ is input source, K_2 is spring coefficient, R_4 is damper coefficient and M is weight of mass.

by rearranging the equation above the traditionally equation of mass-damper-spring system can be achieved

$$m \cdot \frac{\partial^2 q_2}{\partial t^2} + R \cdot \frac{\partial q_2}{\partial t} + K \cdot q_2 = F(t), \tag{3.11}$$

Chapter 4

Experimental Platform and Physics Simulator

Using a dynamics model in a simulator framework is very helpful in testing different control and perception algorithms. However, in using a simulator one needs to be careful in set of assumptions which are made about the physical system as one might make wrong conclusions. In order to verify the effectiveness of a method, tests on a physical platform needs to be performed. When it comes to self driving vehicles, doing experiments on actual vehicles require obeying many of Department of Motor Vehicle rules and considering many safety conditions such that experiments don't cause any harm. Modifications to a normal to make it able to be driven with a computer is also not a task easily achievable. For this purpose a smaller scale of a four wheel drive car has been developed to be used in physical experiments. Smaller vehicle gives us the opportunity to do the experiments in a much smaller area while not requiring any regulations. Having a robotic platform in lab environment is both convenient to use and reduces cost of experiments.

Robotic vehicle (Parkour Car) in Figure. 4.1 shows the final assembled robot. This robot has the following specifications.

- Scaled at $\frac{1}{8}^{th}$ of actual vehicle size
- A Core-i7 computer with multiple USB3 connections ports
- Wifi Connection
- Wireless Gamepad connection
- Four wheel drive drivetrain

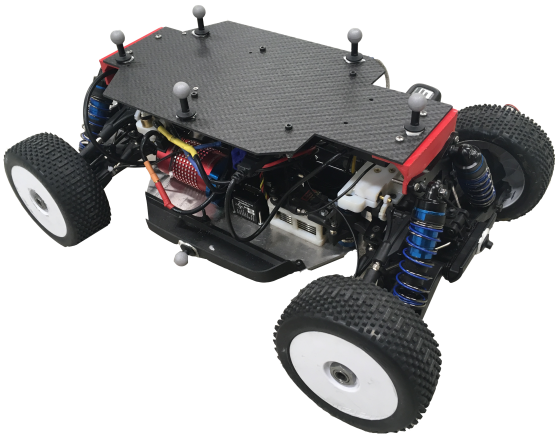

Figure 4.1: Parkour Car Robot designed to be used in different autonomous driving experiments

- High torque electric brushless Motor
- Servo driven front and back steering
- Single ECU controlling actuators, sensors and power distribution
- Swing Arm sensors to measure suspension deflection
- Steering angle position sensors
- Four wheel velocity/position optical encoders
- Engine shaft Hall effect encoder for engine speed feedback
- Nine degree of freedom inertial measurement unit
- Two front facing high speed global shutter cameras
- Carbon fiber enforced chassis

4.1 Design Details

4.1.1 Electronics Design

Parkour car includes custom made electronics as well as of-the-shelf compute boards, A Hierarchical view of overall electrical connections is given at Figure. 4.2.

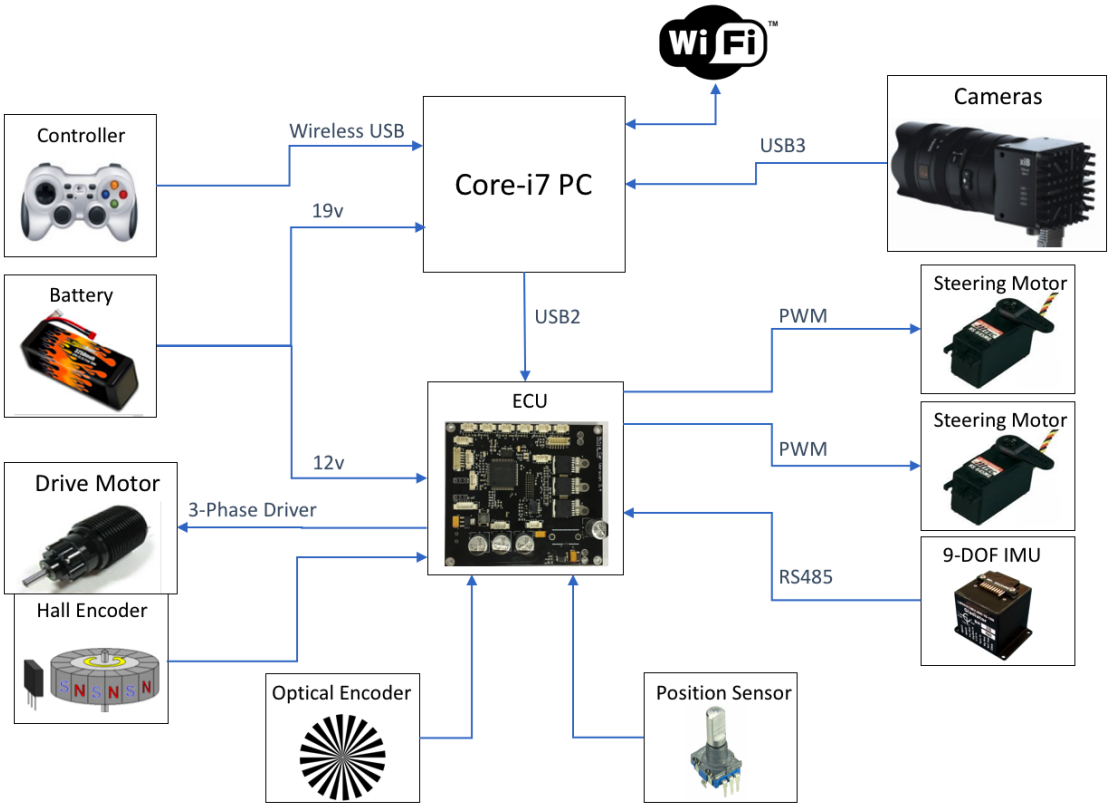

Figure 4.2: Hierarchical view of parkour car's wiring connection

Parkour car's ECU includes an ARM-Cortex-M4 processor and handles all the signal conditioning for the main PC. Signals are captured through different interfaces like RS-232, RS485, digital and analog stored in memory. Once a new data is captured, a data packet is constructed and transmitted through a USB2.0 connection to the man PC. Transmitted packet includes the information from all the sensors attached to the ECU. In same fashion, ECU receives a data packet from main PC which has the motor torque and steering servo angles and corresponding signals are applied to the motors. Communication between ECU and the main PC happens in baud rate of 115.2Kbps and data packets include header and checksum bytes in both directions to avoid transmission of corrupted data.

Two methods has been tested in ECU's firmware design. Effectiveness of FreeRTOS realtime OS with 1 msec updates of OS (maximum achievable update rate) has been compared to bare-metal implementation while considering execution times of different processes. Each process has been

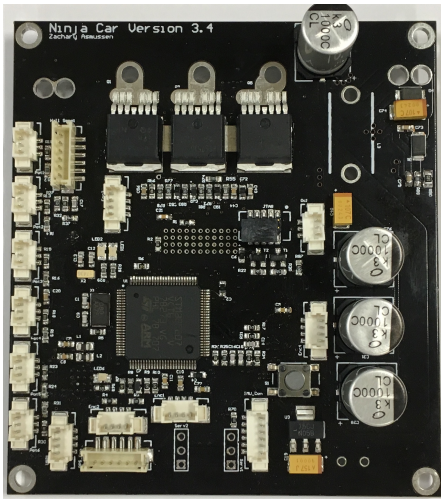

Figure 4.3: Top view of ECU

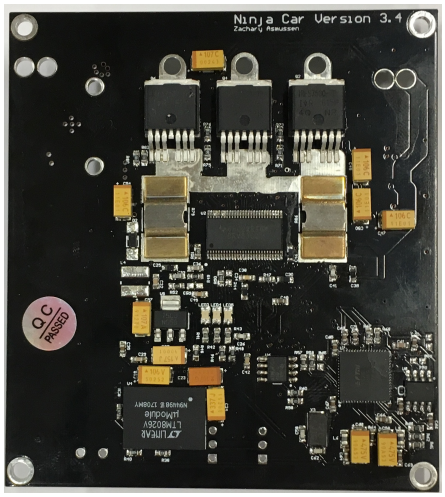

Figure 4.4: Bottom view of ECU

timed for execution while data capturing from different channels as well as main PC communication happens through DMA(Direct Memory Access) channels available in STM32F407 processor. Final results showed that bare-metal implementation has higher performance due to low processing time for sum of all process times (<1ms) and FreeRTOS's overhead for OS actually slows the system down.

For safety of system, a watchdog timer monitors the communication channel. Watchdog resets every time that a new command is received from main PC and as a result actuators can operate. If new data packet does not arrive on a timely fashion then the steering wheels are reset to zero angle and the main engine is halted.

Drive terrain has been one of the difficult part of the design but after five iterations of this ECU final result has been promising. BLDC motor drive circuit is capable of monitoring thermal condition of the drive circuit and shutdown the power in the case of high temperature. It has been observed that if vehicle is driven continuously for long time with high torque then the ECU would turn the system off. Time of operation of vehicle can be increased by adding a cooling system to the ECU in the future.

ECU has been optimized in size to fit in small robotic platforms so it measures dimentions of $9 \times 8 \times 2$ cm, has a four layer PCB layout and includes components in both sides of the board.

Both top and bottom views of the ECU are shown in Figures 4.4 and 4.3.

ECU design is open source and further details of the system as well as schematics can be found in **??** and production files are available in ARPG Lab's Github page [115].

Two different computers has been tested with this vehicle. First, a Gigabyte Brix PC (Figure 4.6) was used which had a Intel Core-i7 processor with four separate cores and total of eight hyper threads. Later, a intel NUC board was installed which has the similar performance to Gigabyte model but has lower wight.

Parkour car is equipped to a three cell 5000 mAh LiPo battery which powers the main PC, ECU, Sensors and actuators. Our tests has shown that vehicle is capable of operating for 45 minute when CPU is used at 100% and vehicle is driven with 30% average torque.

4.1.2 Mechanical Design

Most of Parkour car's mechanical parts (Steering mechanism, Differentials, Suspension system) are taken from Team Associated's RC8.2e RC car but dual steering system and carbon fiber chassis has been designed to reduce the system weight and make the vehicle capable of taking more interesting maneuvers. Figure 4.5 and 4.6 shows location of different parts inside vehicle chassis. A Brushless DC motor create the torque required to propel the vehicle. Motor torque splits in half between front wheels and back wheels with a central differential and lated a front and back differential splits the input torque between left and right wheels. A removable junction between differentials and also between wheels and differentials makes it possible to achieve different configurations for the drive terrain and test different vehicle behaviors. For Example, connection between front wheels and front differential can be removed and front axis of central differential can be fixed to achieve a back wheel drive vehicle drive terrain. Similarly, a front wheel drive mechanism can be handled.

Steering mechanism can also be defined by fixing one of the steering servos to have front/back wheel steering or have dual steering by commanding both steering servo motors. Dual steering mechanism has shown a big advantage in reducing turn radius and increasing steering response of

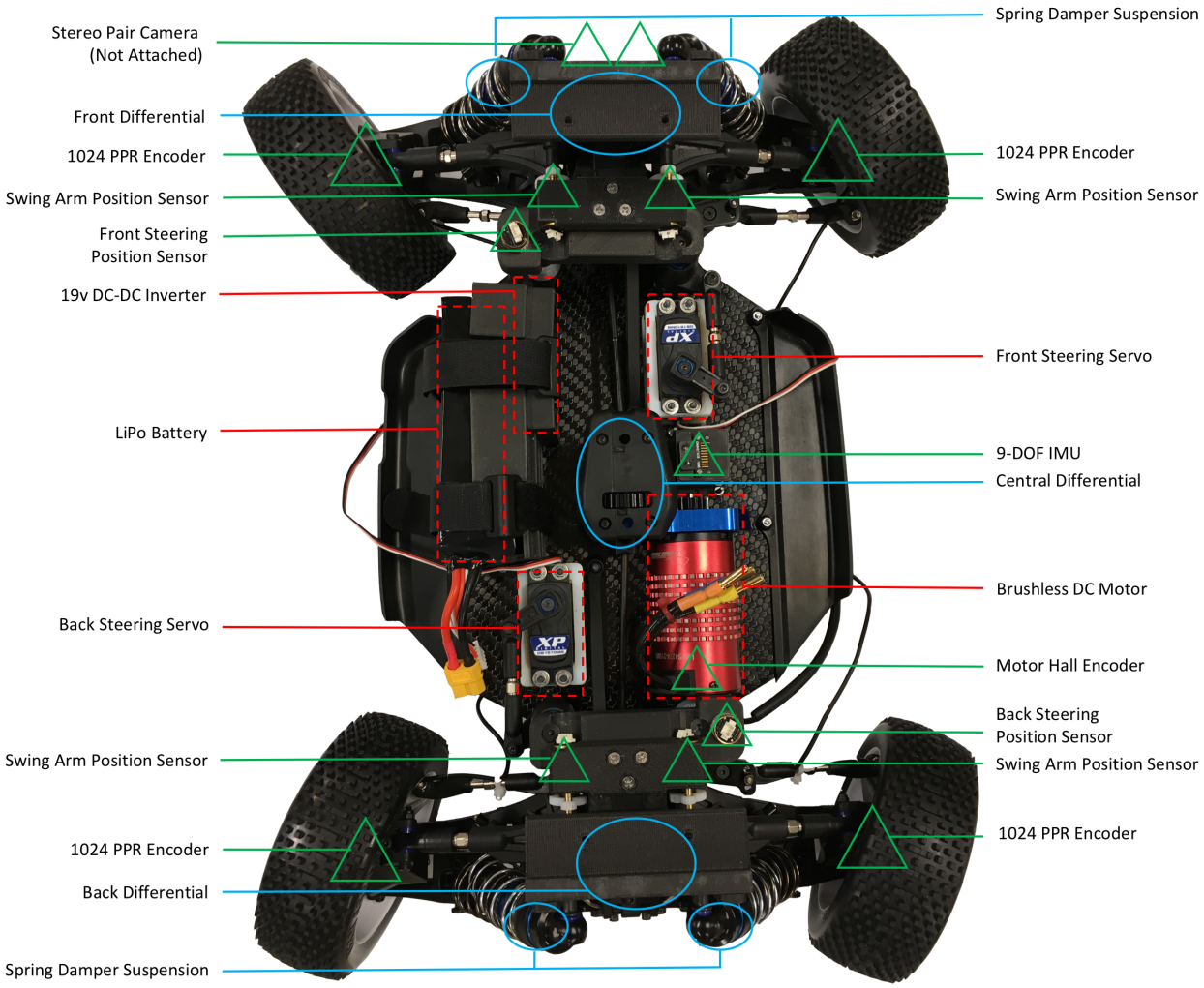

Figure 4.5: Parkour Car's parts. Triangles show vehicle sensors, ovals represent major mechanical parts and rectangles correspond to actuators and the battery

the vehicle. Steering two wheels in a vehicle can be non intuitive for human drivers if not possible but since a self driving vehicle is driven with a computer algorithm achieving this task can be trivial.

4.1.3 Software Drivers

A software pipeline has been developed which abstracts the complications of operating the vehicle and reading its sensor information. This has been developed as a part of ARPG's open-source HAL Library [**?**]. User can access the vehicles data by defining a callback function and an

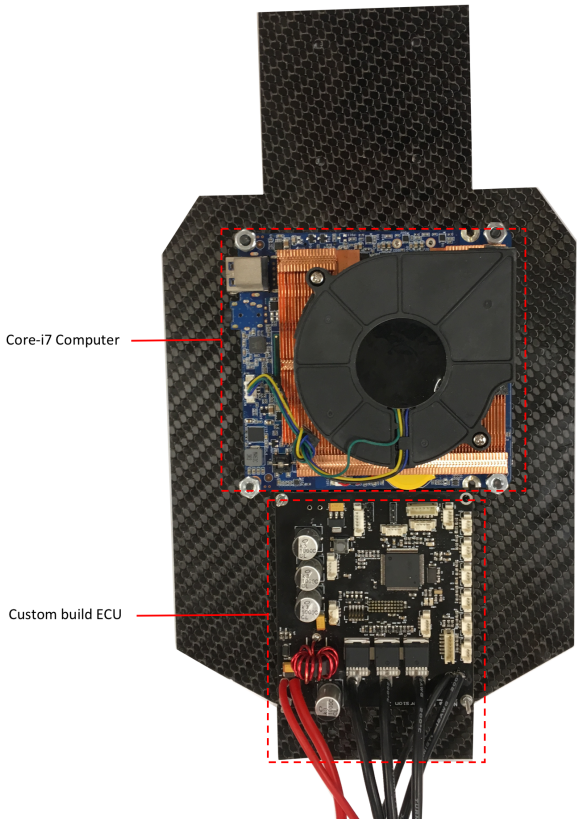

Figure 4.6: Parkour Car's processing Units attached to top plate in chassis

interrupt will call this function as soon as a new data packet is received by main PC. Similarly, user can update steering and accelerating variables in the driver and software pipeline will handle transmission of this update to the ECU.

4.2 Physics Engine Dynamic Model (Spirit)

Different approaches in modeling of a car has been discussed in Chapter 2. In the conclusion it was shown that dynamics physics engines are among strong choices of models which can model behavior of the robotic platform as well as its interactions with the environment. For the purpose of this thesis a vehicle simulator called **Spirit** has been developed in c++ language [3] which is based on bullet physics engine library [24]. Using this library, a user can define vehicles of different size and geometry with multiple wheels and drive terrain and simulate trajectories of the designed model.

This pipeline also provides derivatives of vehicle trajectories with respects to desired parameters of vehicle which are used in methods explained later in Chapter 5 6 . One of the major difficulties in design of control systems is that the designer needs to have a very strong mathematical background and modeling experience in order to be able to design a proper model, Using this method modeling task is simplified to defining geometric constraints of the physical system and a model is developed automatically. This idea of moving from geometric constraints to a mathemtical model has also been kept in mind while designing model of vehicle in Chapter 7.

Spirit library also is capable of using multiple processor cores while in operation which speeds up the processing of higher level algorithms. A graphical user interface has also been implemented in this library to give proper feedback to the user which can be very helpful in debugging problems when working with a higher level algorithm.

Chapter 5

Online System Identification and Calibration of Dynamic Models

This chapter is concerned with system identification and the calibration of parameters of dynamic models used in different robotic platforms. Independent of what method is used for modeling a robotic platform, there are always parameters in the model which require calibration in order to get similar behavior between the model and physical platform. For this purpose a constant time algorithm has been developed in order to automatically calibrate the parameters of a high-fidelity dynamical model for a robotic platform. The presented method is capable of choosing informative motion segments in order to calibrate model parameters in constant time while also calculating a confidence level on each estimated parameter. Experiments in both simulated environment as well as on the parkour car platform of Chapter 4 has been done to verify effectiveness and accuracy of proposed method.

5.1 Introduction

Dynamical models are used in planning, control and state estimation of robotic platforms; however, finding a model which describes a real physical platform well is nontrivial. Despite the amount of effort one might put into designing a high-fidelity dynamical model, there exists yet another challenge that arises: such models typically have a large number of parameters which must be tuned very accurately. For instance, when operating a ground vehicle around a turn at high speed, a small inaccuracy in the coefficient of friction could send the vehicle into slip through a discontinuous change in dynamics. Such behavior would be deleterious and potentially hazardous

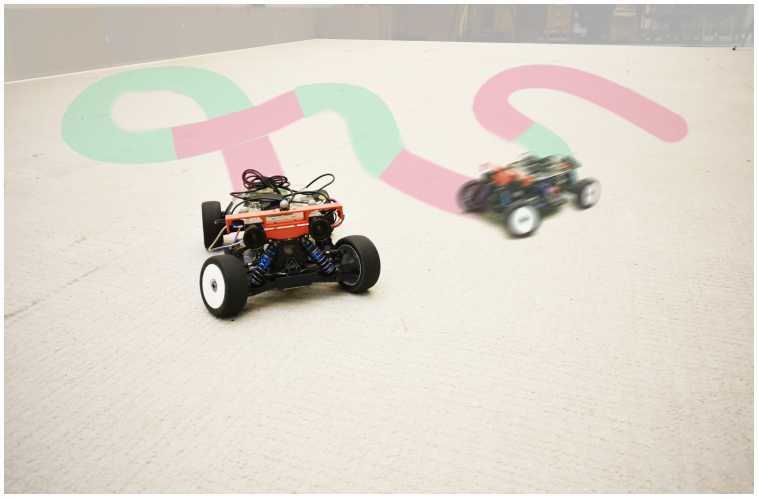

Figure 5.1: Informativeness of trajectory of parkour car where green color correspond to informative segments and red segments are non informative segments

unless it were predictable, in which case it might be used to an advantage in e.g. guiding the vehicle through a tight corner. Even if an accurate set of parameters are chosen via expert tuning, some of these parameters would inevitably change in time due to mechanical degradation or environmental disturbances.

Even more complicating to the scenario is that only some of these parameters are imperceptible in driving unless an obscure set of conditions are met. For instance, in a ground vehicle the static coefficient of friction between tires and the ground is only measurable at the point of slip; the dynamic coefficient is only measurable **when slipping,** which is potentially even more challenging to observe. Also, it is not immediately obvious which types of motions a vehicle platform might undergo that allow a human operator to determine the suspension stiffness. No one motion will allow all of these parameters to be observed, thereby creating a quandary of how to operate a vehicle such that all relevant physical parameters might be known to some degree of certainty.

And yet despite these difficulties the goal of grounding a system's representation in concrete parameters that represent physical relationships remains highly desirable. Were the estimation of these physical parameters to be robust and reflexive for different operating scenarios, it is possible that model-predictive control could be applied in contexts that are currently challenging, such as

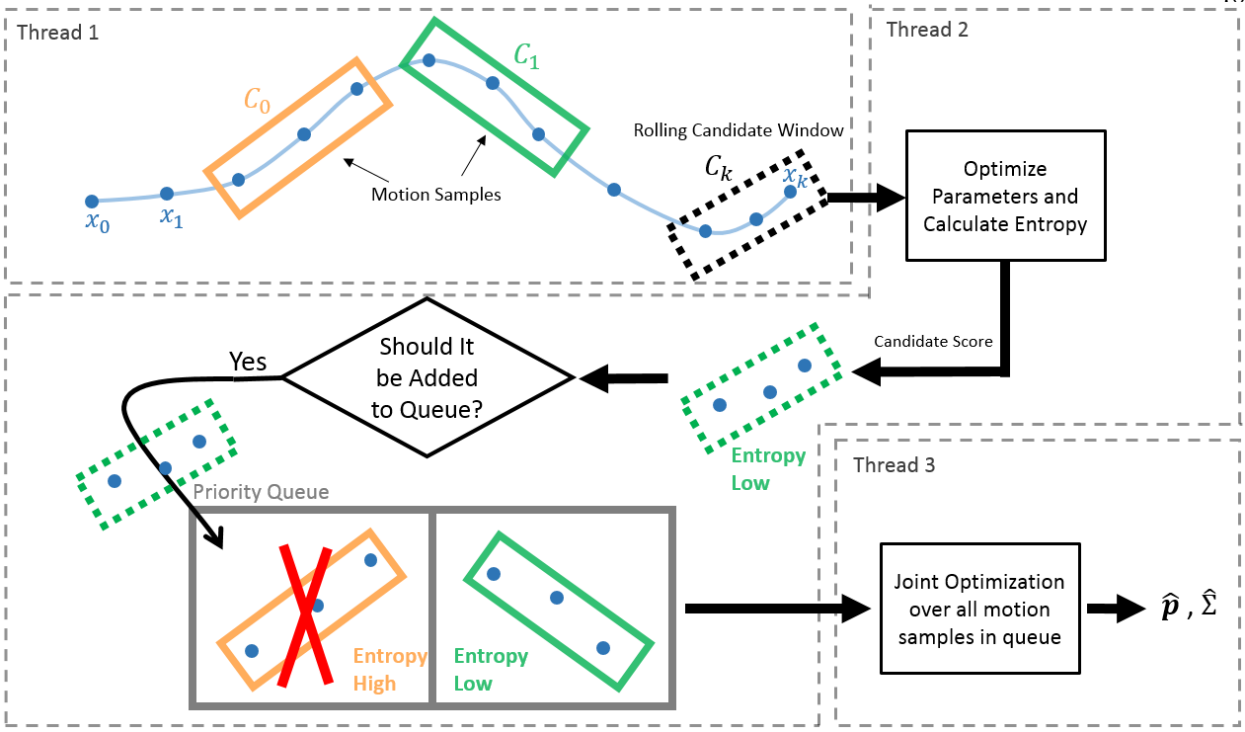

Figure 5.2: A flow chart overview of the method. First, a sequence of state information known as a "motion sample" is captured within a candidate window, which is used to estimate calibration parameters. Next, the motion sample is compared to a priority queue of motion samples. The comparison operator in this priority queue determines if the sample in the candidate window reduces the entropy over the whole calibration parameter set while maintaining a balance of informative motion segments across all calibration parameters. If the motion sample is comparatively better than any within the priority queue, the candidate window is augmented to the priority queue and the lowest-scoring window in the queue is purged. This pipeline consists of three threads which run in parallel with synchronization tokens exchanged between threads.

when dynamic parameters are changing in time or are unknown before system operation. Furthermore, these parameters might be estimated through measurements through external sensors, e.g. cameras or infrared, providing another mechanism to directly influence controllers built on these models. Finally, reasoning about such systems can be transparent and tractable from a nearly intuitive standpoint.

To address this, the present work develops a probabilistic calibration method which may be used to estimate parameters of a dynamical model of a ground vehicle.

5.2 Related Work

The parameter estimation problem has been well-studied on robotic platforms, traditionally aimed at particular parameters on certain experimental platforms; these efforts have largely grown out of a need to estimate parameters that cannot be directly measured, such as the coefficient of friction for wheels or the extrinsic transformation between an IMU sensor and image sensor of a camera. Another possibility is that such parameters are not truly static and change during operation. For example in the case of ground vehicles, GPS measurements have been used in order to estimate vehicle mass [7], roll stiffness and damping ratio [108] or tire/road friction coefficient [125]. In the case of quadrotor UAVs, IMU measurements have been used to estimate the body moment of inertia [1]. For other platforms, the 6-DOF pose measurements have been leveraged to estimate inertial parameters of a grasped object [77] or thrust factor of actuators [67]. Geometric/kinematic parameters have also been estimated for different platforms, like linkage length in a robotic arm [46, 103], as well as distances and angles between a car chassis coordinate frame and its tire frames [122]. More recently in the robotics community, probabilistic approaches have been developed in order to estimate camera/IMU/laser scanner extrinsic parameters [58, 86, 111].

The calibration task, frequently known as "system identification" for physical platforms, generally involves estimating some parameters of the model from some indirect sensory measurements, however the choice of sensors and their attachment location on an autonomous platform is very important for achieving accurate calibration estimates, since only some specific motions of robot would render parameters observable. Regressing parameters on data which does not render parameters of interest observable generally leads to wildly inaccurate results or a lack of convergence. One approach previously taken is to attach a sensor in a specific location which properly excites its output signal; however, finding the ideal location might be very hard if not impossible [61, 91]. Two ways to avoid calibrating a platform using a dataset which does not make parameters observable is to restrict motions of robot to some pre-planned trajectories which have good excitation properties [122, 45] or to algorithmically choose the most informative motion segments [111, 58, 86] from a

platforms trajectory. In these approaches, one must be careful in choosing informative segments of motion since an uneven amount of information could bias the results toward selecting segments only for parameters with more informative segments without balancing this with accuracy over the entire parameter set. To avoid this problem, techniques like normalization of the posterior estimate has been proposed [56], but this has been exclusively applied to the SLAM problem.

5.3 Methodology

In this work we develop a self-calibration approach to the estimation of parameters for a dynamic physical model of a ground platform using as input the state estimates to the vehicle. The approach develops an algorithm for choosing informative motion segments for all calibration parameters, and is extensible to intrinsic and extrinsic parameters (e.g. tire friction coefficients and wheel base) for the vehicle.

In general, dynamic models of robotic platforms are developed under the assumption that the platform operate only in the same regimes as those in which they are calibrated. This work represents a step toward loosening this highly constraining assumption, allowing for such parameters to potentially change in a dynamic environment and for the system to estimate these new parameters in constant time. Furthermore, the procedure developed allows for the estimation of a high-fidelity (and high-dimensional) model which, from a principled standpoint, has proven to be quite elusive. Many approaches for example assume a four-wheeled vehicle is reducible to a bicycle model [63], however the assumptions implicit in such a model are very constraining: a vehicle must drive on a flat surface, with tires always in contact with the ground and without any roll. Many vehicles in normal operation, and especially in challenging off-road conditions, do not exhibit such behavior.

We now will provide a high-level explanation of the algorithm, accompanied by a flow chart in Figure 5.2. At the core of the method, and the key innovation in this work, is the evaluation of discrete-segment motion samples by assigning a "score" that quantifies the sample's informativeness toward estimating the parameters. The collection of these motion samples in a priority queue are used to jointly estimate calibration parameters online. Since both the candidate window and

priority queue have fixed sizes, the calibration parameter estimates are produced in constant time. Our approach is also highly parallelizable, with the consumption, candidate window and priority queue optimization operations each running in separate threads. Note that the parallelization of this algorithm is shown using dotted windows for each part of algorithm in Figure 5.2.

5.3.1 Dynamics Model

First we suppose that we have a model of the dynamics of the platform given as:

$$\boldsymbol{x}'(t) = \boldsymbol{\Phi}(\boldsymbol{x}(t), F(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{5.1}$$

in which $\boldsymbol{\Phi}(\cdot)$ represents the dynamical model, $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state of the system, $\boldsymbol{x}'(t)$ is the derivative of the state under the governing dynamics, $F(t)$ consolidate external forces applied to the system, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is a control input (for vehicles, generally the steering and acceleration), and $\boldsymbol{p} \in \mathbb{R}^w$ is model parameter vector to be calibrated. Distinctions between the system's "real dynamics" and modeled ones are ignored with the restriction that modeling assumptions are not violated when operating the platform. For example, if a physical model is utilized which restricts the vehicle to not be airborne, it is assumed the vehicle will never go airborne. This model is discretized in time, such that $\boldsymbol{x}_k$, $k \in \mathbb{N}$ is the n-dimensional state of the platform at time step k.

Strictly speaking, the generality of the method we described may be extended to non-ground vehicle platforms; however, we will describe this method as we have developed it for such systems. The state $\boldsymbol{x}_i$ includes the $\mathbb{SE}(3)$ pose of the wheels and chassis as well as wheel speeds and rigid body angular momenta. Many of these quantities may be directly estimated through, e.g. modern SLAM algorithms or properly placed sensors like linear or angular encoders. For a simple car model, a calibration parameter vector might include vehicle properties like tire and chassis geometry; more complicated models might consider inertial properties, linear and rolling friction coefficients of all bodies, suspension anchor points, suspensio damping and stiffness coefficients, motor speed-torque parameters for steering and acceleration motors and a chassis's center of mass location.

The control input $\boldsymbol{u}(t)$ is discretized by sampling the continuous input signal over time

intervals $\Delta t_k = t_k - t_{k-1}$ which need not be uniform. Through this, it is assumed that the control input is constant within the interval $[t_k, t_k + \Delta t_k]$. Current control inputs, system state and the timestamp of the system are sampled and stored as a triplet $\boldsymbol{s}_k = [\boldsymbol{x}_k, \boldsymbol{u}_k, t_k]$ which compose the motion samples at timestep k.

5.3.2 Rolling Candidate Window

In order to score motion segments based on how much information they contribute to the calibration parameter estimation problem, a rolling window of size n_c is used. The rolling candidate window at time step k is $\boldsymbol{C}_k = [\boldsymbol{s}_{k-n_c}, \ldots, \boldsymbol{s}_k]$ which is composed of the n_c latest motion samples. Within $\boldsymbol{C}_k$ a score of informativeness is calculated. The score is calculated by estimating parameters $\boldsymbol{p}$ and calculating a score based on the entropy of the posterior. Since the entropy is a way to quantify uncertainty, lower scores represent more desirable motion samples contained within the window.

For a given candidate window $\boldsymbol{C}_k$, a cost function is constructed as follows:

(1) initialize Eq. (5.1) at a state/input pair given at the beginning of the window as the canonical estimate of that state; i.e.,

$$\hat{\boldsymbol{s}}_{k-n_c} = \boldsymbol{s}_{k-n_c}; \tag{5.2}$$

(2) integrate Eq. (5.1) forward with the measured control inputs over the time intervals stored in $\boldsymbol{C}_k$ to get $\hat{\boldsymbol{C}}_k$:

$$\hat{\boldsymbol{C}}_k = [\hat{\boldsymbol{s}}_{k-n_c}, \ldots, \hat{\boldsymbol{s}}_k]\,; \tag{5.3}$$

(3) and finally, construct a cost function:

$$\boldsymbol{\Psi}_{\boldsymbol{C}_k} = \boldsymbol{C}_k\,[\boldsymbol{x}] \boxminus \hat{\boldsymbol{C}}_k\,[\hat{\boldsymbol{x}}]\,, \tag{5.4}$$

where $\boldsymbol{C}_k\,[\boldsymbol{x}]$ represents only the vector of states (from the triplet $\boldsymbol{s}_k$) in the corresponding window, and operator $\boxminus$ calculates a weighted error between two state vectors.

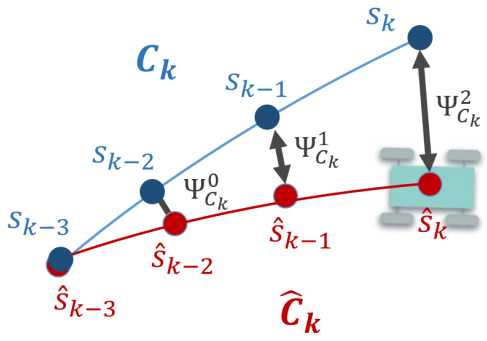

Figure 5.3: Elements of cost function before candidate window optimization. For a given motion segment $\boldsymbol{C}_k$ (blue dots) in candidate window for a window of size $n_c = 4$ and an integrated path $\hat{\boldsymbol{C}}_k$ with initial parameter values (red dots), a corresponding cost function $\boldsymbol{\Psi}_{\boldsymbol{C}_k}$ is constructed.

The cost function $\boldsymbol{\Psi}_{\boldsymbol{C}_k}$ is later minimized in order to calculate the covariance of the posterior $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{C}_k}$ and a score for current window. The candidate window $\boldsymbol{C}_k$ is a rolling window, meaning that when a new measurement arrives, $\boldsymbol{s}_{k-n_c}$ is removed and the new measurement is pushed back to the window at $\boldsymbol{s}_k$; a new score is then calculated.

5.3.3 Optimization

Calibration parameters are estimated in a nonlinear maximum likelihood estimation framework. The joint probability distribution of parameter $\boldsymbol{p}$ given state measurements $\boldsymbol{s}_k$ in window $\boldsymbol{C}_k$ and dynamics provided by Eq. (5.1) is:

$$P(\boldsymbol{p}, \boldsymbol{Q}_k[\boldsymbol{x}]) = P(\boldsymbol{Q}_k[\boldsymbol{x}]|\boldsymbol{p})P(\boldsymbol{p}) = \prod_{i=1}^{n_q} P(\boldsymbol{C}_i[\boldsymbol{x}]|\boldsymbol{p}), \tag{5.5}$$

where $\boldsymbol{Q}_k = [\boldsymbol{C}_0, \ldots, \boldsymbol{C}_{n_Q-1}]$, represents all candidate windows in the queue at time step k for a queue of size n_Q. In Eq. (5.5), the likelihood term $P(\boldsymbol{Q}_k[\boldsymbol{x}]|\boldsymbol{p})$ is factored since different candidate windows are selected such that they are independent of each other (see Section 5.3.4). Note that the prior term $P(\boldsymbol{p})$ can be dropped since the priority queue will carry the prior of the estimate.

Noting that the prior term in Eq. (5.5) vanishes since the first state variable is always fixed

to the first measurement (see Eq. (5.2)), hence we have:

$$P(\boldsymbol{p}, \boldsymbol{C}_k[\boldsymbol{x}]) = P(\boldsymbol{C}_k[\boldsymbol{x}]|\boldsymbol{p}), \tag{5.6}$$

An optimal estimate $\hat{\boldsymbol{p}}$ for the parameter vector $\boldsymbol{p}$ can be calculated by minimizing the joint probability:

$$\hat{\boldsymbol{p}} = \underset{p}{\operatorname{argmax}}\, P(\boldsymbol{p}, \boldsymbol{C}_k[\boldsymbol{x}]), \tag{5.7}$$

According to Eq. (5.5), the solution to this estimation problem can also be achieved by maximizing the likelihood term. Assuming that parameter noise is Gaussian distributed, we can write probability density function of each likelihood term in Eq. (5.5) as:

$$\begin{aligned} P(\boldsymbol{C}_k[\boldsymbol{x}]|\boldsymbol{p}) &\propto \exp(-\frac{1}{2}\|\boldsymbol{C}_k[\boldsymbol{x}] \boxminus \Phi(\boldsymbol{C}_k, F(t), \boldsymbol{p})\|^2_\Sigma) \\ &\propto \exp(-\frac{1}{2}\|\boldsymbol{C}_k[\boldsymbol{x}] \boxminus \hat{\boldsymbol{C}}_k[\boldsymbol{x}]\|^2_\Sigma) \\ &\propto \exp(-\frac{1}{2}\|\boldsymbol{\Psi}_{\boldsymbol{C}_k}\|^2_\Sigma), \end{aligned} \tag{5.8}$$

in which $||\cdot||^2_\Sigma$ signifies squared Mahalanobis distance given measurement uncertainty $\boldsymbol{\Sigma}$ and $\boldsymbol{C}_k[\boldsymbol{x}]$ for the k^{th} candidate window. In the case of a rolling candidate window, the equation above will only have one likelihood term as mentioned at Eq. (5.6). then Eq. (5.7) is solved by iteratively updating parameter vector $\boldsymbol{p}$ with a trust region method [2]. Given the Gaussian distribution assumption, the covariance of the posterior is computed over calibration parameters by inverting the Fisher information matrix:

$$\hat{\boldsymbol{\Sigma}}(\hat{\boldsymbol{p}}) = Cov(\hat{\boldsymbol{p}}) = (\boldsymbol{J}^\intercal(\hat{\boldsymbol{p}})\, \boldsymbol{J}(\hat{\boldsymbol{p}}))^{-1}, \tag{5.9}$$

Here $\boldsymbol{J}(\hat{\boldsymbol{p}})$ is Jacobian of $\boldsymbol{\Psi}_{\boldsymbol{C}_i}$ at $\hat{\boldsymbol{p}}$. The Jacobian is checked to make sure it is full rank before calculating the covariance; if it is not of full rank, then the current motion sample is ignored due to lacking observability in all calibration parameters.

5.3.4 Entropy Score Board

Solving the MLE problem within a candidate window results in the covariance of the estimate $\hat{\boldsymbol{\Sigma}}$, which is used to calculate a score vector for the current motion sample. An entropy value for each parameter is given by $\boldsymbol{\kappa}_k[i] = \frac{1}{2}\ln(2\pi e\sigma_i)$ where $\boldsymbol{\kappa}_k$ is vector of entropy values of matrix $\hat{\boldsymbol{\Sigma}}$ and σ is diagonal vector of $\hat{\boldsymbol{\Sigma}}$ for the candidate window $\boldsymbol{C}_k$. Recall that lower entropy implies better candidate.

One of the primary considerations is that we would contain the number of motion samples we are tracking at any given time; this was part of our comparison operator design, and was found to aid robustness in collecting salient and informative segments for all parameters. To do this, we introduce the notion of a score board which is part of our comparison as follows:

(1) The maximal entropy for each parameter from all motion samples in the priority queue is stored in a vector $\boldsymbol{\lambda} \in \mathbb{R}^{n_p}$ (since each element of $\boldsymbol{\lambda}$ corresponds to an element of $\boldsymbol{p}$) where n_p is number of parameters to be estimated.

(2) A $n_p \times n_Q$ table $\mathbf{T}$ is constructed to keeps track of which candidate windows are informative for which parameters. $\mathbf{T} \in \mathbb{N}^{n_p \times n_Q}$ contains $\{0, 1\}$ values in its entries and an entry 1 in index $\mathbf{T}[i, j]$ shows that candidate segment in index j of priority queue is informative for parameter i.

(3) A vector $\boldsymbol{\tau} \in \mathbb{N}^{n_p}$ keeps track of the total number of informative segments for each parameter which currently exist in priority queue by summing over the rows of $\mathbf{T}$:

$$\boldsymbol{\tau}_i = \sum_{j=0}^{n_Q} \mathbf{T}[i, j] \qquad \mathbf{T}[i, j] \in \{0, 1\}. \tag{5.10}$$

For each newly calculated $\boldsymbol{\kappa}_k$, we compare corresponding indexes in this vector to current worst entropy $\boldsymbol{\lambda}$ according to Alg. 1.

Note in Algorithm 1, β is a tuning parameter representing the minimum acceptable score, and $\alpha = 0.95$ is a safety margin to ensure at least a 5% reduction on maximum entropy occurs when

Data: $\boldsymbol{C}_k, \boldsymbol{\kappa}_k, n_p, \boldsymbol{\lambda}, \beta, \alpha$
Output: $update\_condition, current\_score$
$current\_score \leftarrow 0$;
for $i \in \{0, \ldots, n_p - 1\}$ **do**
  if $\boldsymbol{\kappa}_k[i] < \alpha \times \boldsymbol{\lambda}[i]$ **then**
    $current\_score \leftarrow current\_score + 1$;
  end
end
if $current\_score \geq \beta$ **then**
  $update\_condition \leftarrow true$;
else
  $update\_condition \leftarrow false$;
end

Algorithm 1: Priority queue update condition

adding a motion sample to the priority queue. A candidate window might overlap with previous candidates in queue which would result in double counting the information in the queue and biasing the results; to avoid this, the current candidate window is checked for such a condition. If there is an overlap, then the current candidate is compared with the sample in the priority queue with an overlap as in Alg. 1 and replaces the entry in the queue if the resulting score is higher.

In the case that the update condition in Alg. 1 is satisfied and there are not any common segments between the current candidate window and any candidates in the priority queue, then a final condition is checked to make sure the amount of data entered is not biased toward a group of parameters. In this check, the new candidate is added to priority queue and corresponding entries of $\mathbf{T}$ which reduce entropy of parameter i are marked with *1*s and corresponding $\boldsymbol{\tau_i}$ and $\boldsymbol{\lambda_i}$ values are updated. If adding a new 1 to the table falsifies the queue limit inequality $\boldsymbol{\tau}_i \leq n_s$ on any parameters (n_s being maximum number of desired samples per parameter), then the table entry which previously had the highest entropy in vector $\boldsymbol{\lambda}$ is changed to a 0. Figure 5.4 shows a flow chart for this procedure.

To initialize this process, the very first candidate is accepted without any checks and later candidates are checked for having an overlapping window or if they have a lower entropy score but no replacements (with higher entropy candidates) takes place until the queue is full. Once a decision is made a pruning step takes effect by removing columns of table $\mathbf{T}$ which have all zeros

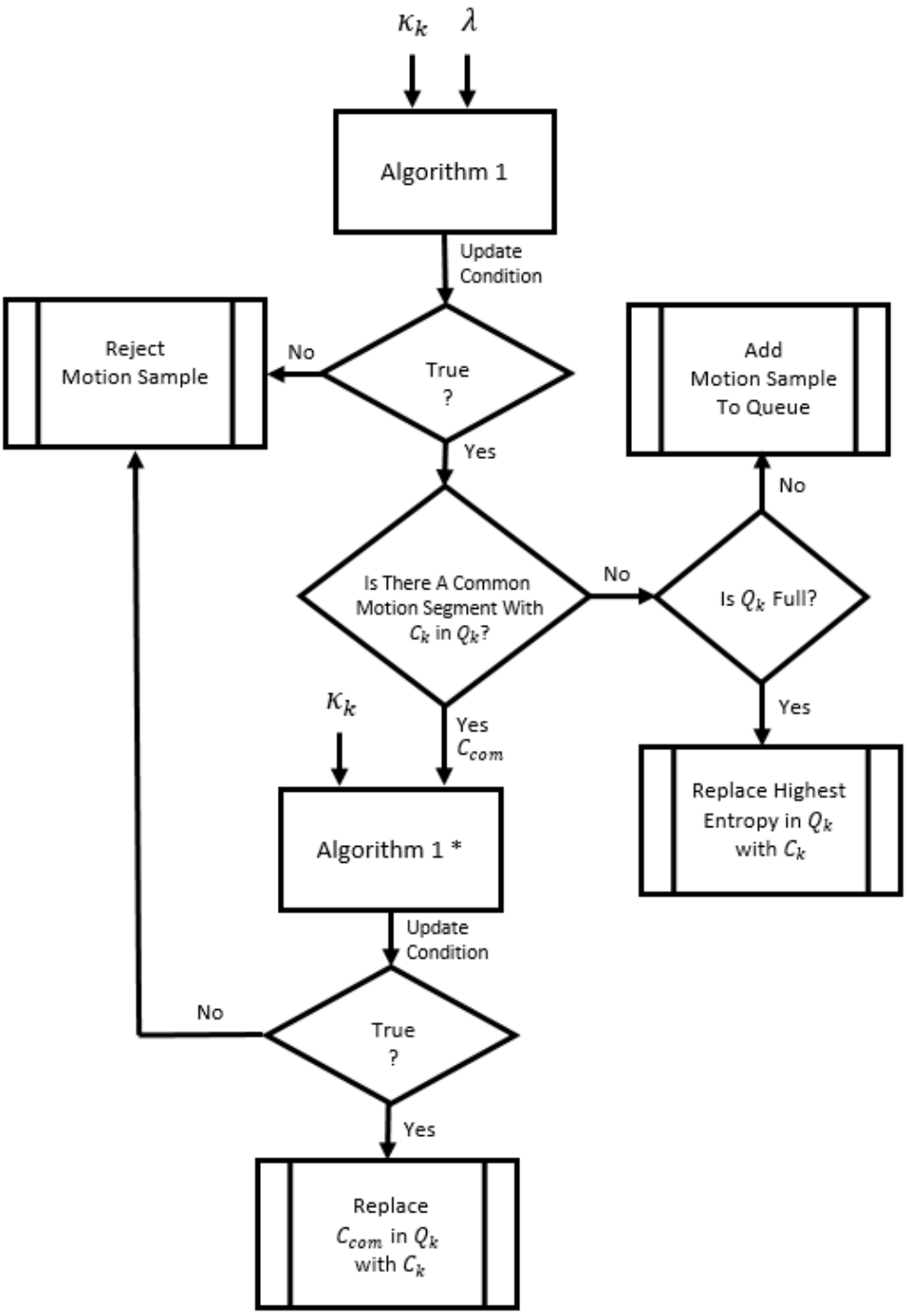

Figure 5.4: Flowchart for deciding if a motion sample should be added to the priority queue or not. In this chart $\boldsymbol{C}_{com}$ is the motion sample in queue which has overlapping segment with the candidate in current rolling window. In the figure, the boxed Algorithm 1 compares the current window's $\boldsymbol{\kappa}_k$ to the entropy vector of $\boldsymbol{C}_{com}$. This is opposed to $\boldsymbol{\lambda}$ in Alg. 1, where $\boldsymbol{\lambda} \leftarrow \boldsymbol{\kappa}_{\boldsymbol{C}_{com}}$.

and also removing corresponding candidate window from the queue.

5.3.5 Priority Queue

The priority queue $\boldsymbol{Q}_k$, is responsible for storing the set of candidate windows which have been selected using the criteria described in Section 5.3.4. At every update to this queue we construct a cost function and jointly optimize over all candidate windows in the queue. from Eq. (5.4) we can write

$$\boldsymbol{\Psi}_{Q_k} = \sum_{i=0}^{n_Q} (\boldsymbol{\Psi}_{C_i}), \tag{5.11}$$

The cost function $\boldsymbol{\Psi}_{Q_k}$ is then used in the optimization step (see Section 5.3.3) to find a new estimate $\hat{\boldsymbol{p}}$ as well as its corresponding covariance $\hat{\boldsymbol{\Sigma}}$. The estimated covariance is then used to asses the confidence level on each estimated parameter.

5.4 Experiments

In order to validate this algorithm, we have performed simulations on a model four wheel drive vehicle as well as experiments on a modified $\frac{1}{8}^{\text{th}}$-scale vehicle platform. The method as described has been implemented in the C++ language and tested in both simulation and experiment, and has reliably performed in constant time on an Intel Core-i7 CPU. As shown in Figure 5.2 with dashed lines, the pipeline consists of three main threads. The first thread is responsible for capturing motion samples in discrete time and constructing a rolling candidate windows as described in Section 5.3.2. When this task is complete, this thread signals to a second thread that the motion sample is ready for analysis. This second thread then minimizes the cost $\boldsymbol{\Psi}_{\boldsymbol{c}_k}$ using the optimization technique explained at Section 5.3.3. In addition to assigning a score to a given motion sample, this thread also handles the decision making explained in Section 5.3.4 and updating $\boldsymbol{Q}_k$. This thread then provides a synchronization token to a third thread which calculates and updates the cost $\boldsymbol{\Psi}_{Q_k}$. The optimization in the third thread results in a new parameter and covariance estimate.

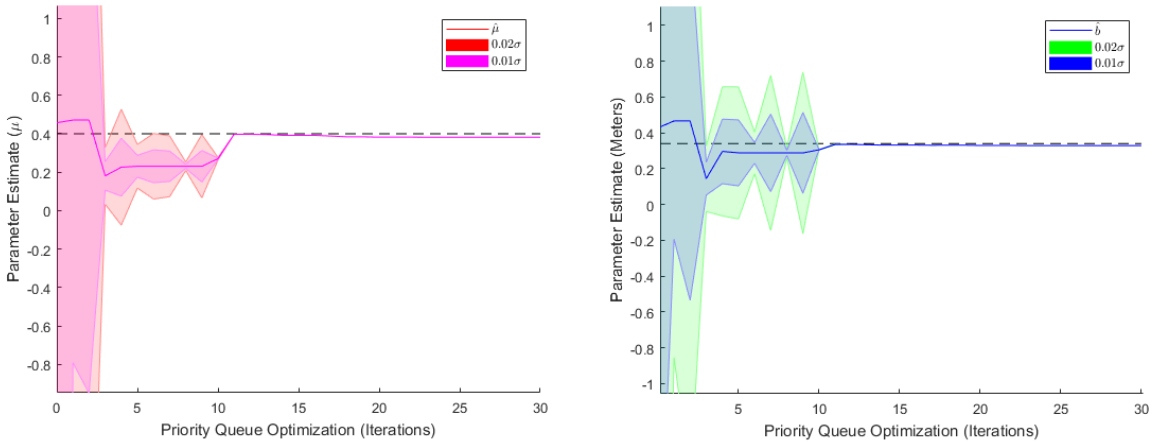

Figure 5.5: Estimation results for friction coefficient (left), and wheel base (right) of simulated car. Solid colored line represents mean value, filled areas are scaled variances and black dashed line is ground truth value. Note that error bands are drawn in 0.01σ scale to achieve graphics capable of representing both the mean value changes (in small scale) and rapid uncertainty reduction (in larger scale). Also this should be noted that despite variations in mean of estimate, It always stays in 1σ bound. After 12th iteration of priority queue optimization, both parameters reach very low uncertainty of $\sim 10^{-5}$ since dynamics of simulated car is exactly same as the one used for estimating parameters.

As a model for experiments, a physics based dynamics model using the Bullet Physics Engine [25] is constructed. To accomplish this, a high fidelity four-wheel drive vehicle model was implemented; the model includes parameters such as mass, inertia and geometry for each wheel and the chassis, dampers and springs as a suspension for each wheel and their anchor points, steering and acceleration motor speed/torque properties and speed limits and also rigid body linear/rolling friction coefficients. This model is also capable of interacting with a map of the environment and modeling contact forces [21].

We will now describe the two sets of experiments we conducted in procedural detail, and present their results.

5.4.1 Simulation

For this experiment a simulated world with a flat floor and constant friction coefficient at every point was constructed. The simulated vehicle was developed to accept steering and throttle

commands from a user who would manually drive the vehicle in the simulated environment using a gamepad. The state of the vehicle used in this case is:

$$\boldsymbol{x}_k = [\boldsymbol{x}_{ps}, \dot{\boldsymbol{x}}_{ps}, \boldsymbol{\omega}], \tag{5.12}$$

where $\boldsymbol{x}_{ps} \in \mathbb{SE}(3)$ is the pose vector of the chassis with rotations in axis-angle format, $\dot{\boldsymbol{x}}_{ps}$ is vector of linear and rotational velocities and $\boldsymbol{\omega}$ is vector of wheel velocities. In these experiments, the model for the simulated vehicle is exactly equal to those in Eq. (5.1) used to regress the calibration parameters, however the parameters are significantly perturbed. The target of this experiment is to identify the perturbation and regress the correct physical parameters.

In this experiment, user first drives the simulated car in a back and forth motion in the virtual environment until a sufficient number of priority queue updates have occurred (in our case four), suggesting a reasonable estimate may have been found. The user then drives on circular trajectories in order to consider alternate motions that may influence the parameter calibration. This sequence of driving demonstrated what is intuitively expected: in the first type of motion, both the friction and wheel base estimates are very uncertain since a back and forth motion does not render these parameters observable. In the second type of motion however, and especially when introducing some obvious sliding in the driving mode, more informative segments for these calibration parameters are added to the priority queue. Through the combination of these motions, both parameters eventually converge very close to their ground truth values with very low uncertainty.

A set of two calibration parameters are considered in this case: the car's wheel base and the ground/tire friction coefficient are simultaneously estimated. Since these parameters are known for the simulated vehicle, we may compare the resulted estimates to the ground truth values directly. Figure 5.5 shows the results for applying this calibration method with $n_C = 20$ and $n_Q = 10$ for about one minute of driving with discretization steps of 100ms. In both charts, the solid line shows the mean of the estimate and the two shaded areas with different colors show variance bounds on the estimate while a black dashed line shows the ground truth value.

5.4.2 Parkour Car Tests

For experiments with an actual system, the parkour car platform in Chapter 4 is used. As mentioned this vehicle accepts throttle and steering commands and is capable of measuring wheel speeds, current steering angle position and suspension lengths in real time. In order to get pose and velocity measurements of chassis, experiments were performed in a large open room equipped with a motion capture system. In these trials, the vehicle has been modeled with same physics based model explained in previous section and the state vector is the same as that considered in Eq. (5.12). Figure 5.6 shows the priority queue estimates of friction coefficient and wheel base on our model vehicle. The wheel base estimate of 0.32 meters is fairly close to the measured value of 0.34, this error could be due to the differences of the physics based dynamic model and the actual vehicle. As one might realize, the parameter confidence after convergence on data from physical car is low compared to simulated data, which is a result of using a less accurate model. Since comparison of the friction coefficient estimate to a ground truth value is impractical we evaluate it qualitatively. Friction coefficient is a number in $[0, 1]$ where zero means no friction. Since the experiment happened on a carpeted floor with treaded tires, a high value of $\hat{\mu}$ was expected and the priority queue estimate shows a similar value (0.85).

5.5 Conclusions

As described before parameter estimation is a crucial task for autonomous ground vehicles in the sense that wrong parameters might result in unstable systems, which use a dynamical models as their core to predict vehicle's behavior. In this paper a new method in estimating parameters has been shown which chooses most informative motion samples of the trajectory of vehicle and estimates the parameters while avoiding any biasing toward a group of parameters with higher chance to be excited. This method has been tested both in simulation and physical experiments; results show the usefulness of this method while giving a confidence level on the current estimate.

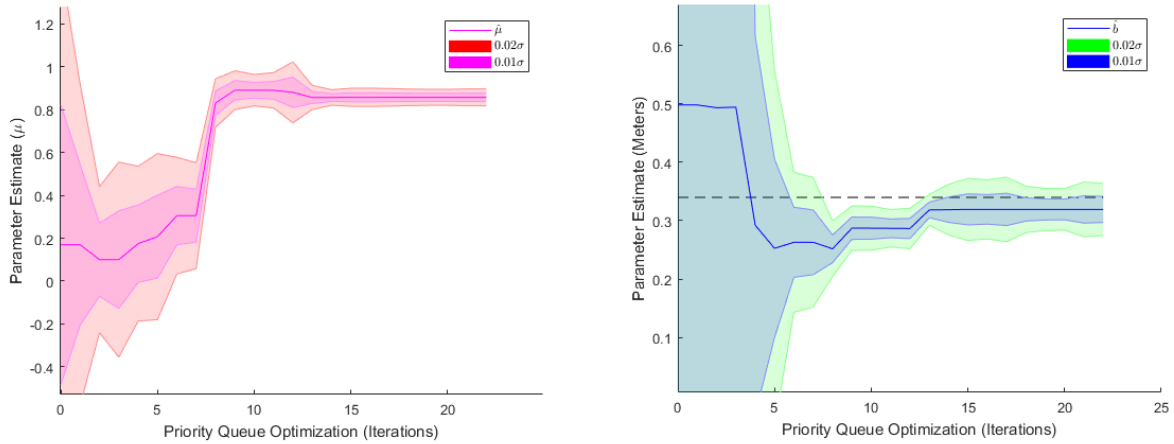

Figure 5.6: Estimation results for friction coefficient between tire and carpet (left), and wheel base of a four wheel drive $\frac{1}{8}^{th}$ scale car (right). Solid colored line represents mean value of estimate, filled areas are scaled variances and black dashed line is ground truth value of car's wheel base measured manually.

Chapter 6

Terrain Aware Model Predictive Controller

This chapter presents a new model predictive control approach for control of autonomous vehicles. Proposed method takes a geometric definition of robotic platform and some arbitrary points showing path of the vehicle is capable of controlling the vehicle through those waypoints. Some intermediate steps such as calibration of parameters (as on Chapter 5) and local planning are required to achieve the final result which are explained in more details in this chapter.

6.1 Introduction

With rapid progress in the development of autonomous vehicles, the need for more stable and accurate controllers which can accept more complicated vehicle dynamics into consideration while handling environmental uncertainties is greater than ever. Research in "enhanced driver assists" like adaptive cruise control [50] or lane-keeping [96, 113] for cars has been widely researched in the past. These control systems largely concern themselves with stabilizing a vehicle to a desired state. Some vehicle models has been significantly simplified in order to have differentiable dynamics while considering other dynamic effects as noise input to the system, e.g. [27, 72]. Common simplifications include ignoring collision forces due to their complicated effects. However, in order to achieve a stable control response it is required to use as much information from the environment as possible to make action devisions. Developments in new sensor technologies and estimation algorithms allow control systems engineers to capture more detailed information from the environment like 3D pose estimation of the robot or depth maps of the environment using cameras[40] or depth sensors [82].

Modern control algorithms should use this information in their determinations.

In contrast to classical control methods in which simplified dynamics of the autonomous vehicle is discovered in form of explicit equations, and a controller is designed to stabilize that model, model-predictive controllers (MPC) [14] solve the optimal control problem in every time step. To do this, the algorithms are given the current state of the model up to a finite time horizon, but the solution to this problem (control command) is only applied to the system during the next time step and for the step after that it needs to be recalculated. Solving an optimization problem for every time step makes MPC algorithms computationally expensive, specially when dynamics of the vehicle gets more complicated finding a solution to this problem might take a longer time than time step period. Note that control problem requires a goal and this case autonomous vehicle would need a reference trajectory to follow. Nagy et al. [84] proposed a method to parameterize a trajectory for car-like robots using cubic curvature polynomials which is a good estimate for path of the autonomous vehicle in flat surfaces but it might not be accessible by the robot in a rough terrain.

In order to find a solution to these problems, we wish to develop an intuitive, computationally efficient method that is robust to uncertainty. In most of autonomous control pipelines a global planning algorithm decides a generic path for a vehicle by setting waypoints over a map for a vehicle to traverse, then a local planning algorithm might be used to generate a path between two consecutive waypoints and later these paths are used in a controller to control the vehicle toward the reference path. Design of each stage in this approach requires a model of the vehicle in most cases the final planner/controller has model behavior backed in it and model changes require redesign/synthesis of the controller.

In this work, we present an approach to modeling, planning and control of an autonomous vehicle. Proposed method fits with a widely used structure of autonomous planning and control which is shown in Figure 6.1. Based on this model it is assumed that a user or a higher level global planning algorithm defines some middle points between current robot state and final goal state. Global planning algorithms generally use a simpler model to generate the waypoints due to high

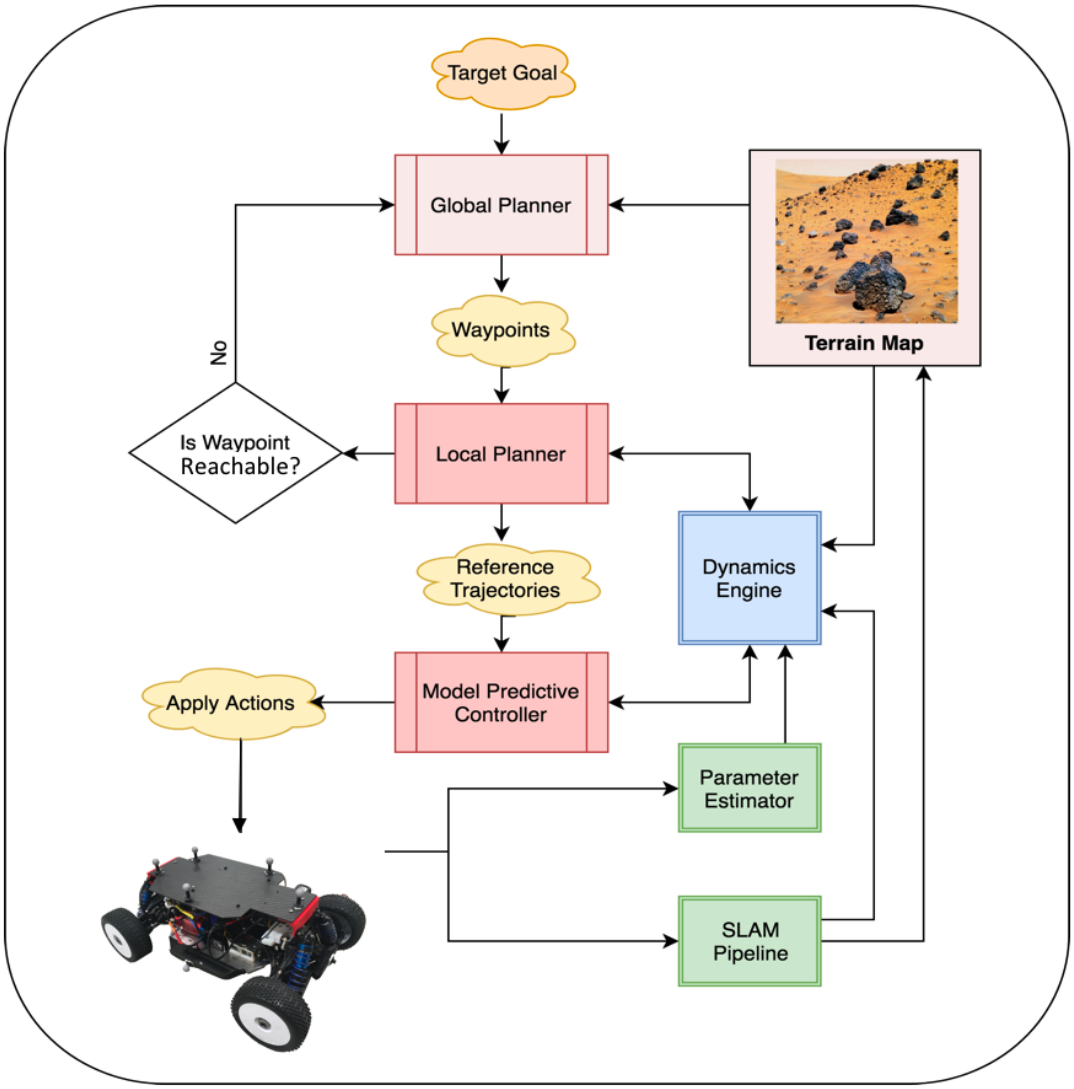

Figure 6.1: Complete planning and control pipeline structure

computational cost of the algorithm however the resulted waypoints might not be actually accessible by the physical robot. So, a local planning algorithm takes every two consecutive waypoints and using a higher fidelity model checks whether the second waypoint is reachable from first waypoint. This stage of the pipeline informs the global planner by either accepting the waypoint, rejecting it or modifying it to a better state. If the waypoint is not rejected a path between two waypoints, representing the drive path of the vehicle is generated as well as the control signals corresponding to that path. This path is later used as a reference trajectory for the model predictive controller. Blue

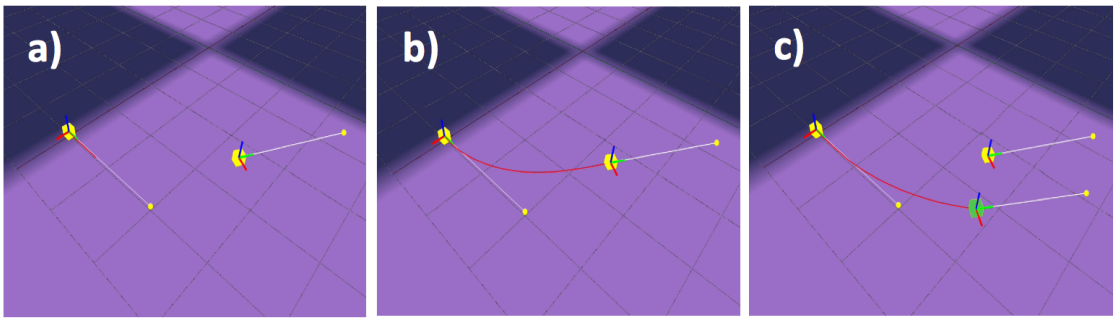

Figure 6.2: A perspective view of the graphical front-end to our MPC software. a) Waypoints with pose and velocity constraints represented by the yellow boxes and white lines respectively. b) A resolved solution after solving the 2-point BVP is designated by the red line. The presence of the red line connecting the waypoints designates that the second waypoint is reachable. c) When no solution is feasible (in this case, because environmental parameters were varied), a closest-feasible waypoint is generated in green.

box in Figure 6.1 shows dynamic model of the vehicle which is separated from Local Planning and MPC stages of the pipeline. This gives us the freedom to change the behavior of the model on the fly and both stages of the pipeline would automatically work with the new model updates. Changes can include anything from parameter updates to the actual model of the vehicle to updates in vehicle's terrain map. One might appreciate adaptability of this approach when perception of robot of its environment changes. For example adding a new sensor to the robot to sense friction of the terrain can be easily added to the terrain map and planner/controller would take in consideration this new information of terrain without changing them. More details of these two stages are given in Sections 6.3.2 and 6.3.3.

In order to model the autonomous vehicle we use a high-fidelity physics simulation engine as described in Sec. 4.2 which takes the dynamics of the autonomous vehicle into account as well as terrain roughness and contact forces applied to the autonomous vehicle. We use the Spirit physic engine based model to solve two problems: 1) the reachability problem between two waypoints constrained to autonomous vehicle dynamics, and 2) the corresponding optimal control problem.

6.2 Problem Statement

We take a model predictive approach to autonomous vehicle control; in this framework, the controller leverages a high fidelity vehicle model and prior environment maps in order to calculate the optimal control signal for the autonomous vehicle. We also develop a local planning method which finds an optimal reference trajectory between waypoints defined by a higher level global planner. Later this trajectory is used as a reference trajectory for model predictive controller.

In our framework, modeled dynamics of the autonomous vehicle should capture behavior of autonomous vehicle as accurately as possible. In order for the autonomous vehicle to travel from one waypoint to another, it is required to first examine the reachability condition between the two waypoints. If the final waypoint is reachable, then it is possible to design the controller in order to navigate the autonomous vehicle to the subsequent waypoint.

6.3 Methodology

6.3.1 Modeling

In order to predict behavior of the autonomous vehicle we use the dynamics model of the vehicle as well as the terrain in a physics engine[25] as discussed in Chapter4.2. Same model was also used in Chapter 5. In this work we evaluate the performance of the method on a front-steering four wheel drive vehicle with spring-damper suspension on each wheel attached to a cuboid chassis. The terrain can be specified as a dense three-dimensional mesh with a friction coefficient specified for each mesh surface. Similar to model given at Eq. 5.1 The governing equations of motion for autonomous vehicle can be shown in more detail as

$$\boldsymbol{x}'(t) = \Phi(\boldsymbol{x}(t), \Pi(\boldsymbol{x}(t), \boldsymbol{g}, \boldsymbol{M}), \boldsymbol{u}(t), \boldsymbol{p}), \tag{6.1}$$

in which $\Phi(\cdot)$ represents the modeled dynamics of the autonomous vehicle, $\boldsymbol{x}(t)$ is current state of autonomous vehicle, and $\boldsymbol{x}'(t)$ is the derivative of the state under the governing dynamics. Π represents contact forces applied to the autonomous vehicle and surface frictions at a given

state $\boldsymbol{x}(t)$, with gravity vector $\boldsymbol{g}$ and the map of terrain $\boldsymbol{M}$. Note also that $\boldsymbol{u}(t)$ are the steering and acceleration inputs, $\boldsymbol{p}$ is vehicle parameter vector. This parameter vector includes vehicle properties like tire and chassis geometry, as well as their inertial properties, linear and rolling friction coefficients of all bodies, suspension anchor points, suspension damping and stiffness coefficients as well as their travel limits, motor speed-torque parameters for steering and acceleration motors and chassis's center of mass location.

The control input $\boldsymbol{u}(t)$ is a continuous signal which must be discretized. As an added challenge, sampling this signal in high frequency would result in a problem with very high computation cost. To avoid this, we apply a parametrization of this signal by a quadratic Bézier curve [53]. We may then rewrite $\boldsymbol{u}(t)$ as

$$\boldsymbol{u_p}(t) = (1-t)^2 p_0 + 2(q-t)tp_1 + t^2 p_2. \tag{6.2}$$

We will later discuss the implications this has in simplifying the corresponding boundary value problem; in particular, it reduces the problem size significantly by solving for p_i, $i \in [0, 2]$ rather than discrete points on $\boldsymbol{u}(t)$. Order of the curve must be chosen such that the resulted signal is capable of generating appropriate reference for underlying actuator. This can be decided from response time of the actuator and inn our case a quadratic curve turns to be the right choice.

It is possible to solve Eq. (6.1) with algorithms like MLCP Dantzig, MLCP Projected-Gauss-Seidel or Sequential Impulse Solver, which are already implemented in the Bullet physics engine [24]. Note that as with all algorithm choices, swapping the solver has an effect on the speed of computation and accuracy of solution.

Software library defined in Chapter 4.2 can generate trajectories of the vehicle by defining initial condition of the vehicle, map of the world and parameters p_i and depending on discretization step defined in the library. Since the integration step doesn't have much processing overhead, input Bézier curve can be sampled as fine as desired to be applied to the vehicle along the path.

6.3.2 Local Planning

Given two waypoints on the terrain mesh we evaluate reachability of waypoints constrained to autonomous vehicle dynamics. We also come up with a reference trajectory for the autonomous vehicle to be followed by the MPC algorithm. By design, a waypoint $\boldsymbol{w_i}$ defines a pose and velocity constraint as

$$\boldsymbol{w}_i = [\boldsymbol{d}_i, \boldsymbol{r}_i, \dot{\boldsymbol{d}}_i, \dot{\boldsymbol{r}}_i], \tag{6.3}$$

in which $\boldsymbol{d}_i \in \mathbb{R}^3$ is position vector of the autonomous vehicle in global Cartesian coordinates, $\boldsymbol{r}_i \in \mathbb{R}^4$ is vector of axis-angle rotation, $\dot{\boldsymbol{d}}_i \in \mathbb{R}^3$ is linear velocity constraint in the waypoint and $\dot{\boldsymbol{r}}_i \in \mathbb{R}^4$ is rotational velocity at the waypoint.

In order to evaluate reachability of waypoint $\boldsymbol{w}_{i+1}$ from $\boldsymbol{w}_i$ we must construct a Boundary Value Problem using dynamic formulation in Eq. (6.1). To construct a BVP problem we take an optimization approach. A cost function can be defined between two consecutive waypoints such that minimizing it would provide the solution for BVP problem. For this we fist initialize the vehicle in the state specified by first waypoint similar to Eq. (6.3) and given a initial parameterized control for the vehicle we integrate the model forward to get an end state. Then we define a cost between resulted end state vector (current state) and state vector of next waypoint (desired state) (Figure 6.3). Minimizing this cost would reduce the error this error while updating parameters of beziér control signal. Eventually, this procedure is terminated if the cost value is below some predefined value or step size of the optimization is very small. Then, solution of the optimization problem as well as trajectory of the vehicle in last iteration of optimization is stored to be used by the controller in next stage of the pipeline.

Cost function can be shown as Eq. 6.4. Dropping the Π and p arguments for simplicity and replacing $\boldsymbol{u}(t)$ from Eq. (6.2), we have:

$$\begin{aligned} p^*_{bvp} = \min_{p} \quad & \lambda \| \boldsymbol{x}_{\boldsymbol{w}_{i+1}(t_{i+1})} \boxminus \boldsymbol{\Phi}(\boldsymbol{x}_{\boldsymbol{w}_i}(t_i), \boldsymbol{u}_{\boldsymbol{p}}(t)) \|_2 \\ & + \gamma \| t_{i+1} - t_i \|_2 \\ & + \| L(\boldsymbol{u}_{\boldsymbol{p}}) \|_2, \end{aligned} \tag{6.4}$$

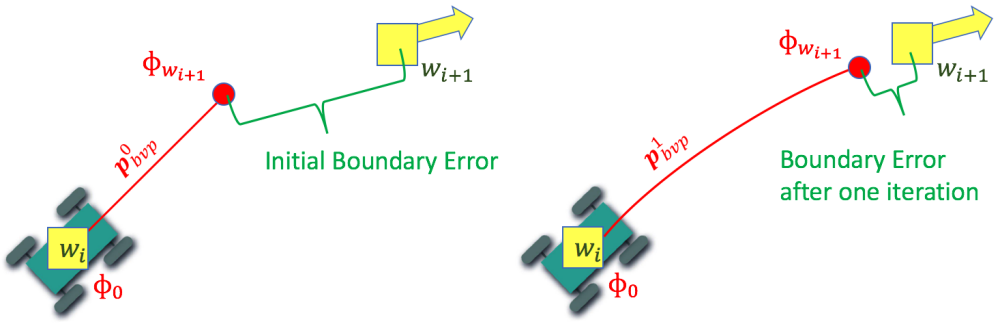

Figure 6.3: Cost function definition for Boundary Value Problem. BVP before optimization (Left) and same problem after one iteration of optimization (right)

where $\boldsymbol{x}_{\boldsymbol{w}_i}(t_i)$ is state of vehicle at time t_i with boundary conditions $\boldsymbol{w}_i$, and $\boldsymbol{\lambda}$ and scalar γ are weighting parameters. The operator $\boxminus$ calculates the velocity and pose error between two vehicle states. The first term in the cost function of optimization problem in Eq. (6.4) defines an error vector between waypoint $\boldsymbol{w}_{i+1}$ and solution of dynamic model in Eq. (6.1) starting from $\boldsymbol{w}_i$ with parameterized input $\boldsymbol{u}_{\boldsymbol{p}}(t)$. Second term is a regularization term which results in a time-optimal trajectory and third term accounts for actuator limits by largely penalizing undesired values in actuator's input space. This function is applied to control points on the Bézier curve and as shown in Section 3.2.1 since the whole curve is subsumed with convex hull of the control points this guarantees that the underlying control signals are going to be inside actuator limits. Actuator limit penalty function $L(\boldsymbol{u}_{\boldsymbol{p}})$ is a piecewise linear function as

$$L(\boldsymbol{u}_{\boldsymbol{p_i}}) = \begin{cases} -a \cdot \boldsymbol{u}_{\boldsymbol{p}_i} + \alpha_l & \boldsymbol{u}_{\boldsymbol{p}_i} \leq \alpha_l \\ 0 & \alpha_l < \boldsymbol{u}_{\boldsymbol{p_i}} < \alpha_h, \\ a \cdot \boldsymbol{u}_{\boldsymbol{p}_i} + \alpha_h & \boldsymbol{u}_{\boldsymbol{p}_i} \geq \alpha_h \end{cases} \tag{6.5}$$

where a defines slope of the cost when actuator's high and low limits α_h and α_l are violated. Eq. (6.4) is solved via an optimization framework which is discussed later. Even though formulation in Eq. (6.5) is non-smooth in boundaries, but since a numerical differentiation is used in calculating derivatives of the cost, this would not be a problem.

After solving Eq. (6.4), if a solution is achieved with cost value lower than some ϵ, then $\boldsymbol{w}_{i+1}$

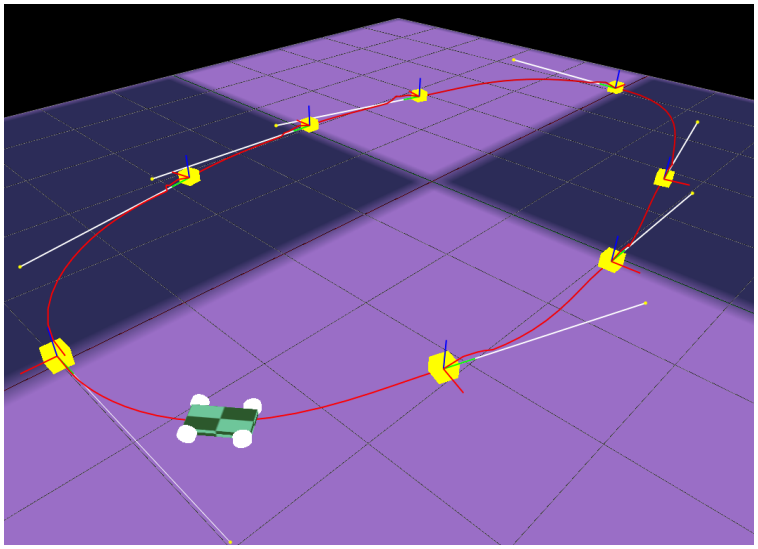

Figure 6.4: Result of solving BVP problem for eight consecutive waypoints(yellow) and resulted trajectories(red) which are used as a reference for controller

is considered reachable (Figure 6.2b). However, if the error is greater than ϵ, then the waypoint is considered not reachable (see Figure 6.2). In this case, one can decide whether to use the solution to adjust $\boldsymbol{w}_{i+1}$ to the current solution and continue to next waypoint or terminate the algorithm. In addition to solving for p^* from Eq. (6.4), we also store the trajectory of the autonomous vehicle between every two consecutive waypoints:

$$\boldsymbol{\tau}_j = \Phi_{\delta_j}(\boldsymbol{x}_{\boldsymbol{w}_i}(t_i), \boldsymbol{c}(p^*_{bvp})) \quad j \in \{0...\lfloor \frac{t_{i+1} - t_i}{\delta_t} \rfloor\}, \tag{6.6}$$

where $\delta_j = t_i + j * \delta_t$ and δ_t are the time step sizes to sample the trajectory. $\Phi_{\delta_j}(\cdot)$ is the state after time δ_j, with the input $\boldsymbol{c}(p^*)$ applied to dynamics model. Trajectory $\boldsymbol{\tau}_j$ does not need any additional computation and can be extracted from last iteration of the optimization in Eq. (6.4).

In order to test the planning algorithm, we used the model in Sec. 4.2 (Figure 6.4). Implementing the local planning approach from Sec. 6.3.2 demonstrated that we can validate reachability of a waypoint and also come up with a reference trajectory for the autonomous vehicle. We also parameterized the control input by Bézier curves [22] to reduce dimensionality of search space from infinite dimensional continues functions to quadratic curves. Figure 6.4 shows the resulted reference trajectories for eight consecutive waypoints.

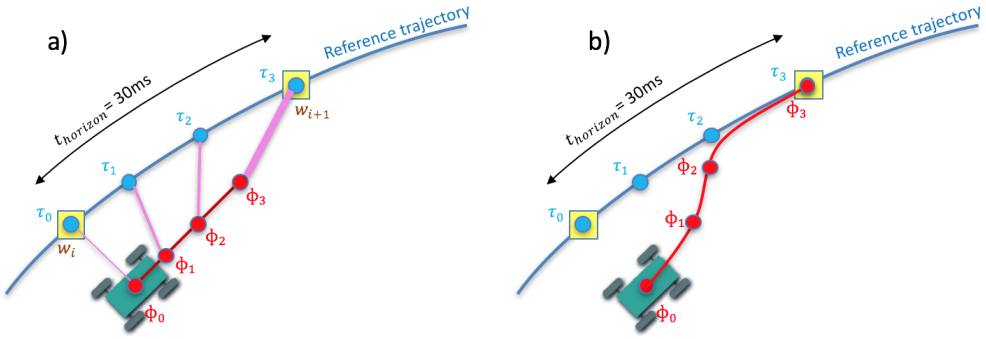

Figure 6.5: Model predictive controller cost function with (red dots) discretized vehicle trajectory points, (blue dots) discretized reference trajectory points, (yellow box) waypoint, (purple line) weighted costs. a) autonomous vehicle with initial control input results in red trajectory before optimization. b) autonomous vehicle's trajectory converges to reference trajectory after convergence

6.3.3 Model Predictive Control

Once a reference trajectory has been calculated as in Eq. (6.6), we design a model predictive controller such as that employed by [15] to stabilize the autonomous vehicle to the reference trajectory. For this we define a "cost-to-go" function as in Eq. (6.7) and construct an optimization problem in order to find the proper control signal. Stability of the controller is proven later in this chapter.

Cost-to-go function of proposed MPC is given as:

$$p^*_{mpc} = \min_{p} \sum_{j=0}^{K} \rho_j \| \tau_j \boxminus \mathbf{\Phi}_{\delta_j}(\boldsymbol{x}_{\boldsymbol{w}_i}(t_i), \boldsymbol{c}(p)) \|_2 + \| L(\boldsymbol{u}_{\boldsymbol{p}}) \|_2, \tag{6.7}$$

where ρ_j is weighting coefficient and K is the length of the sequence of states over the discretized time horizon. Since most vehicles have control constraints arising from the steering angle of the wheels, one might apply less weight to closer points to the current position of autonomous vehicle since control signal can not reduce the error due to nonholonomic constraint of wheels. Weights can also be adjusted according to importance of pose or velocity error in the problem.

Figure 6.5 shows the trajectory of a four-wheeled autonomous vehicle before and after solving Eq. (6.7). Once a control command $\boldsymbol{c}(p^*_{mpc})$ is calculated, it gets executed on the autonomous vehicle

for a short period of time—in our case, until next control command is calculated.

6.3.4 Optimization

For both BVP and MPC problems we minimize a cost function in a least-squares sense. We solve these optimization problems using the Levenberg-Marquardt [83] algorithm. We also must calculate the Jacobian of the error function with respect to its parameters $\boldsymbol{J} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{p}}$, for which we take a finite differencing approach to calculate derivatives of Eq. (6.1). Finite differencing on physics engine of Sec. 4.2 is calculated by perturbing a control signal parameter with a small value ($\epsilon = 0.05$) and finding difference of state vectors in state space.

Since columns of the Jacobian matrix are not correlated with one another, finite differencing can occur in parallel. Our multi-threaded algorithm is capable of calculating the Jacobian at each iteration of optimization quickly.

6.3.5 Trajectory Tracking vs. Maneuver Regulation

A trajectory tracking controller has been defined in Sec. 6.3.3 where discrete points on trajectory are spaced in a timed fashion. This means that if the vehicle falls behind in time of execution an appropriate control signal will be selected such that it minimizes the distance between the time varying reference and vehicle's current position. Another perspective for the controller is one were vehicle always tries to merge to a reference maneuver where the trajectory itself is not timed and control goal is to push the vehicle onto the maneuver as soon as possible. A maneuver regulator can be implemented in Method of Sec. 6.3.3 by first defining a continuous reference trajectory between waypoints (parameterized curve fitting through discrete points of Planner in Sec. 6.3.2) and then re-defining the operator $\boxminus$ such that it returns the closest distance between vehicle's current discrete trajectory points and the continuous reference curve. In this mode, velocity of the vehicle is regulated by defining a constant linear forward velocity for the maneuver rather than velocity terms in Eq. (6.3).

6.3.6 Adaptive Horizon MPC

One of the important parameters in performance of MPC methods is choice of horizon. A useful approach should try to minimize the required length of horizon as much as possible to achieve a high performance. In order to choose the correct horizon length we need to understand under what conditions the controller would be stabilizing. For stability of receding horizon controller there are two main approaches where one uses a terminal cost in the cost function terms and minimizing the resulted cost would give a stabilizing controller [100]. In another approach, a terminal constraint can be added to the cost function where if tail of the horizon falls in the final set then resulted MPC would be stabilizing, However defining this final set can not be easily calculated for other than some well known systems with simple dynamics. Also, Jadbabaie and Hauser [51] give a proof for existence of a finite horizon in which the corresponding receding horizon scheme is stabilizing without the use of a terminal cost or constraint but it is not known how length of horizon can be determined.

In this thesis we use the result of a recent work from Krener [64], which uses a local controller and a corresponding lyapunove function in a terminal cost and based on a heuristic decides whether tail of the horizon is in basin of attraction of the local controller or not and extends or retracts the horizon length. As per [64] an adaptive horizon can be selected by

(1) Assuming set of states $\mathbb{X}$ and set of terminal states $\mathbb{X}_f$ are colsed, $\mathbb{X}_f \in \mathbb{X}$, set of inputs $\mathbb{U}$ is compact and $\mathbb{X}$, $\mathbb{X}_f$ and $\mathbb{U}$ contain neighborhoods of their respective origins. However it is not required that $\mathbb{X}_f$ is known explicitly.

(2) A discrete timed controlled dynamics $f(x, u)$, a lagrangian l(x,u), a constraint pair $(h(x, u), Y)$ and a terminal cost $V_f(x)$ satisfying following assumption (Assumption 2.2 [100]).

Assumption: The functions, discrete dynamics $f(x, u)$, discrete dynamics $l(x, u)$, constraints $h(x, u)$ and terminal lyapunov function $V_f(x)$ are continuous on some open set containing $(X) \times \mathbb{U}$, $l(x, u)$ is nonegative definite in (x, u) and positive definite in u on this open set, V_f is positive definite on $\mathbb{X}_f$ and $f(0, 0) = 0$, $l(0, 0) = 0$, $V_f(0) = 0$.

(3) A terminal feedback $u = \kappa_f(x)$ and a class K_∞ function $\alpha(\cdot)$ defined for all $x \in \mathbb{X}_f$ and satisfying

$$\begin{aligned} V_f(x) &\geq \alpha(|x|) \\ f(x, \kappa_f(x)) &\in \mathbb{X}_f \\ V_f(x) - V_f(f(x, \kappa_f(x))) &\geq \alpha(|x|) \\ h(x, \kappa_f(x)) &\in (Y) \end{aligned} \tag{6.8}$$

AHMPC method proposed by Krener [64] does not require an explicit definition of $d\mathbb{X}_f$ however it requires a pair of terminal cost funciton $V_f(x)$ and controller $\kappa_f(x)$ to approximately solve infinite horizon dynamic program equation on some neighborhood of the origin. An example of such a pair can be constructed using linear quadratic regulator where LQR cost can be used as the terminal cost term and resulted local controller can be used as the control law $\kappa_f(x)$. Since in general it is not possible to find a terminal set $\mathbb{X}_f$ from terminal pairs $V_f(x)$ and $\kappa_f(x)$ such that conditions (1) to (3) are satisfied, a method to check whether state of the system in horizon is contained in $\mathbb{X}_f$ has been proposed.

Based on this method, once the MPC problem has been solved for horizon N, given that V_f is defined in $x(N)$ then the tail can be extended using the terminal feedback $u = \kappa_f(x)$ for another L steps. If terminal state $x(N)$ is not in $\mathbb{X}_f$ then horizon N needs to increase and MPC problem should be solved before extending tail. Once the tail is extended following conditions are checked

$$\begin{aligned} V_f(x_N(k)) &\geq \alpha(|x_N(k)|) \\ V_f(x_N(k)) - V_f(x_N(k+1)) &\geq \alpha(|x_N(k)|) \end{aligned} \tag{6.9}$$

for $k = N, \ldots, N + L + 1$. This condition checks if value of the terminal cost has sufficient decrease and if so it is concluded that $x_N \in \mathbb{X}_f$. Otherwise it is assumed that $x_N \notin \mathbb{X}_f$ and horizon should increase to $N + 1$ steps and this method should be checked again. Length of tail L is a design parameter and is selected by user.

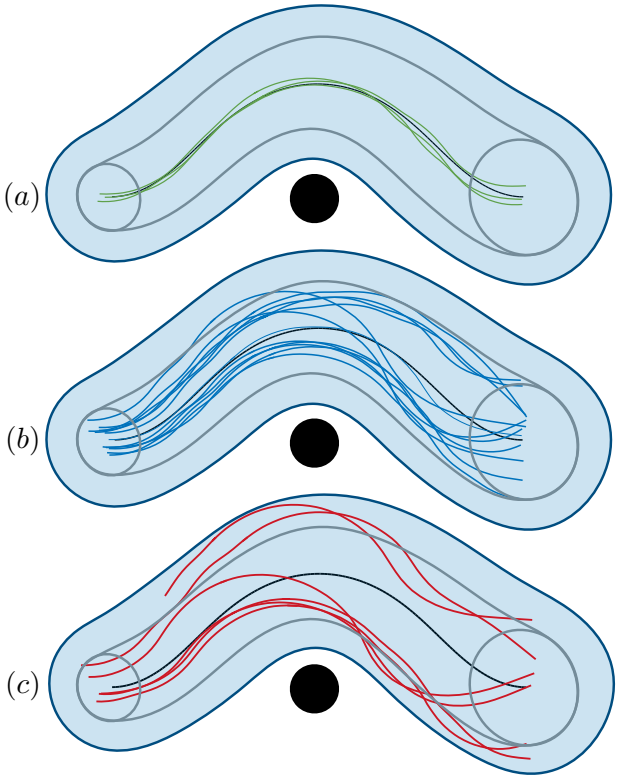

Figure 6.6: Projection of trajectories on $x - -y$ plane generated by Parkour car for the obstacle avoidance problem. Funnel boundary is shown in gray and as long as the center of car is in the funnel, the whole body of the car remains in the blue region. (a) guaranteed traces, (b) not guaranteed but safe traces, (c) not guaranteed and unsafe traces (from [98]).

6.3.7 Synthesizing a Local Controller

As explained in Sec. 6.3.6 in order to choose an adaptive horizon, a pair of local lyapunov function and a local controller is required. For this, result from Ravanbakhsh *et al.* [99] is used to design a local controller. In [98] we introduce a control funnel function which in addition to the lyapunov function and the control law, a safe set is produced which guarantees that trajectories of the state would stay inside safe set if the system starts from a state in input of the funnel. Reader must refer to [98] for more details on implementation of this approach but we will discuss benefits of using this approach in a terminal cost term of Sec. 6.3.6.

The control funnel function of [98] uses thedemonnstrator-based learning framework of [99] to synthesize the control funnel functions. The BVP problem in Sec. 6.3.2 in combination with

model of Sec. 4.2 is used as a Demonstrator stage of [99] in order to find an optimal control action. Later a Learner stage of pipeline tries to find a control lyapunov candidate function $V(x)$ and in next step this function is checked to verify that it is indeed a CLF (Control Lyapunov Function). Even though, Demonstrator uses the complicated model of Sec. 4.2 but the CLF function found in Learner stage uses a kinematic bicycle model similar to Eq. 2.1 to find and verify the CLF. Resulted CLF is used to get a feedback law based on result of Artsein [5]. Resulted control law maps a set of start states to final states such that if vehicle starts in start set it is guaranteed that it would end up inside end set. Experiments on the model vehicle of Chapter. 4 has been done to verify effectiveness of this method. Figure 6.6 shows results of stabilizing the vehicle to an obstacle avoidance reference trajectory. Synthesized controller can also be used in form of trajectory tracking or maneuver regulation(path folowing) in the sense described in Sec. 6.3.5. Experimental results of two described modes of controllers are shown in Figure. 6.7 which represent successful stabilization of vehicle to a circular reference trajectory. Different modes of approaching control problem has been discussed in Sec. 6.3.5, and difference between trajectories of two approaches starting from same initial conditions can be observed in experiments with parkour car as well (Figure. 6.8).

As it has been discussed in Sec. 6.3.6 a lyapunov function or corresponding control law can be used in deciding horizon length of MPC method in Sec. 6.3.3 such that it is stabilizing. For the purpose of this thesis, the end state of the MPC horizon x_N can be checked in synthesized CLF defined in this section to confirm if the pair (x_N, u_N) gives a lower value for the CLF than expected threshold which would guarantee that the end state is indeed in start set of the local control funnel function. If CLF returns a larger value than expected threshold then horizon of the MPC problem can be extended.

6.4 Conclusions

Our experiments showed that dynamics engines may be used to model complicated behavior of autonomous vehicles and the same model could be used in local planning and control problems without further modification. This simplifies the control problem to defining geometric and dynamic

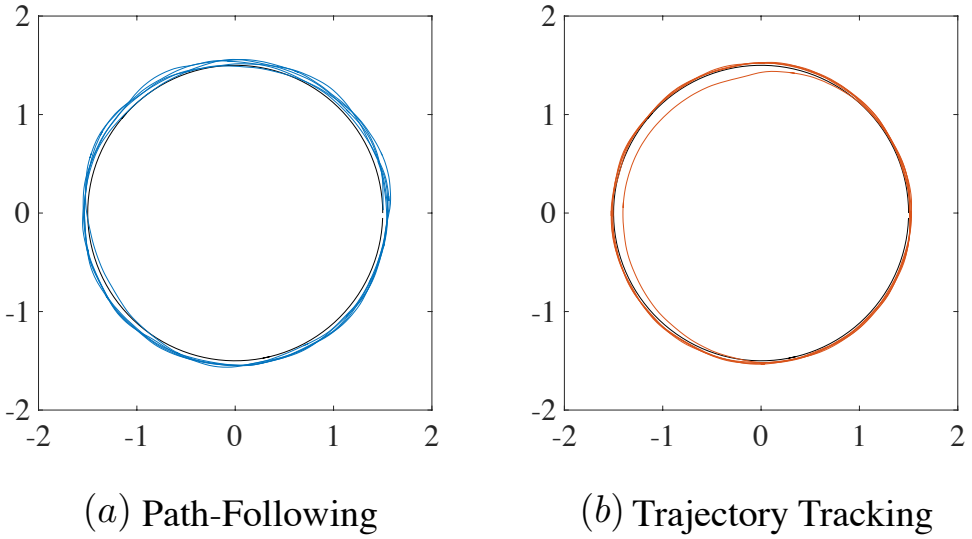

Figure 6.7: (a)Maneuver Regulation (Path Following) controller and (b) trajectory tracking controller stabilizing parkour car to circular trajectory, starting from distant initial condition. As discussed in Sec. 6.3.5 a trajectory tracking controller will take shortcut trajectories to compensate for delay in timed trajectory reference. (from [98]).

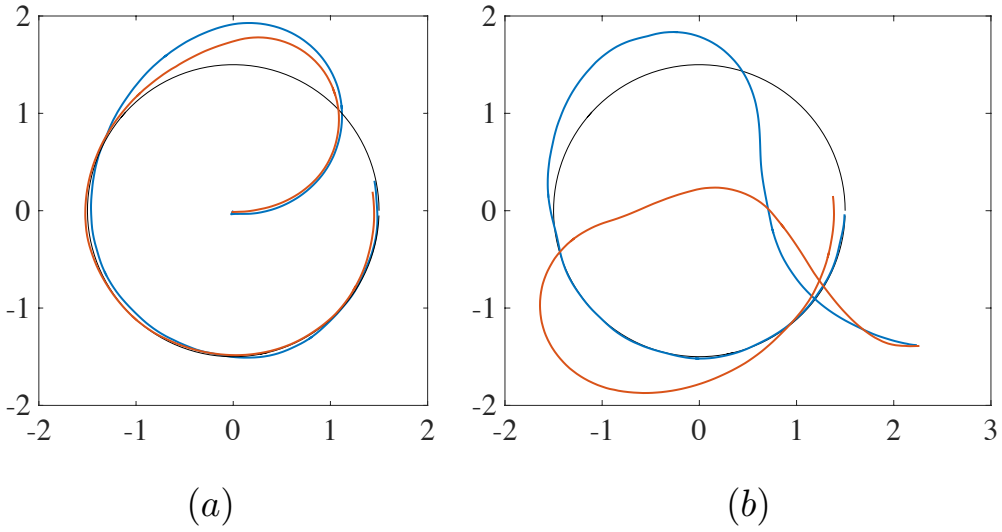

Figure 6.8: (a)Maneuver Regulation (Path Following) controller and (b) trajectory tracking controller stabilizing parkour car to circular trajectory. (from [98]).

properties of the autonomous vehicle and defining some waypoints for the robot to follow.

Some of the challenges which have been observed was execution cost of model in Sec. 4.2. Using this model in the control pipeline of Sec. 6.3.3 and running both in simulation and also on

parkour car we were able to stabilize the vehicle to a circular trajectory. In simulation this method is able to stabilize to desired path as expected when the control signal computation time is ignored, but due to high control signal latency (0.2 sec) in physical experiments, Tests has only been limited to slow speed (0.5 - 1 $\frac{m}{s}$) drives with horizon of 0.9 sec. The main bottle neck for this problem is heavy calculation required by bullet physics engine for integration of model which includes many constraint and contact problem solving. Despite the benefits that exist for modeling contact forces, one might ignore them to achieve higher processing speed and as a result a faster control loop. In next chapter we will develop a simplified model of vehicle which also can be defined geometrically and it can operate in about three orders of magnitude faster than the model in Sec. 4.2.

Chapter 7

Vehicle Dynamic Modeling and Analysis

One of the drawbacks of using physics engine based model is the calculation cost. Physics engine based models due to generic design of their software structure and object to object contact calculation take a considerable time in executing model integration in time. This causes an large drop in execution frequency of the controller and eventually bad performance of the controller. In order to solve this issue we design a new model for the vehicle while maintaining desired properties. In addition to low execution time we require it to be adaptable to model changes without need of a expert system modeler. Also, we would like to be able to define geometric relation between different parts of system and extract mathematical relations of the system automatically. And lastly, model should capture dynamic behaviors of the physical system accurately.

To achieve this goal we use a modeling approach called Bond Graph. In bond graph modeling, different parts of vehicle and force and velocity constraints between them can be described using a graph. For details on how this method is implemented readers can refer to [55]. A simple example of extracting a graph for a mass-spring-damper system is given in Section 3.4 . Once the model is extracted from the bond graph we investigate similarity of a more detailed sports car model in [106] to verify its function.

A four wheel drive vehicle can be modeled in simple form as a 2-D model shown in Figure 7.1.

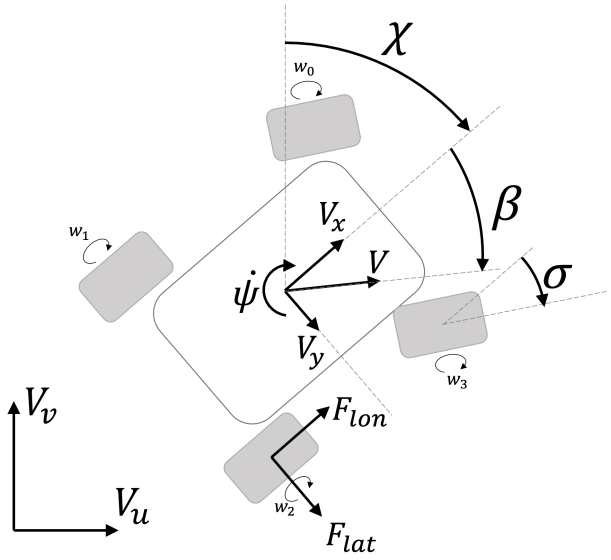

Figure 7.1: Vehicle's configuration as well as parameter names.

We parameterize the wheels with a distance and angle metric from geometric center of the car as shown in Figure 7.2

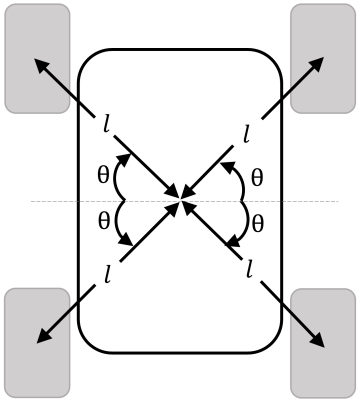

Figure 7.2: Wheel location parametrization

Then Bond Graph model of the system can be designed as in Figure. 7.3. Engine torque is applied through a torque equalizing junction 0 to four longitudinal models of the wheels which represents the model of a transmission in the car. Forces generated from tires are applied to linear and rotational inertial elements in the model shown as I_x, I_y and J_z in the graph of Figure 7.3. "TF" elements in this graph are transformer elements with coefficients of c_{ij} which correspond to

kinematic of the model which will be described later. Tire models are shown as boxes $TireLon$ and $TireLat$ and their corresponding graph is shown in Figures 7.4 and 7.5.

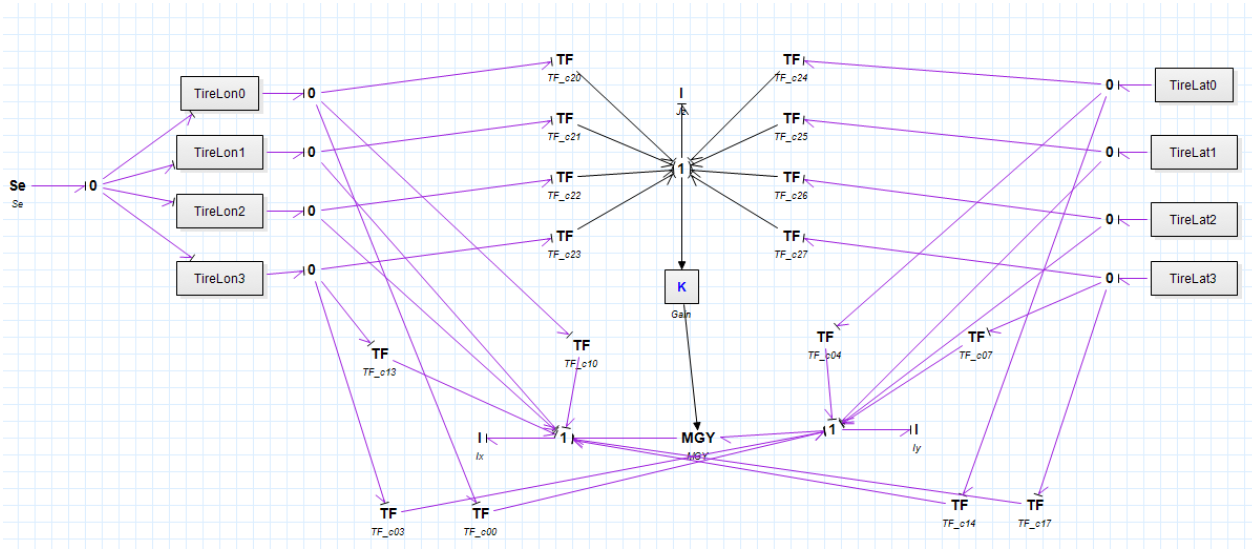

Figure 7.3: Bond Graph Model of four wheel drive car

Lateral model of the tire is shown at Figures 7.4 which is modeled independent of longitudinal force. Longitudinal tire model accepts a torque input from transmission and outputs the amount of force generated by tire. TF object is a kinematic parameter defining radius of the wheel, I is inertia of the wheel and R is friction coefficient in both models. 7.5

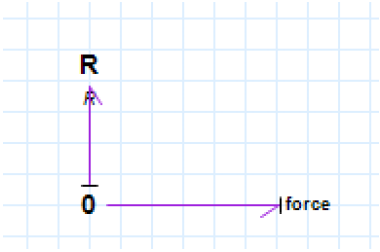

Figure 7.4: Bond Graph model of wheel lateral dynamics

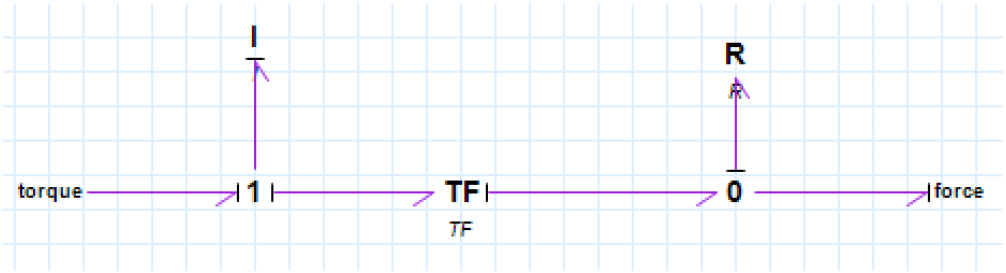

Figure 7.5: Bond Graph model of wheel llongitudinal dynamics

As mentioned, R components in both graphs of Figure 7.4 and 7.5 represent the friction coefficient of tire which can be set to a constants value for a linear tire model or more complicated model like Pacejka tire model which can be represented as follows for longitudinal friction in tire coordinate frame:

$$\mu_{n_i} = d_x \sin(c_x \arctan(b_x(1-e_x)\kappa_i + e_x \arctan(b_x\kappa_i))) \tag{7.1}$$

and lateral friction as

$$\mu_{t_i} = d_y \sin(c_y \arctan(b_y(1-e_y)\beta_i + e_y \arctan(b_y\beta_i))) \tag{7.2}$$

It is assumed that tires have same friction properties.

Given Bond Graph of of the system we can extract dynamic equations of system in about $1ms$. Resulted ODE of system is in multiple short equations which can be used directly to predict behavior of the model but for sake of better representation we simplify it to the following equation.

$$\dot{w_0} = \tau - r\mu_{n_0}(r\frac{w0}{I_{t0}} - c_{00}\frac{p_y}{I_y} - c_{10}\frac{p_x}{I_x} - c_{20}\frac{p_\omega}{I_\omega}); \tag{7.3}$$

$$\dot{w_1} = \tau - r\mu_{n_1}(r\frac{w_1}{I_{t1}} - \frac{p_x}{I_x} - c_{21}\frac{p_\omega}{I_\omega}); \tag{7.4}$$

$$\dot{w_2} = \tau - r\mu_{n_2}(r\frac{w_2}{I_{t2}} - \frac{p_x}{I_x} - c_{22}\frac{p_\omega}{I_\omega}); \tag{7.5}$$

$$\dot{w_3} = \tau - r\mu_{n_3}(r\frac{w_3}{I_{t3}} - c_{03}\frac{p_y}{I_y} - c_{13}\frac{p_x}{I_x} - c_{23}\frac{p_\omega}{I_\omega}); \tag{7.6}$$

$$
\begin{aligned}
\dot{p_x} =& \mu_{n_0} c_{10} (r \frac{w_0}{I_{t0}} - \frac{p_y}{I_y} c_{00} - \frac{p_x}{I_x} c_{10} - \frac{p_\omega}{I_\omega} c_{20}) \\
&+ \mu_{n_2} (r \frac{w_2}{I_{t2}} - \frac{p_x}{I_x} - \frac{p_\omega}{I_\omega} c_{22}) \\
&+ \mu_{n_1} (r \frac{w_1}{I_{t1}} - \frac{p_x}{I_x} - \frac{p_\omega}{I_\omega} c_{21}) \\
&- \mu_{t_3} c_{17} (\frac{p_\omega}{I_\omega} c_{27} + \frac{p_x}{I_x} c_{17} + \frac{p_y}{I_y} c_{07}) \\
&+ -\mu_{t_0} c_{14} (\frac{p_\omega}{I_\omega} c_{24} + \frac{p_x}{I_x} c_{14} + \frac{p_y}{I_y} c_{04}) \\
&+ \mu_{n_3} c_{13} (r \frac{w_3}{I_{t3}} - \frac{p_y}{I_y} c_{03} - \frac{p_x}{I_x} c_{13} - \frac{p_\omega}{I_\omega} c_{23}) \\
&+ m \frac{p_\omega}{I_\omega} \frac{p_y}{I_y};
\end{aligned} \tag{7.7}
$$

$$
\begin{aligned}
\dot{p_y} =& - \mu_{t_0} c_{04} \frac{p_y}{I_y} (c_{24} \frac{p_\omega}{I_\omega} + c_{14} \frac{p_x}{I_x} + c_{04}) \\
&- \mu_{t_2} (c_{26} \frac{p_\omega}{I_\omega} + \frac{p_y}{I_y}) \\
&- \mu_{t_1} (c_{25} \frac{p_\omega}{I_\omega} + \frac{p_y}{I_y}) \\
&+ \mu_{n_3} c_{03} \frac{p_\omega}{I_\omega} (r \frac{w_3}{I_{t3}} - c_{03} \frac{p_y}{I_y} - c_{13} \frac{p_x}{I_x} - c_{23}) \\
&+ \mu_{n_0} c_{00} \frac{p_\omega}{I_\omega} (r \frac{w_0}{I_{t0}} - c_{00} \frac{p_y}{I_y} - c_{10} \frac{p_x}{I_x} - c_{20}) \\
&- \mu_{t_3} c_{07} (\frac{p_\omega}{I_\omega} c_{27} + \frac{p_x}{I_x} c_{17} + \frac{p_y}{I_y} c_{07}) \\
&- m \frac{p_\omega}{I_\omega} \frac{p_x}{I_x};
\end{aligned} \tag{7.8}
$$

$$
\begin{aligned}
\dot{p_\omega} = &- \mu_{t_3} c_{27} (c_{27} \frac{p_\omega}{I_\omega} + c_{17} \frac{p_x}{I_x} + c_{07} \frac{p_y}{I_y}) \\
&+ \mu_{n_0} c_{20} (r \frac{w_0}{I_{t0}} - (c_{00} \frac{p_y}{I_y} + c_{10} \frac{p_x}{I_x} + c_{20} \frac{p_\omega}{I_\omega})) \\
&+ \mu_{n_1} c_{21} (r \frac{w_1}{I_{t1}} - (\frac{p_x}{I_x} + c_{21} \frac{p_\omega}{I_\omega})) \\
&+ \mu_{n_2} c_{22} (r \frac{w_2}{I_{t2}} - (\frac{p_x}{I_x} + c_{22} \frac{p_\omega}{I_\omega})) \\
&+ \mu_{n_3} c_{23} (r \frac{w_3}{I_{t3}} - (c_{03} \frac{p_y}{I_y} + c_{13} \frac{p_x}{I_x} + c_{23} \frac{p_\omega}{I_\omega})) \\
&- \mu_{t_0} c_{24} (c_{24} \frac{p_\omega}{I_\omega} + c_{14} \frac{p_x}{I_x} + c_{04} \frac{p_y}{I_y}) \\
&- \mu_{t_1} c_{25} (c_{25} \frac{p_\omega}{I_\omega} + \frac{p_y}{I_y}) \\
&- \mu_{t_2} c_{26} (c_{26} \frac{p_\omega}{I_\omega} + \frac{p_y}{I_y});
\end{aligned}
\tag{7.9}
$$

In the equation above w_i s are rotational wheel velocity, τ is engine torque, r is wheel radius, I correspond to inertia of related rigid body, μ is corresponding friction coefficient, p_x p_y p_ω are linear and rotational momentum of chassis, m is mass of chassis and parameters c_{ij} correspond to the terms in i^{th} row and j^{th} column of kinematic matrix K below

$$
\begin{pmatrix} F_x \\ F_y \\ F_\omega \end{pmatrix} = K(\sigma, \theta) \begin{pmatrix} {F_0}^{lon} \\ {F_1}^{lon} \\ {F_2}^{lon} \\ {F_3}^{lon} \\ {F_0}^{lat} \\ {F_1}^{lat} \\ {F_2}^{lat} \\ {F_3}^{lat} \end{pmatrix}, \tag{7.10}
$$

where

$$
K(\sigma, \theta) = \begin{pmatrix} \cos(\sigma) & 1 & 1 & \cos(\sigma) & -\sin(\sigma) & 0 & 0 & -\sin(\sigma) \\ \sin(\sigma) & 0 & 0 & \sin(\sigma) & \cos(\sigma) & 1 & 1 & \cos(\sigma) \\ d\cos(\theta - \sigma) & d\cos(\theta) & -d\cos(\theta) & -d\cos(\theta + \sigma) & d\sin(\theta - \sigma) & -d\sin(\theta) & -d\sin(\theta) & d\sin(\sigma + \theta) \end{pmatrix}, \tag{7.11}
$$

In order to calculate tire friction coefficients Pacejka tire model requires forward slip κ and side slip angle β which can be written as

$$\begin{aligned} \kappa_0 = \kappa_3 &= \frac{|r \cdot w_i - v_x \cdot \cos(\sigma)|}{\max(|r \cdot w_i|, |v_x|)} \\ \kappa_1 = \kappa_2 &= \frac{|r \cdot w_i - v_x|}{\max(|r \cdot w_i|, |v_x|)}, \end{aligned} \tag{7.12}$$

$$\begin{aligned} \beta_0 = \beta_3 &= |\arctan(\frac{v_y}{v_x}) - \sigma| \\ \beta_1 = \beta_2 &= \arctan(\frac{v_y}{v_x}), \end{aligned} \tag{7.13}$$

Trajectories of model can be generated by integrating the model using Runge Kutta integration method. A sample trajectory showing vehicle with steering in direction of turn with engine torque = 10nm (Figure. 7.7) and steering in opposite direction when engine torque = 100nm (Figure 7.6). Similar behavior is expected from actual vehicle which shows possible effectiveness of model.

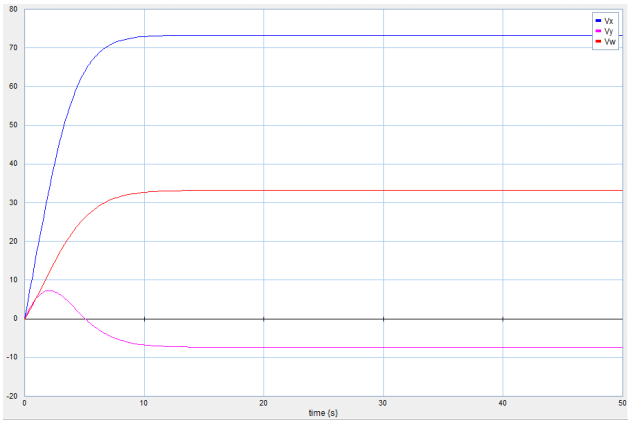

Figure 7.6: Vechile trajectory to 100nm engine torque

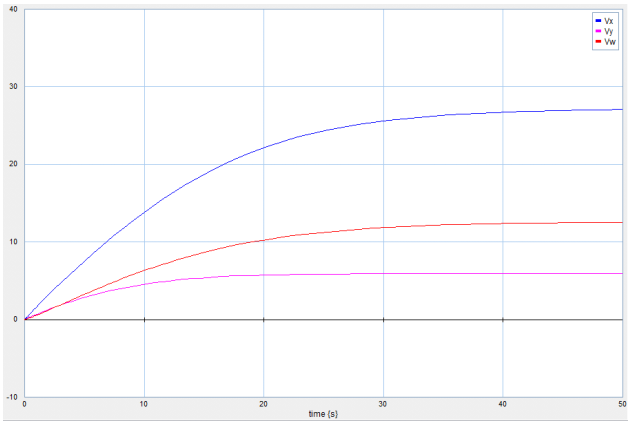

Figure 7.7: Vechile trajectory to 10nm engine torque

In another example we simulate trajectories of vehicle for a constant input with two different tire models. The response curve of Bond Graph model with a linear tire model and pacejka tire model show that they both converge to a circle which is an expected constant trajectory of a car (Figure 7.8).

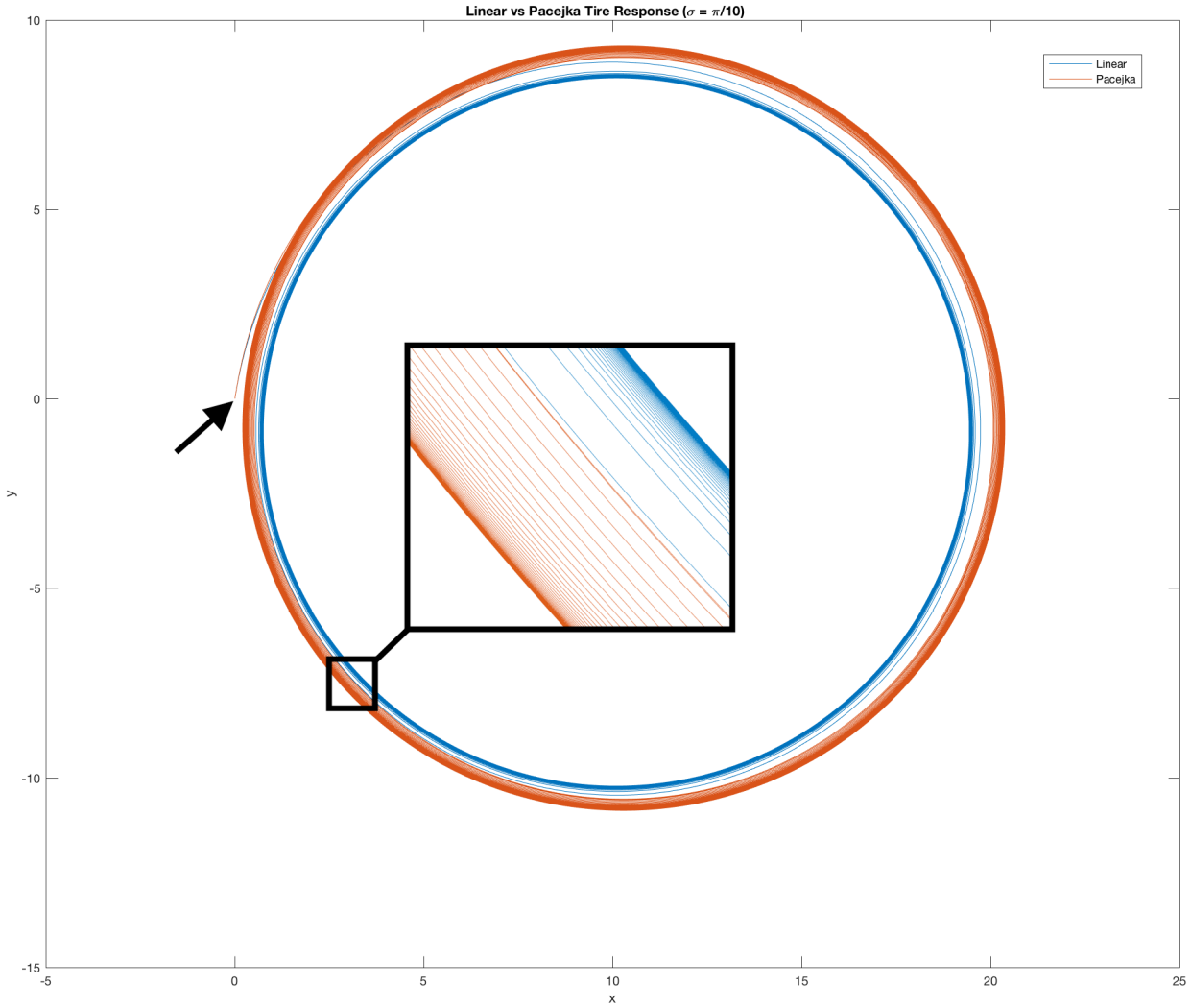

Figure 7.8: Vehicles trajectory in constant operating condition

7.1 Equilibrium Manifold

Results in Figures 7.6,7.7 and 7.8 show similarity of trajectories to an actual car but this does not verify effectiveness of model in whole range of it. To further investigate behavior of this model we extract constant trajectories of dynamically involved variables (LHS variables in system ODE) and compare it to the one in [106]. We analyze equilibrium manifold of the vehicle by studying set of trajectories of system that can be executed using constant inputs. In order to achieve this we use a numerical zero finding method to compute the equilibrium manifold on whole range of tire

operation.

Trajectory $(x(t), u(t)) \equiv (x_e, u_e)$ is a constant trajectory if it satisfies the following equation

$$f(x_e, u_e) = 0, \tag{7.14}$$

where $f(x_e, u_e)$ represents ODE of the system dynamics and x_e, u_e are system state and input in equilibrium. We solve this nonlinear system of equations using Matlab's $fsolve$ function.

Equilibrium manifold of the model is defined as a the set of constant trajectories calculated from Eq. 7.14 by iteratively solving it while perturbing one state/input and initializing the problem with the solution from last step.

By solving Eq. 7.14 slices of equilibrium manifold can are calculated. Different slices of equilibrium manifold for vehicle with linear tire model are shown in following figures.

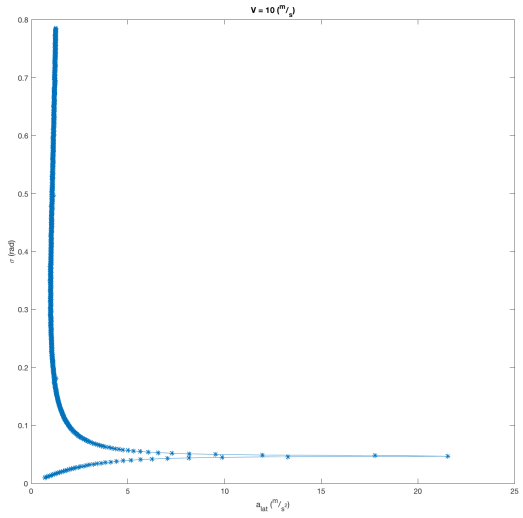

Figure 7.9: Slice of equilibrium manifold(linear tire), a_{lat} vs σ for V=10m/s

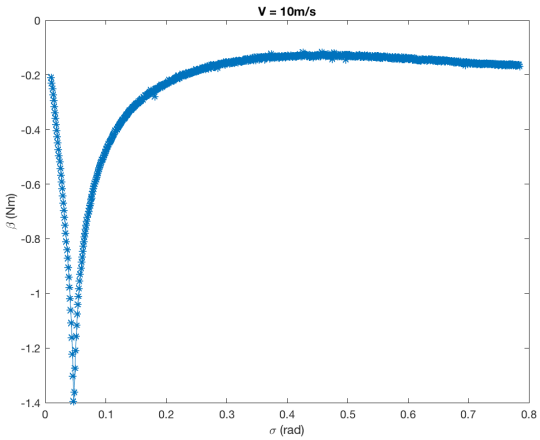

Figure 7.10: Slice of equilibrium manifold(linear tire), σ vs β for V=10m/s

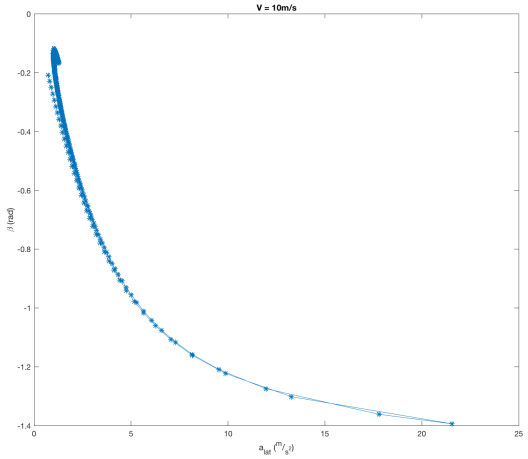

Figure 7.11: Slice of equilibrium manifold(linear tire), a_{lat} vs β for V=10m/s

Similarly, Slices of equilibrium manifold for nonlinear tire friction model which depends on side slip formulations of Eq. 7.12 and 7.13 is shown in following figures.

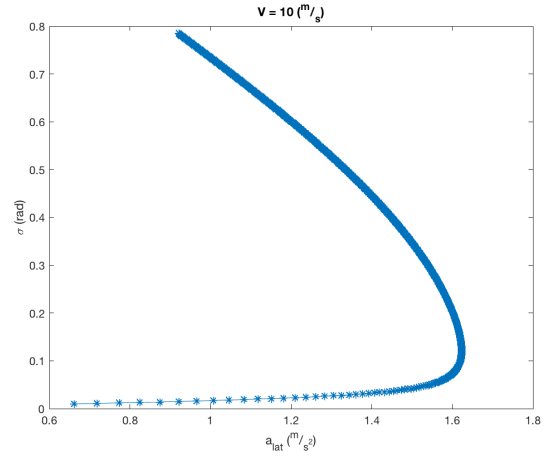

Figure 7.12: Slice of equilibrium manifold(nonlinear tire), a_{lat} vs σ for V=10m/s

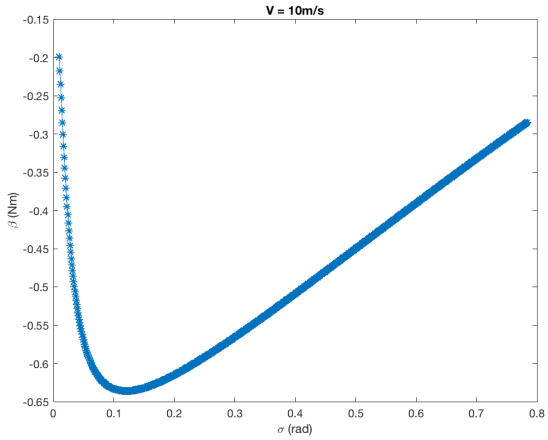

Figure 7.13: Slice of equilibrium manifold(nonlinear tire), σ vs β for V=10m/s

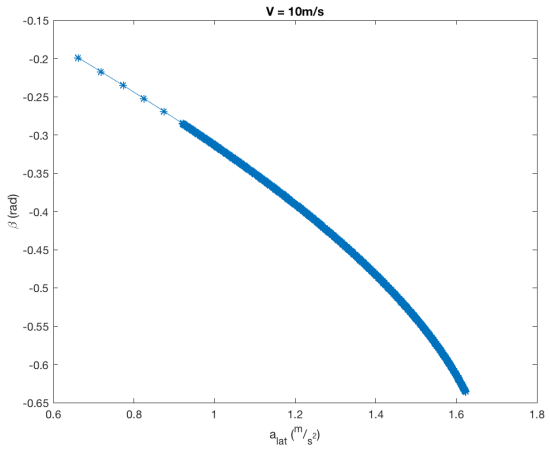

Figure 7.14: Slice of equilibrium manifold(nonlinear tire), β vs a_{lat} for V=10m/s

Chapter 8

Conclusion

8.1 Summary

Controllers designed for autonomous vehicles are required to be able to maneuver in dangerous conditions. Two classes of controllers has been designed for this purpose in the past. First, those which use simplified models of actual system to synthesize a control law and second, those which require lots of data from different conditions to learn a control law. Synthesized control laws which are based on simplified dynamics does not take benefit from capabilities of system in its limits and for second class, designing controllers for all combinations of possible changes in dynamics of the vehicle might not be possible. In this thesis we resolve the problem by removing dependency of controller and planner from model itself where they can both interact with a given model at anytime. Then, model can be updated in realtime while controller and planner would continue their task without any changes. This required design of a new local planning, controller and calibration pipeline which can interact with a separated model.

8.2 Future Work

The investigation of new control and planning method in this thesis creates number of avenues for further research:

- The controller discussed in Chapter 6 is capable of updating control action based on feedback from new sensory information. Smart tires are capable of classifying type of material

which a vehicle is driving on. This information could be used in order to update tire friction coefficient in the model.

- One of the drawbacks of using the MPC method explained in 6 is slow convergence rate for more complicated models. However, if a proper initial condition is provided, a faster convergence has been observed. Neural networks can be very capable when it comes to learning nonlinear functions and in this case they can be used in order to seed the optimization problem to a closer point to the solution.

- Slices of equilibrium manifold has been extracted for the model in Chapter 7, similar approach can be used to extract slices of equilibrium manifold for the physical vehicle and eventually compared to the calibrated model in the ODE form of Chapter 7 to verify its similarity in its whole range of performance.

Bibliography

[1] Norafizah Abas, Ari Legowo, and Rini Akmeliawati. Parameter identification of an autonomous quadrotor. In Mechatronics (ICOM), 2011 4th International Conference On, pages 1–8. IEEE, 2011.

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[3] Sina Aghli. Vehicle calibration and control pipeline.

[4] M Aicardi, G Cannata, G Casalino, and G Indiveri. On the stabilization of the unicycle model projecting a holonomic solution. In 8th Int. Symposium on Robotics with Applications, ISORA 2000, pages 11–16, 2000.

[5] Zvi Artstein. Stabilization with relaxed controls. Nonlinear Analysis: Theory, Methods & Applications, 7(11):1163–1173, 1983.

[6] MA Ataei, R Esmaeilzadeh, and Gh Alizadeh. Robust feedback linearization. In Proceedings of the 4th WSEAS International Conference on Non-linear Analysis, Non-linear Systems and Chaos, pages 114–119. World Scientific and Engineering Academy and Society (WSEAS), 2005.

[7] Hong S Bae, Jihan Ryu, and J Christian Gerdes. Road grade and vehicle parameter estimation for longitudinal control using gps. In Proceedings of the IEEE Conference on Intelligent Transportation Systems, pages 25–29, 2001.

[8] Egbert Bakker, Hans B Pacejka, and Lars Lidner. A new tire model with an application in vehicle dynamics studies. Technical report, SAE technical paper, 1989.

[9] Devin J Balkcom and Matthew T Mason. Time optimal trajectories for bounded velocity differential drive vehicles. The International Journal of Robotics Research, 21(3):199–217, 2002.

[10] Aaron Becker and Timothy Bretl. Approximate steering of a unicycle under bounded model perturbation using ensemble control. IEEE Transactions on Robotics, 28(3):580–591, 2012.

[11] Brett Bethke, Luca F Bertuccelli, and Jonathan P How. Experimental demonstration of adaptive mdp-based planning with model uncertainty. AIAA Guidance Navigation and Control, Honolulu, Hawaii, 19:23, 2008.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

[13] Jonathan Bom, Benoit Thuilot, François Marmoiton, and Philippe Martinet. A global control strategy for urban vehicles platooning relying on nonlinear decoupling laws. In Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on, pages 2875–2880. IEEE, 2005.

[14] F Borrelli, A Bemporad, and M Morari. Predictive control for linear and hybrid systems lecture notes. UC Berkley, 2014.

[15] Patrick Bouffard, Anil Aswani, and Claire Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 279–284. IEEE, 2012.

[16] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. Linear matrix inequalities in system and control theory. SIAM, 1994.

[17] Roger W Brockett, Richard-S Millman, and Hector J Sussmann. Differential geometric control theory. 1982.

[18] Carlos Canudas-de Wit, Panagiotis Tsiotras, Efstathios Velenis, Michel Basset, and Gerard Gissinger. Dynamic friction models for road/tire longitudinal interaction. Vehicle System Dynamics, 39(3):189–226, 2003.

[19] Ashwin Carvalho, Yiqi Gao, Andrew Gray, H Eric Tseng, and Francesco Borrelli. Predictive control of an autonomous ground vehicle using an iterative linearization approach. In Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on, pages 2335–2340. IEEE, 2013.

[20] Ashwin Carvalho, Yiqi Gao, Stéphanie Lefevre, and Francesco Borrelli. Stochastic predictive control of autonomous vehicles in uncertain environments. In 12th International Symposium on Advanced Vehicle Control, 2014.

[21] Erin Catto. Iterative dynamics with temporal coherence. In Game developer conference, volume 2, page 5, 2005.

[22] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path planning based on bézier curve for autonomous ground vehicles. In World Congress on Engineering and Computer Science 2008, WCECS'08. Advances in Electrical and Electronics Engineering-IAENG Special Edition of the, pages 158–166. IEEE, 2008.

[23] Dongkyoung Chwa. Tracking control of differential-drive wheeled mobile robots using a backstepping-like feedback linearization. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 40(6):1285–1295, 2010.

[24] Erwin Coumans. Bullet physics library.

[25] Erwin Coumans. Bullet physics engine. Open Source Software: http://bulletphysics. org, 1, 2010.

[26] Xiaohui Dai, Chi-Kwong Li, and Ahmad B Rad. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. IEEE Transactions on Intelligent Transportation Systems, 6(3):285–293, 2005.

[27] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. Feedback control of a nonholonomic car-like robot. Robot motion planning and control, pages 171–253, 1998.

[28] C Canudas De Wit, Hans Olsson, Karl Johan Astrom, and Pablo Lischinsky. A new model for control of systems with friction. IEEE Transactions on automatic control, 40(3):419–425, 1995.

[29] C Canudas De Wit and Panagiotis Tsiotras. Dynamic tire friction models for vehicle traction control. In Decision and Control, 1999. Proceedings of the 38th IEEE Conference on, volume 4, pages 3746–3751. IEEE, 1999.

[30] Ernst D Dickmanns and Alfred Zapp. Autonomous high speed road vehicle guidance by computer vision. In International Federation of Automatic Control. World Congress (10th). Automatic control: world congress., volume 1, 1988.

[31] Adam W Divelbiss and John T Wen. Trajectory tracking control of a car-trailer system. IEEE Transactions on Control systems technology, 5(3):269–278, 1997.

[32] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. Aggressive deep driving: Model predictive control with a cnn cost model. arXiv preprint arXiv:1707.05303, 2017.

[33] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In Robotics and Automation (ICRA), 2015 IEEE International Conference on, pages 4397–4404. IEEE, 2015.

[34] Kenny Erleben. Stable, robust, and versatile multibody dynamics animation. Unpublished Ph. D. Thesis, University of Copenhagen, Copenhagen, 2004.

[35] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. IEEE Transactions on control systems technology, 15(3):566–580, 2007.

[36] Th Fraichard and Ph Garnier. Fuzzy control to drive car-like vehicles. Robotics and autonomous systems, 34(1):1–22, 2001.

[37] R Frezza, A Beghi, and G Notarstefano. Almost kinematic reducibility of a car model with small lateral slip angle for control design. In IEEE International Symposium on Industrial Electronics, Dubrovnik, Croatia, pages 20–23, 2005.

[38] Jean Gallier. Geometric methods and applications: for computer science and engineering, volume 38. Springer Science & Business Media, 2011.

[39] Yiqi Gao, Andrew Gray, Janick V Frasch, Theresa Lin, Eric Tseng, J Karl Hedrick, and Francesco Borrelli. Spatial predictive control for agile semi-autonomous ground vehicles. In Proceedings of the 11th International Symposium on Advanced Vehicle Control, 2012.

[40] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In Asian conference on computer vision, pages 25–38. Springer, 2010.

[41] Giancarlo Genta. Motor vehicle dynamics: modeling and simulation, volume 43. World Scientific, 1997.

[42] D Gorinevsky, A Kapitanovsky, and A Goldenberg. Neural network architecture for trajectory generation and control of automated car parking. IEEE Transactions on Control Systems Technology, 4(1):50–56, 1996.

[43] Hervé Guillard and Henri Bourles. Robust feedback linearization. In Proc. 14 th International Symposium on Mathematical Theory of Networks and Systems, 2000.

[44] John L Harned, LE Johnston, and G_ Scharpf. Measurement of tire brake force characteristics as related to wheel slip (antilock) control system design. Technical report, SAE Technical Paper, 1969.

[45] Karol Hausman, James Preiss, Gaurav S Sukhatme, and Stephan Weiss. Observability-aware trajectory optimization for self-calibration with application to uavs. IEEE Robotics and Automation Letters, 2017.

[46] Ruibo He, Yingjun Zhao, Shunian Yang, and Shuzi Yang. Kinematic-parameter identification for serial-robot calibration based on poe formula. IEEE Transactions on Robotics, 26(3):411–423, 2010.

[47] CR Heckman, N Keivan, and G Sibley. Simulation-in-the-loop for planning and model-predictive control. Robotics Science and Systems (RSS) Workshop on Realistic, Rapid and Repeatable Robot Simulation, 2015.

[48] Jeong hwan Jeon, Raghvendra V Cowlagi, Steven C Peters, Sertac Karaman, Emilio Frazzoli, Panagiotis Tsiotras, and Karl Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In American Control Conference (ACC), 2013, pages 188–193. IEEE, 2013.

[49] Giovanni Indiveri, Andreas Nuchter, and Kai Lingemann. High speed differential drive mobile robot path following control with bounded wheel speed commands. In Robotics and Automation, 2007 IEEE International Conference on, pages 2202–2207. IEEE, 2007.

[50] Petros A Ioannou and Cheng-Chih Chien. Autonomous intelligent cruise control. IEEE Transactions on Vehicular technology, 42(4):657–672, 1993.

[51] Ali Jadbabaie and John Hauser. On the stability of receding horizon control with a general terminal cost. IEEE Transactions on Automatic Control, 50(5):674–678, 2005.

[52] D Peter Joseph and T Julius Tou. On linear control theory. Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry, 80(4):193–196, 1961.

[53] Mike Kamermans. A primer on bzier curves. https://pomax.github.io/bezierinfo/.

[54] Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, and Tetsuo Noguchi. A stable tracking control method for a non-holonomic mobile robot. In Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on, pages 1236–1241. IEEE, 1991.

[55] Dean C Karnopp, Donald L Margolis, and Ronald C Rosenberg. System dynamics: modeling, simulation, and control of mechatronic systems. John Wiley & Sons, 2012.

[56] Nima Keivan and Gabe Sibley. Constant-time monocular self-calibration. In Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on, pages 1590–1595. IEEE, 2014.

[57] Nima Keivan and Gabe Sibley. Realtime simulation-in-the-loop control for agile ground vehicles. In Ashutosh Natraj, Stephen Cameron, Chris Melhuish, and Mark Witkowski, editors, Towards Autonomous Robotic Systems, volume 8069 of Lecture Notes in Computer Science, pages 276–287. Springer Berlin Heidelberg, 2014.

[58] Nima Keivan and Gabe Sibley. Asynchronous adaptive conditioning for visual–inertial slam. The International Journal of Robotics Research, 34(13):1573–1589, 2015.

[59] Nima Keivan and Gabe Sibley. Online slam with any-time self-calibration and automatic change detection. In International Conference on Robotics and Automation (ICRA), 2015.

[60] Hassan K Khalil. Noninear systems. Prentice-Hall, New Jersey, 2(5):5–1, 1996.

[61] A Khan, Darek Ceglarek, and Jun Ni. Sensor location optimization for fault diagnosis in multi-fixture assembly systems. Journal of manufacturing science and engineering, 120(4):781–792, 1998.

[62] KRS Kodagoda, W Sardha Wijesoma, and Eam Khwang Teoh. Fuzzy speed and steering control of an agv. IEEE Transactions on control systems technology, 10(1):112–120, 2002.

[63] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In Intelligent Vehicles Symposium (IV), 2015 IEEE, pages 1094–1099. IEEE, 2015.

[64] Arthur J Krener. Adaptive horizon model predictive control. arXiv preprint arXiv:1602.08619, 2016.

[65] Krisada Kritayakirana and J Christian Gerdes. Autonomous vehicle control at the limits of handling. International Journal of Vehicle Autonomous Systems, 10(4):271–296, 2012.

[66] Steven M. LaValle. Planning algorithms, 2006.

[67] Keun Uk Lee, Young Hun Yun, Wook Chang, Jin Bae Park, and Yoon Ho Choi. Modeling and altitude control of quad-rotor uav. In Control, Automation and Systems (ICCAS), 2011 11th International Conference on, pages 1897–1902. IEEE, 2011.

[68] Kooktae Lee, Dalhyung Kim, Woojin Chung, Hyo Whan Chang, and Paljoo Yoon. Car parking control using a trajectory tracking controller. In SICE-ICASE, 2006. International Joint Conference, pages 2058–2063. IEEE, 2006.

[69] Ti-Chung Lee, Kai-Tai Song, Ching-Hung Lee, and Ching-Cheng Teng. Tracking control of unicycle-modeled mobile robots using a saturation feedback controller. IEEE transactions on control systems technology, 9(2):305–318, 2001.

[70] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[71] C-T Lin and C. S. George Lee. Neural-network-based fuzzy logic control and decision system. IEEE Transactions on computers, 40(12):1320–1336, 1991.

[72] Chang Boon Low and Danwei Wang. Gps-based path following control for a car-like wheeled mobile robot with skidding and slipping. IEEE Transactions on control systems technology, 16(2):340–347, 2008.

[73] Yi Ma, Jana Kosecka, and S Shankar Sastry. Vision guided navigation for a nonholonomic mobile robot. IEEE Transactions on robotics and automation, 15(3):521–536, 1999.

[74] Markus Maurer, Reinhold Behringer, Dirk Dickmanns, Thomas Hildebrandt, Frank Thomanek, Joachim Schiehlen, and Ernst Dieter Dickmanns. Vamors-p: An advanced platform for visual autonomous road vehicle guidance. In Mobile Robots IX, volume 2352, pages 239–249. International Society for Optics and Photonics, 1995.

[75] Jérôme Maye, Hannes Sommer, Gabriel Agamennoni, Roland Siegwart, and Paul Furgale. Online self-calibration for robotic systems. The International Journal of Robotics Research, 35(4):357–380, 2016.

[76] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. Automatica, 36(6):789–814, 2000.

[77] Daniel Mellinger, Quentin Lindsey, Michael Shomin, and Vijay Kumar. Design, modeling, estimation and control for aerial grasping and manipulation. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pages 2668–2673. IEEE, 2011.

[78] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In Proceedings of the 22nd international conference on Machine learning, pages 593–600. ACM, 2005.

[79] William F Milliken, Douglas L Milliken, et al. Race car vehicle dynamics, volume 400. Society of Automotive Engineers Warrendale, 1995.

[80] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[81] Seungwuk Moon and Kyongsu Yi. Human driving data-based design of a vehicle adaptive cruise control algorithm. Vehicle System Dynamics, 46(8):661–690, 2008.

[82] Frank Moosmann and Christoph Stiller. Velodyne slam. In Intelligent Vehicles Symposium (IV), 2011 IEEE, pages 393–398. IEEE, 2011.

[83] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In Numerical analysis, pages 105–116. Springer, 1978.

[84] Bryan Nagy and Alonzo Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. Field and Service Robots, 11, 2001.

[85] Jose E Naranjo, Carlos Gonzalez, Ricardo Garcia, and Teresa De Pedro. Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver. IEEE Transactions on Intelligent Transportation Systems, 9(3):438–450, 2008.

[86] Fernando Nobre, Michael Kasper, and Christoffer Heckman. Drift-correcting self-calibration for visual-inertial slam. In Robotics and Automation (ICRA), 2017 IEEE International Conference on, pages 6525–6532. IEEE, 2017.

[87] Julio E Normey-Rico, Ismael Alcalá, Juan Gómez-Ortega, and Eduardo F Camacho. Mobile robot path tracking using a robust pid controller. Control Engineering Practice, 9(11):1209–1214, 2001.

[88] Se-Young Oh, Jeong-Hoon Lee, and Doo-Hyun Choi. A new reinforcement learning vehicle control architecture for vision-based road following. IEEE Transactions on Vehicular Technology, 49(3):997–1005, 2000.

[89] Hans Pacejka. Tire and vehicle dynamics. Elsevier, 2005.

[90] Hans B Pacejka and Egbert Bakker. The magic formula tyre model. Vehicle system dynamics, 21(S1):1–18, 1992.

[91] Sharon L Padula and Rex K Kincaid. Optimization strategies for sensor and actuator placement. 1999.

[92] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. arXiv preprint arXiv:1709.07174, 2017.

[93] Witold Pedrycz. Fuzzy control and fuzzy systems (2nd. Research Studies Press Ltd., 1993.

[94] Romain Pepy, Alain Lambert, and Hugues Mounier. Path planning using a dynamic vehicle model. In Information and Communication Technologies, 2006. ICTTA'06. 2nd, volume 1, pages 781–786. IEEE, 2006.

[95] Rajesh Rajamani. Vehicle dynamics and control. Springer Science & Business Media, 2011.

[96] Pongsathorn Raksincharoensak, Masao Nagai, and Motoki Shino. Lane keeping control strategy with direct yaw moment control input by considering dynamics of electric vehicle. Vehicle System Dynamics, 44(sup1):192–201, 2006.

[97] Carl Edward Rasmussen, Malte Kuss, et al. Gaussian processes in reinforcement learning. In NIPS, volume 4, page 1, 2003.

[98] Hadi Ravanbakhsh, Sina Aghli, Christoffer Heckman, and Sriram Sankaranarayanan. Path-following through control funnel functions. arXiv preprint arXiv:1804.05288, 2018.

[99] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Learning lyapunov (potential) functions from counterexamples and demonstrations. arXiv preprint arXiv:1705.09619, 2017.

[100] J. B. Rawlings and D. Q. Mayne. Model predictive control: Theory and design. Nob Hill Pub., 2009.

[101] Philipp Reist, Pascal Preiswerk, and Russ Tedrake. Feedback-motion-planning with simulation-based lqr-trees. The International Journal of Robotics Research, 35(11):1393–1416, 2016.

[102] Philipp Reist and Russ Tedrake. Simulation-based lqr-trees with input and state constraints. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 5504–5510. IEEE, 2010.

[103] J-M Renders, Eric Rossignol, Marc Becquet, and Raymond Hanus. Kinematic calibration and geometrical parameter identification for robots. IEEE Transactions on robotics and automation, 7(6):721–732, 1991.

[104] Martin Riedmiller, Mike Montemerlo, and Hendrik Dahlkamp. Learning to drive a real car in 20 minutes. In Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007, pages 645–650. IEEE, 2007.

[105] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. On a reduced-order two-track car model including longitudinal and lateral load transfer. In Control Conference (ECC), 2013 European, pages 980–985. IEEE, 2013.

[106] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. Optimal control based dynamics exploration of a rigid car with longitudinal load transfer. IEEE Transactions on Control Systems Technology, 22(3):1070–1077, 2014.

[107] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. An efficient minimum-time trajectory generation strategy for two-track car vehicles. IEEE Transactions on Control Systems Technology, 23(4):1505–1519, 2015.

[108] Jihan Ryu, Eric J Rossetter, and J Christian Gerdes. Vehicle sideslip and roll parameter estimation using gps. In Proceedings of the AVEC International Symposium on Advanced Vehicle Control, 2002.

[109] Claude Samson. Control of chained systems application to path following and time-varying point-stabilization of mobile robots. IEEE transactions on Automatic Control, 40(1):64–77, 1995.

[110] K Schilling and C Jungius. Mobile robots for planetary exploration. Control Engineering Practice, 4(4):513–524, 1996.

[111] Thomas Schneider, Mingyang Li, Michael Burri, Juan Nieto, Roland Siegwart, and Igor Gilitschenski. Visual-inertial self-calibration on informative motion segments. In Robotics and Automation (ICRA), 2017 IEEE International Conference on, pages 6487–6494. IEEE, 2017.

[112] Thomas W Sederberg. Computer aided geometric design. 2012.

[113] Masayasu Shimakage, Shigeki Satoh, Kenya Uenuma, and Hiroshi Mouri. Design of lane-keeping control with steering torque input. JSAE review, 23(3):317–323, 2002.

[114] C Sierra, E Tseng, A Jain, and H Peng. Cornering stiffness estimation based on vehicle lateral dynamics. Vehicle System Dynamics, 44(sup1):24–38, 2006.

[115] Zachary Johann Asmussen Sina Aghli. Parkour car ecu design files.

[116] Jakub Stkepien. Physics-based animation of articulated rigid body systems for virtual environments.

[117] Russ Tedrake. Lqr-trees: Feedback motion planning on sparse randomized trees. 2009.

[118] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. The International Journal of Robotics Research, 29(8):1038–1052, 2010.

[119] The New York Times. Robot set to explore a volcano that's too dangerous for humans to enter, 1994.

[120] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 5026–5033. IEEE, 2012.

[121] Veeravalli S Varadarajan. Lie groups, Lie algebras, and their representations, volume 102. Springer Science & Business Media, 2013.

[122] Gentiane Venture, P-J Ripert, Wisama Khalil, Maxime Gautier, and Philippe Bodson. Modeling and identification of passenger car dynamics using robotics formalism. IEEE Transactions on Intelligent Transportation Systems, 7(3):349–359, 2006.

[123] Mathukuma Vidyasagar. Nonlinear systems analysis. Prentice Hall, 1978.

[124] Goran S Vorotović, Branislav B Rakicević, Saša R Mitić, and Dragan D Stamenković. Determination of cornering stiffness through integration of a mathematical model and real vehicle exploitation parameters. FME Transactions, 41(1):66–71, 2013.

[125] Junmin Wang, Lee Alexander, and Rajesh Rajamani. Friction estimation on highway vehicles using longitudinal measurements. Journal of dynamic systems, measurement, and control, 126(2):265–275, 2004.

[126] Wikipedia. Automated guided vehicle — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-November-2017].

[127] Wikipedia. History of autonomous cars — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-November-2017].

[128] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. arXiv preprint arXiv:1509.01149, 2015.

[129] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information theoretic model predictive control: Theory and applications to autonomous driving. arXiv preprint arXiv:1707.02342, 2017.

[130] Grady Williams, Brian Goldfain, Paul Drews, James M Rehg, and Evangelos A Theodorou. Autonomous racing with autorally vehicles and differential games. arXiv preprint arXiv:1707.04540, 2017.

[131] Grady Williams, Eric Rombokas, and Tom Daniel. Gpu based path integral control with learned dynamics. arXiv preprint arXiv:1503.00330, 2015.

[132] Stephen J Wright. Numerical optimization.