STABLE MODEL PREDICTIVE PATH INTEGRAL CONTROL FOR
AGGRESSIVE AUTONOMOUS DRIVING

A Thesis
Presented to
The Academic Faculty

by

Matthieu J. Capuano

In Partial Fulfillment
of the Requirements for the Degree
Bachelors of Science in the
School of Computer Science

Georgia Institute of Technology
April 2019
STABLE MODEL PREDICTIVE PATH INTEGRAL CONTROL FOR
AGGRESSIVE AUTONOMOUS DRIVING

Approved by:

Dr. Evangelos Theodorou, Advisor
School of Aerospace Engineering
*Georgia Institute of Technology*

Dr. James Rehg
School of Computational Science and Engineering
*Georgia Institute of Technology*

Date Approved: 13th of December, 2019
To Evangeline and Marie-Bénédicte Capuano.
ACKNOWLEDGEMENTS

I wish to thank Grady Williams, Paul Drews, and Brian Goldfain for spending the past several years conducting the foundational research on which this thesis is based and for always being available for help and guidance. I particularly wish to thank Dr. Williams for having been my direct mentor and research guide for the past year and a half, without which this work would not have taken place. I also want to thank Dr. Evangelos Theodorou, Associate Professor in the school of Aerospace Engineering, and my research advisor, for his academic and research guidance.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACKNOWLEDGEMENTS</strong></td>
<td>v</td>
</tr>
<tr>
<td><strong>LIST OF TABLES</strong></td>
<td>vii</td>
</tr>
<tr>
<td><strong>LIST OF FIGURES</strong></td>
<td>viii</td>
</tr>
<tr>
<td><strong>LIST OF SYMBOLS AND ABBREVIATIONS</strong></td>
<td>ix</td>
</tr>
<tr>
<td><strong>SUMMARY</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>1  <strong>INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>2  <strong>LITERATURE REVIEW</strong></td>
<td>5</td>
</tr>
<tr>
<td>3  <strong>MODEL PREDICTIVE PATH INTEGRAL CONTROL &amp; STABILITY</strong></td>
<td>11</td>
</tr>
<tr>
<td>Model Predictive Path Integral Control</td>
<td>11</td>
</tr>
<tr>
<td>Stability Problems</td>
<td>13</td>
</tr>
<tr>
<td>4  <strong>TUBE-MPPI</strong></td>
<td>15</td>
</tr>
<tr>
<td>5  <strong>EXPERIMENTS</strong></td>
<td>17</td>
</tr>
<tr>
<td>Disclaimer</td>
<td>17</td>
</tr>
<tr>
<td>Experimental Results</td>
<td>17</td>
</tr>
<tr>
<td>PID Controller</td>
<td>21</td>
</tr>
<tr>
<td>6  <strong>CONCLUSIONS AND FUTURE WORKS</strong></td>
<td>24</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>26</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1: [7] Results on the simulated driving task 20
Table 2: PID performance on simulated driving task 22
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Expected state sequence given a fixed control sequence</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Adding Gaussian noise to control sequence to sample trajectories</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Perturbed system state</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Baseline-MPPI performance on the point-mass task</td>
<td>18</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Disturbance-MPPI performance on the point-mass task</td>
<td>19</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Tube-MPPI performance on the point-mass task</td>
<td>19</td>
</tr>
</tbody>
</table>
## LIST OF SYMBOLS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>State disturbance</td>
</tr>
<tr>
<td>$F$</td>
<td>Discrete-time state propagation function</td>
</tr>
<tr>
<td>$Q^r$</td>
<td>Optimal sampling distribution</td>
</tr>
<tr>
<td>$Q$</td>
<td>Actual open-loop sampling distribution</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Instantaneous state cost function</td>
</tr>
<tr>
<td>$u_t$</td>
<td>Commanded control value at time step $t$</td>
</tr>
<tr>
<td>$v_t$</td>
<td>Actual control value at time step $t$</td>
</tr>
<tr>
<td>$w$</td>
<td>Weighting factor</td>
</tr>
<tr>
<td>$x_t$</td>
<td>System state at time step $t$</td>
</tr>
</tbody>
</table>
SUMMARY

This thesis discusses the development of a stability improving modification to Model Predictive Path Integral (MPPI) Control, a model predictive control algorithm that can optimize for non-linear systems in stochastic environments, subject to complex cost criteria. The method is highly effective at controlling systems close to their limits of handling, such as a self-driving race car pushed to its speed limits. More specifically, this work details the original formulation of MPPI, the stability challenges it faced in response to drastic state changes due to strongly stochastic environments, and the resulting framework developed to address this challenge. Note that the contributions of this work, and thus of this author, are limited to the development of a PID controller within the resulting framework to perform a tracking task. The overall framework itself is the product of research conducted by the authors of papers referenced throughout, the key of which are named in the acknowledgments. The final method proposed differs from the original MPPI framework in that, as opposed to previous iterations of the algorithm which did not theoretically account for stochasticity in the environment, the new framework does so by separately controlling a nominal system and a separate, more realistic, system. By acknowledging the disturbed system and how it deviates from the nominal one, we are able to add a controller with the specific task of tracking the actual system so that it stays within a certain bound of the nominal system. We propose the experimental verification of this method, dubbed Tube-MPPI by the authors of [7], on Auto-Rally, a one-fifth scale autonomous vehicle for aggressive self-driving tasks.
CHAPTER 1

INTRODUCTION

Modern robotic systems are increasingly being developed to function in rapidly changing, nonlinear environments while asked to perform safety critical tasks that require complex decision-making processes. To be as effective as possible in their motion planning and execution, these systems need to make full use of their dynamic abilities, while still accomplishing the given objective. Dynamic autonomous vehicle motion at the limits of handling is a highly active area of research in robotics for systems that aim to accomplish a goal as rapidly as possible.

In theory, a stochastic optimal control framework can generate very high performance with reliable, and diverse behaviors for nonlinear systems, but requires the specification of a cost or reward function. Trajectories or a control plan are generated by continuously minimizing this cost function with respect to the system dynamics of the vehicle [2]. This is often used in Model Predictive Control as the minimization and motion planning can be done in parallel [9].

This approach is highly successful and has seen many applications but specifying the cost/reward function is challenging and tedious because the system dynamics of the vehicle constrain the solutions to the optimal control problem. So writing such cost functions becomes more about having few local minima than about having an easily programmable formula. It is also often impossible to test these functions in all possible environments that the system might operate in, particularly for highly complex platforms with many degrees of freedom.
Rather than minimizing a cost function through gradient-based approaches, one alternative is to use gradient-free sampling methods [23, 24]. In such methods, a simple encoding of a specific task within a given environment can be created. Instead of minimizing a cost function, the encoding is used to sample from and generate control value sequences. Multiple such encodings can be created to accomplish multiple similar tasks that don’t interfere with each other. This alternative works well with Model Predictive Control (MPC), but the lack of information in the encoding function makes the MPC control planning unstable and unable to recover upon unexpected disturbances. This is because the sampling often follows an importance sampling scheme that is based on the previous sequence, which does not account for stochastic state disturbances. It is also somewhat fragile in nonlinear dynamics [7].

The overarching reason for these difficulties is that sampling methods are local search approaches, and that it is impossible to sample across entire state spaces, particularly if they are high dimensional. Consequently, these methods are generally initialized with the parameters that the framework hopes to sample. In MPC, the initialization generally consists of a sequence of control values that could carry the system forward in time without difficulty or harm. It is important that the initialization sequence is reliable. In subsequent iterations, future sampling is based around that sequence in the search of an optimal series of controls.

Again, the issue is that this process does not account for sudden changes in states that are misaligned with expectations. The sampling will continue to occur based on the previous sequence regardless of whether the state has dramatically changed or not and, without a reliable gradient, the system will never be pushed back towards a low-cost
region. Essentially, the system can get caught in local minima whenever it is pushed off course or heading. Mathematically, we consider the general discrete time, non-linear system:

$$x_{t+1} = F(x_t, u_t + \epsilon_t) + w_t$$

Where $x \in \mathbb{R}^N$ is the state, $u \in \mathbb{R}^M$ is the control input, $\epsilon$ is noise on the control, and $w$ is noise on the system state. The issue lies in the $w_t$ term, when the neural net that models the system dynamics is trained, it is not given highly stochastic behaviors at data points to train on, and it should not be as the its purpose is to accurately predict behavior given a specific set of inputs. Essentially, the planned control sequence at time $t$: $(u_t, u_{t+1}, \ldots, u_{t+T})$ is assumed to be near-optimal for conditions drawn from:

$$\tilde{x} \sim F(x_t, u_t + \epsilon)$$

But the actual system state is drawn from:

$$\tilde{x} \sim F(x_t, u_t + \epsilon) + w$$

If the perturbation is large, then the actual state can be very different from the previous one, making the near-optimality assumption valid and the importance sampling scheme problematic.

A recently proposed solution to this is to use the nonlinear Tube-MPC framework with two model predictive controllers [7]. The first, named nominal controller, “solves the primary optimal control problem for an idealized nominal state” (Grady et. al, 2018) while the second, the ancillary controller aims to reject disturbances to keep the system close to the nominal state. The approach was successful at stabilizing the importance sampling distribution, thus helping the system recover from large disturbances. However, the work has not yet provided theoretical guarantees for some aspects of Tube-MPC, such
as how to guarantee that bounds for the tube exist, and how to incorporate them into the optimal control problem solved by the nominal controller. Again, the contribution of this work is the development of a PID controller for the tracking task.

In this proposal, we investigate two contributions: (1) A potential improvement to this Tube-MPC framework and (2) a theoretical alternative for stabilizing the importance sampling distribution. The current ancillary controller in Tube-MPPI is an iLQG that solves the actual system state within a tube centered about the nominal state. It provides good performance at a small computational cost, but its performance was not evaluated within Tube-MPPI against other possible ancillary controllers. We thus begin by investigating a different ancillary controller and comparing differences in performance. The new ancillary controller is a double PID – one for throttle and one for steering in the case of the AutoRally platform, on which we conduct our physical experiments.

Second, we consider an alternative to Tube-MPPI altogether whereby we use the same double PID controller as a preliminary processor ahead of MPPI. That is, using a set of PID controllers for each control value, a neural network for state propagation, and an approximately optimal path that we wish to follow, we generate a set of control values to take the system around this optimal path. This is done by computing the control values that drive the system to the next nominal waypoint, projecting the state forward, and repeating the process. These control sequences can then be used as inputs to MPPI as the new importance sampling baseline, this is on contrast to using the optimal sequence at the previous iteration, as was done in [1]-[3].
CHAPTER 2
LITERATURE REVIEW

Path planning and tracking are fundamental functions of any autonomous system that needs to move in physical space to execute a given set of goals. That is, an autonomous system, such as a bipedal robot, given a task that requires moving in 2D or 3D space will need to perform some type of planning for how it will move in this space, as well as planning for how it will control its actuators to execute that motion plan, and then track its progression according to it. Alternatively, it will need to use online learning techniques with real-time decision making. Robotic systems are increasingly being asked to perform tasks as quickly and efficiently as possible, but most control and learning based frameworks for motion planning are not designed for highly dynamic, “aggressive” autonomous maneuvers, so the field of controls and motion planning at the limits of what a specific system is capable of handling has been growing in importance.

The field of autonomous driving specifically is relatively new, with one of the most notable milestones being the successful completion of the DARPA challenge in 2006 by a Stanford research team directed by Sebastian Thrun [1]. Though there has been an enormous and continuously accelerating rate in the amount of literature in this field since, control at the limits of handling for dynamic systems in general, with the vast majority of applications being in self driving cars, has only taken off over the past few years. Yet it is an incredibly important area of research if autonomous vehicles are ever to be deployed, as they could greatly outperform human ability to handle a seemingly out-of-control system at such limits [3]. For instance, a fast-driving car suddenly skidding on icy road may be too difficult for an average driver to handle, but a control system
designed to handle such situations could act appropriately to make the vehicle act as close as possible to a desired behavior, such as minimizing spin, navigating a coming turn to avoid a collision, or decelerating as fast but safely as possible for passengers. Several methods have been developed to handle dynamic systems at the limits of handling though, as [2] points out, most of them suffer from imperfect stability margins or excessive sensitiveness to vehicle model uncertainty.

One key factor that defines what it means to be driving at the limits of handling for autonomous vehicles is the friction on tires and its upper bounds, which impact the tire force that can be produced by the vehicle. One attempt for control of autonomous cars at the limits of friction was implemented in [3] and involves using a dual controller, one for speed-tracking and one for steering control along a desired path, as well as in [4] by using then recent improvements in steering-torque measurements, as well as vehicle side-slip and tire-road friction estimates, in a Model Predictive Control framework. Both of these methods were relatively successful but had the requirement that friction on the tires needed to be known throughout tracking. Stability of execution was acceptable, but within a provided envelope, and the vehicle used some guidance from a human driver, rather than independent autonomy. The limitations in their results were in part due to their use of a tire-road friction coefficient that was computed a priori, instead of continuously estimated and updated in real time. Source [5] developed an algorithm to estimate tire-road friction (and tire cornering stiffness) in real-time which was used by Laurense et al. in [6] to determine how accurately friction coefficient needs to be known for efficient path tracking at the limits of friction. They experimentally determined that tire-road friction needs to be known within about 2% for an autonomous vehicle that uses
a dual controller system as in [3] and [4]. This requirement is more than what real-time friction estimation algorithms can reliably provide so, instead, they proposed a slip angle-based steering controller that maintained the front tires at the slip angle that maximized tire force, and a second controller for path-tracking of the speed. Their controller has less strict requirements and is promising but has not yet had its performance compared to other path-tracking control schemes when friction is under or over expanded, nor has it been applied to tracks with varying friction coefficients, or ones where friction is not the limiting factor. It has also shown some susceptibility to inaccuracies in the dynamics model used. Dr. Gerdes’ lab is actively building on this research in the hopes of providing a robust and proven optimal controller for aggressive driving maneuvers.

A very separate approach from these methods, which does not suffer from sensitiveness to vehicle model uncertainty, has been the development of Model Predictive Path Integral (MPPI) Control. The previously discussed methods generally use a hierarchal control framework, whereby they plan a trajectory through knowledge of the position and velocity of the vehicle, and then use a feedback controller to track the trajectory. This works very well if the problem specifications are within the limits of what the vehicle can handle, or if the velocity and path profile are defined in the problem structure but, as just discussed, generating feasible path requires precise knowledge of the system dynamics and coefficient of friction. The MPPI approach combines planning and execution by accounting for the dynamics during optimization. This framework is derived from a stochastic optimal control foundation, which presents the challenge that solving the optimal control problem for nonlinear systems is computationally expensive. One solution is to solve for the optimal control inputs offline, which removes real-time
constraints on the system. This is employed in [7], which solves for an open loop control sequence offline, and in [8], which solves for aggressive collision-avoiding, sliding maneuvers and generates optimal trajectories offline with various initial conditions, through general MPC methods. Note that the MPPI sampling and execution method follows the procedure of an MPC algorithm. Unlike [7] and [8], however, the MPPI approach generates new trajectories in real-time by taking advantage of recent developments in GPU programming. The approach is a stochastic trajectory optimization framework and is based on path integral control. Path integral control [9] is useful in obtaining optimal control algorithms based on stochastic sampling of trajectories, without need for derivatives of the dynamics or cost functions. That is, it allows the derivation of algorithms that can sample stochastic trajectories for given systems. MPPI was originally derived using such an approach [10], but was restricted to systems with control affine dynamics.

In [11] the method was extended to a larger class of stochastic problems and representations. Specifically, they showed how MPPI’s update law could be derived through an information theoretic framework without the control affine assumption. Other key concepts in the derivation were the information theoretic principles of free energy and KL divergence. [11] showed that the controller, given a neural network trained to approximate the system dynamics of several vehicles, was able to reliably take a quadcopter simulation around a set of obstacles, or solve a cart-pole swing up simulation. In particular, they tested their approach on a one fifth scale rally car, and showed that their controller could reliably take it around a 30-meter-long track at over 8m/s. An
interesting success of this approach was the vehicle’s ability to perform power slides around turns and drift in a controlled manner at the limits of friction.

A simple explanation of how the MPPI framework functions is to think of it as a Model Predictive Control framework that samples multiple control trajectories over a given time horizon, and projects it’s system in space based on those control values using a neural network. It thus never produces infeasible sequences, since they are generated using a system dynamics model trained from real driving. By averaging all the control sequences, an optimal sequence is then obtained. The first step is executed before the process is repeated. As most sampling based (and MPC) frameworks do, it then re-uses the previously optimized control sequence as a Gaussian distribution for importance sampling of the next sequences. This makes the method more capable of being run on-line but, as was the case with some of the approaches in [1]-[6], reduces robustness under large disturbances. In this case, this is because the planned control sequence \((u_1, u_2, \ldots, u_T)\) is assumed near optimal at the current time step, and is re-used to base a Gaussian distribution at the next time step, but at any given point a disturbance may have dramatically changed the system state [11]. Thus, the sequence will produce behaviors vastly different from original expectations.

In [12], this issue is addressed using a Tube-MPC framework. Two controllers, a nominal controller and an ancillary controller, are used to create bounds for an autonomous car and to ensure that the vehicle remains within those constraints. The nominal controller is a non-linear model predictive controller that computes solutions to the path planning problem (providing solutions in the form of a policy with controls for expected states) while ignoring system disturbances. The ancillary controller then tracks
the nominal system state and ensures that the actual system state stays within a tube centered about the nominal state. When the nominal controller used was MPPI, and the ancillary controller was an iLQG, [12] showed that Tube-MPPI outperforms a highly tuned regular MPPI controller in top speed. More importantly, it provided a procedure that stabilizes the importance sampling distribution, meaning that a gradient was not necessary in recovering from large disturbances, and allowing the use of simple cost functions.

The purpose of the current project is twofold. The first goal is to evaluate a potential improvement in the Tube-MPPI framework by replacing the iLQG with a double PID controller for tracking of nominal steering and throttle states. That is, at every time step the steering and throttle control inputs for ensuring that the system remains within the bounding Tube would be computed using a PID for both values. This should improve the system’s ability to perform the desired goal (e.g. driving as rapidly as possible around a track) by providing an ancillary controller better capable of enforcing the boundary constraints, while maintaining the advantage of a normalized importance sampling distribution. Thus allowing the use of simple cost functions like weighted sums of indicator functions. Second, it has been suggested that replacing the system model from one that simply accounts for the system dynamics to one that also accounts for a simple tracking controller (in this case the double PID) would also improve speed and optimality of recovery under strong disturbances. This can be both combined with the Tube-MPPI framework and used independently with the original MPPI algorithm for comparative purposes in evaluating performance.
CHAPTER 3

MODEL PREDICTIVE PATH INTEGRAL CONTROL AND STABILITY

A. Model Predictive Path Integral Control

We now more precisely detail the functioning of MPPI as formulated in [2], as well as the stability challenge addressed, in preparation for a discussion of the proposed solution, Tube-MPPI. Model Predictive Path Integral Control operates as any MPC framework does: it begins with a sequence of control values over a pre-determined time horizon, executes the first value in that sequence, shifts its timestep by one, and re-computes the control sequence at the next time step, in this case using the sequence at the previous timestep as a sampling reference. More specifically, the information theoretic derivation of MPPI presented in [2] establishes the following uncontrolled system distribution \( p(v) \), and open loop distribution, \( q(v) \):

\[
p(v) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2} v_t^T \Sigma^{-1} v_t \right)
\]

\[
q(v) = \prod_{t=0}^{T-1} Z^{-1} \exp\left(-\frac{1}{2} (v_t - u_t)^T \Sigma^{-1} (v_t - u_t) \right)
\]

\[
Z = ((2\pi)^m |\Sigma|)^{\frac{1}{2}}
\]

Where \( u_t \) is the commanded input, which is disturbed by some controller noise such that the actual input is \( v_t \overset{\sim}{\sim} N(u_t, \Sigma) \). By using the information theoretic concept of free energy to define a lower bound on the general cost function:

\[
C(x_1, x_2, ..., x_T) = \phi(x_T) + \sum_{t=1}^{T-1} q(x_t)
\]
They are then able to show that there exists an optimal distribution from which controls could be sampled:

\[ q^*(V) = \frac{1}{\eta} \exp \left( -\frac{1}{\lambda} S(V) \right) p(V) \]

Where, they compute:

\[ p(V) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp \left( \sum_{t=0}^{T-1} \Sigma^{-1} v_t^T \Sigma^{-1} v_t \right) \]

\[ S(V) = \phi(x_T) + \Sigma_{t=0}^{T-1} \left( k(x)^T Q k(x) + \Sigma_{i=1}^{N} w_i \epsilon_i(x) \right) \]

Note that the term within the summation is a predefined cost function for the system. The goal then becomes the minimization of the KL Divergence beyond the open loop distribution and the optimal distribution:

\[ U^* = \arg \min_U \mathcal{D}_{KL}(Q^* || Q) \]

\[ = \int q^*(V) V dV \]

This equation is impossible to compute directly but can be approximated over time with the iterative importance sampling scheme:

\[ U_{k+1} = U_k + \sum_{i=1}^{N} w(\epsilon_i) \epsilon_i \]

\[ w(\epsilon_i) = \frac{1}{\eta} \exp \left( -\frac{1}{\lambda} \left( S(U_k + \epsilon_i) + \lambda \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} u_t \right) \right) \]

Programmatically, this takes the form of Algorithm 2 in [2], which works as follows: beginning with a warm start of control values, we choose to sample a predetermined \( K \) number of paths over a time horizon \( T \) by adding random gaussian to the past control.
values at every timestep. Visually, if our predicted path for the control sequence at time $t$ is the following:

![Figure 1. Expected state sequence given a fixed control sequence](image)

One of the $K$ paths generated by adding random gaussian noise to our control values and projecting the state forward in time might be:

![Figure 2. Adding Gaussian noise to control sequence to sample trajectories](image)

Note that, in practice, the paths are not as jerky as illustrated since the standard deviation and time steps are both small. We then take a weighted average of all such paths, which generates a smooth trajectory, and update it using the iterative update law for $U_{k+1}$. At the next time step, the process is repeated.

**B. Stability Problems**

The cost function embedded within the $S(V)$ term, if correctly defined, is capable of pushing the system back towards a desired region of the state space, but this isn’t always
enough for the system to recover if it is suddenly perturbed. That is to say, if the system state suddenly changes at a given time step, as depicted in Figure 3

![Perturbed system state](image)

**Figure 3.** Perturbed system state

Then the sampling procedure does not account for this change beyond applying higher costs to paths that continue off course, which isn’t always enough to prevent the system state from continuing out of desirable regions in the state space. In the case of the autonomous driving task, this is particularly important as the vehicle is driving as rapidly as possible, so there may be very little time between when a system is perturbed and when it goes beyond the track boundaries. Addressing this problem is the focus of this work, of which Tube-MPPI has been a proposed solution.
CHAPTER 4

TUBE-MPPI

Reflecting on why the failure described in Chapter 3 occurs, we see that this is a common error in MPC methods. If at time step $t$ the system has control solution $(u_t, u_{t+1}, \ldots, u_{t+T})$, then $(u_{t+1}, \ldots)$ will be used as the initialization for the next time step. This assumes that the state at the next time step will be close to the one at the current time step which, in the presence of strong disturbances, is not necessarily true. Repeating the equations from Chapter 1, the MPC framework assumes that the state evolves according to:

$$x_{t+1} = F(x_t, u_t + \epsilon)$$

Where $\epsilon \in \mathcal{N}(0, \Sigma)$ is noise on the control input. When, in reality, the system includes state noise from modeling errors or unobserved environmental disturbances:

$$x_{t+1} = F(x_t, u_t + \epsilon) + w_t$$

Investigated in parallel to this thesis project was the framework presented in [7], to which this work itself is future work and contribution.\(^1\) In that work, the authors adapt the traditional non-linear Tube-MPC method to MPPI. In that method, the two systems listed above are considered together and followed by two controllers. One, the nominal controller is a non-linear MPC controller that computes a mapping from sequences of states to sequences of controls while ignoring system disturbances. It can also assume that the initial state fed into it is not the same as the actual state. The second, ancillary

\(^1\)Specifically, the work discussed in this thesis is part of the method described in this chapter, which this author helped work on though the main contribution was in the PID controller.
controller is another MPC method that solves the tracking problem. Note that the nominal controller does not consider the initial nominal state as an input, but can consider the actual system state in cases where the control output by the ancillary controller outperforms that of the nominal controller. Using this method, it is possible to prove that the actual system state remains within a certain distance (or tube) centered around the nominal state sequence, but the size of this tube is challenging to compute.

For the nominal controller, the authors use MPPI in the form described throughout this paper. Unlike in the original description of non-linear Tube-MPC, in which the initial nominal state is set as the actual state and then simulated forward without ever getting any actual state feedback, [7] follows an approach somewhat akin to that of [13]. Effectively, they run the nominal, MPPI controller on both the nominal state and the actual system state, then accept the solution from the nominal controller if the cost it produces is less than that of the nominal state solution plus some threshold. The algorithm to choose which solution to use is laid out in Algorithm 2 of [7].

For the ancillary controller, the authors use an iterative linear quadratic gaussian controller to solve the tracking problem. This controller serves to maintain the actual system state within a tube around the nominal state.
CHAPTER 5

EXPERIMENTS

A. Disclaimer

The experiments discussed in section B of this chapter were performed by the authors of [7], who this author worked closely with, but without contribution from this author. They are discussed here because they are directly relevant to the project at hand and serve as valuable reference point, though all credit is given to the authors of [7]. Perhaps more importantly, since the experiments conducted in section C by this author were relatively unsuccessful, the discussion in section B serves as evidence that the conceptual framework discussed in this thesis is sound.

B. Experimental Results [7]

This framework was tested in several experiments. In all simulations, the goal was to compare the performance of Tube-MPPI from Chapter 4 to that of vanilla MPPI from chapter 3, particularly in the case where significant disturbances are added. Every experiment thus features three test conditions: baseline-MPPI (from chapter 3), disturbance-MPPI (the controller from chapter 3 in an environment with significantly higher disturbances), tube-MPPI (from chapter 4).

The first experiment conducted was in the task of stabilizing a point mass system as it moved at constant velocity around a ring centered about the origin. Mathematically, we repeat equations (13) and (14) from [7] which display the cost function associated with this test:
\[ C(x_c) = \left( \sqrt{v_x^2 + v_y^2} - v_{des} \right)^2 + 1000(1_c(x)) \]

\[ C = \{ x | 1.875 < \sqrt{x^2 + y^2} < 2.125 \} \]

The noise for the disturbance-MPPI controller is set 10 times higher than for the baseline in this case. Note the hard constraint \( I_C \) in the second term of the cost function that sets the outside of the boundary as having a very high cost. The authors of [7] found that for the low-noise system, Baseline-MPPI performs very well:

![Graph showing Baseline-MPPI performance on the point-mass task](image)

**Figure 4.** Baseline-MPPI performance on the point-mass task

However, when the noisiness of the system is increased ten-fold, baseline-MPPI produces highly undesirable solutions:

---

2 Figures repeated from [7], credits for the plots go to the authors of the paper.
When Tube-MPPI is applied to that same environment, however, the solutions it produces all remain within the desirable bound:

These results clearly show that tube-MPPI outperforms baseline-MPPI in cases where the environment is extra noisy.

The next experiments focused on an autonomous racing task. The first step, which is a computational simulation, is performed in Robot Operating System’s Gazebo environment, which provides real time simulation capabilities on a realistic physics
engine. The simulation involves controlling a 1/5th scale autonomous vehicle around a roughly elliptical race track. The underlying model of the vehicle is not given in the simulations, so the neural-network dynamics model is used instead in the form presented in [29], whereby it is continuously optimized online using LW-PR². The state space of the vehicle is $x = (x, y, \theta, r, v_x, v_y, \dot{\theta})$, and its values throughout the trials are stored for post-processing. The simulations involve making the vehicle drive multiple laps around the track to verify that it is capable of autonomous control with the same order of magnitude efficiency as previous iterations of the algorithm. Performance metrics recorded include average lap time, best lap time, top speed, and maximum slip. The cost function for this task are as defined in (15)-(17) of [7]. The authors of [7] obtained the following results, ignoring uncertainties:

<table>
<thead>
<tr>
<th>Method</th>
<th>Avg. Lap Time (s)</th>
<th>Max Speed (m/s)</th>
<th>Max Slip (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disturbance-MPPI</td>
<td>11.87</td>
<td>5.22</td>
<td>0.04</td>
</tr>
<tr>
<td>Tuned-MPPI</td>
<td>8.33</td>
<td>7.53</td>
<td>0.09</td>
</tr>
<tr>
<td>Tube-MPPI</td>
<td>9.39</td>
<td>7.51</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Here, Tube-MPPI refers to baseline-MPPI after it has gone extensive tuning with a cost function for the track the simulations were run on. Notice how the disturbance-MPPI method sees a significant drop in performance compared to the tuned-MPPI algorithm, but the Tube-MPPI retains approximately equivalent performance despite not having received as much tuning.
Finally experimental validation of the algorithm was then performed on the physical AutoRally robotics platform at the Georgia Tech Autonomous Racing Facility (GT-ARF). The AutoRally platform is an all-electric autonomous race-car testbed that is one-fifth the size of a passenger vehicle. It is equipped with an onboard Mini-ITX computer housed in a metallic enclosure. The sensors available to the computer include two forward facing Point Grey USB3.0 cameras with 70 degree Field of View lenses, a Lord Microstrain 3DM-GX4-25 IMU, an RTC corrected Hemisphere GPS receiver, and Hall Effect wheel speed sensors. The robot weights roughly 22kg, is 0.9 meters long, and has a top speed of 27 m/s. It is capable of fully autonomous testing of all algorithms without relying on any external position system other than a GPS receiver. Build, configuration, and operation instructions for the AutoRally are available, along with the Gazebo simulation at reference [28]. Note that the cost function for this test is such that the system is only penalized for leaving the track boundaries, so it can use the entire track to accomplish its task. The results, shown in Figures 6, and 7 and [7] highlight extremely well how the Tube-MPPI algorithm is capable of driving the AutoRally car around the tracks at speeds of up to 8.52 m/s around a dirt track with a number of very sharp turns.

C. PID Experimental Results

Proportional-Integral-Derivative (PID) controllers are a classical control closed-loop feedback system commonly used across the controls and industrial systems. Given a desired state $x_d$, and the ability to track their given state $x_{\text{actual}}$, PID controllers continuously compute an error term $e(t)$ as the difference between those two states. It then computes control values according to the expression:
\[ u(t) = K_p e(t) + K_i \int_0^t e(t') dt' + K_d \frac{de(t)}{dt} \]

Where the first term contributes proportionately to the control value, the second term accounts for the change in error over time through integration, and the third contributes by looking at the instantaneous change in error.

Such a PID controller was implemented to replace the iLQG with the intention of performing the simulated and physical self-driving experimental tasks, where it was expected that the former outperform the latter. The PID controller was first tasked with independently driving the Autorally around the simulated track by being given a set of waypoints and instantaneously tracking the closest one ahead of it. In these experiments, the controller was given a specific target velocity to maintain. Table 2 illustrates that the controller, after thorough tuning, was able to drive around the track reliably up to speeds of just over 6 m/s, before becoming unable to manage a turn. Note that the set of waypoints contained enough points to complete 5 laps, so if a controller managed 5 laps, then it would most likely be able to run indefinitely.

**Table 2: PID performance on simulated driving task**

<table>
<thead>
<tr>
<th>Target Speed (m/s)</th>
<th>Number of laps successfully driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2.5</td>
</tr>
<tr>
<td>7</td>
<td>0.75</td>
</tr>
<tr>
<td>8</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Where, at 8m/s, the simulated AutoRally controlled by the PID would not make it past the first term. This is not dramatic, as the purpose of the PID controller was primarily to serve as the ancillary controller within the Tube-MPPI framework. However, when
embedded into the Tube-MPPI framework, the system became unable to controller the Autorally at all. It remains unclear why this was the case, as the behavior displayed by the system began with jerky, sporadic forward motions, followed by either aggressive throttle and steering inputs that pushed the system off track, or no response at all. So, while the PID controller itself was capable to driving the system around the tracks indefinitely for speeds of up to roughly 5.5 m/s, the Tube-MPPI framework failed with the PID controller serving as ancillary controller.
CHAPTER 5

CONCLUSIONS AND FUTURE WORKS

While it is disappointing that the PID controllers has not yet proven capable of handling the racing task for which they were designed when placed within Tube-MPPI, the framework discussed to address the stability dilemma of MPPI, which was thoroughly evaluated in [7], has been shown to be theoretically and practically sound. The expectation remains that the PID controller would outperform the iLQG as an ancillary controller in terms of computational time, due to the greater simplicity of the PID computation, and in minimizing divergence of the actual state from the nominal state, provided that it is adequately tuned. However, the root cause of the failure remains unclear, it is likely an underlying error within the Gazebo simulations.

A first direction for future research would be to attempt alternatives to the PID and iLQG ancillary controllers that are known in literature to have potential to perform better. One possible such controller would be an $\mathcal{L}_1$ adaptive controller. The work presented here also did not focus on the theoretical points of the modified MPPI framework, such as computing a bound for the tubes. This could be useful if those bounds could be manually adjusted and defined to ensure specific performance by the controller. So, a future direction for research would be to obtain such guarantees, or make sure that they exist in the first place, and then implement them in the cost function definition.

While primarily tested on an aggressive self-driving task, the framework presented in this thesis is capable of handling any stochastic, non-linear system, provided
that a system dynamics model that can project the state forward in time exists, and that a cost function can be defined. As such, the method is open to a much broader set of applications in which one hopes to have a system accomplish a task as efficiently as possible, while displaying stable, and reliable behavior.
REFERENCES


