Fast Model Predictive Control

Akshay Katpatal
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
aykatpatal@wpi.edu

Bryan Rathos
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
brathos@wpi.edu

Denim Patel
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, USA
dmpatel@wpi.edu

***Abstract*—One of the major areas of focus in the field of autonomous vehicles is safe navigation and control. While there are a plethora of techniques available for control, one of the most studied technique is Model Predictive Control (MPC) of vehicle. An even faster way is the Fast MPC implementation which improves the speed of standard MPC by exploiting the structure of the problem that suits the particular problem. In this paper, we discuss the different aspects of design and implementation of a fast MPC model on the 1/10th scale F1 race car. A non-linear bicycle model is linearized and used. In this paper we posit some techniques that can be incorporated on MPC to reach real-time constraints by reducing the computation cost of optimization. FastMPC provides the suboptimal solution yet satisfies the inequality constrains. This way we can be sure that the control strategy will never produce the control input to the system which can't be implementable.**

***Index Terms*—Model Predictive Control, Primal Barrier methods, Autonomous Vehicles, Racecar**

I. Introduction

With the recent advances in the field of computing and sensing, there has been groundbreaking research in the area of autonomous vehicles. The three major components necessary for the functioning of an autonomous vehicle are localization, perception and control.A major challenge with driverless cars is accurately navigating the desired trajectory ensuring the safety of the passengers and the surrounding environment in the dynamic road environment which involves pedestrians, other vehicles and unforeseen obstacles. .In this paper, we deal with the control aspect of an autonomous vehicle which involves controlling the vehicle's braking, steering and acceleration using Fast Model Predictive Control (MPC). Considering how dynamic the road environment is, MPC seems to be the best possible solution that ensures accuracy as well as safety of everyone involved. In MPC, an optimal control sequence is computed at every time step by solving an optimization problem. Generally, the first control is implemented and for every future time steps a new sequence of control is calculated using new state information.

The robustness of a system depends on its stability when met with disturbances and uncertainties. A common assumption with nominal MPC is that all the external disturbances and model uncertainties can be accounted for in the feedback. However, we cannot assume that a controller that has good performance for a specific model will give us the same results when we try to apply it to a physical system having uncertainties and inaccuracies in the model. For a lot of the published MPC schemes, a dynamic vehicle model is used in combination with linear tire models. The issue with the tire model is that it is computationally expensive . This can be addressed by using a kinematic bicycle model[1] instead of the dynamic model.

II. Problem Statement

The main aim of this project is to fundamentally understand the Model Predictive Control algorithm implement the Fast MPC algorithm in simulation on MatLAB to compare its performance to the existing Nominal MPC implementation in terms of accuracy in tracking the trajectory and the execution time required. The available literature provides adequate proof to support the idea that the Fast MPC formulation can help increase the frequency of operation.

III. Problem Formulation

A. Kinematic Bicycle Model

For the simulation of fast MPC algorithm in MatLab, we will be considering the non-linear kinematic bicycle model(Fig 1). The bicycle model being a simple one, allows for steering of only the front wheel and as a result, the front steering angle δ and the velocity v are the only control inputs.

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{L}\tan\delta \end{cases} \tag{1}$$

or, in a compact form as,

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{2}$$

Here, x and y represent the center of mass of the vehicle, δ gives us the steering angle of the car which is one of the control inputs.

B. State Estimation

When implementing MPC on a real car system we estimate the car's state with the use of GPS, IMUs (Inertial Measurement Units), RADAR (Radio Detection and Ranging) and LIDAR (Light Detection and Ranging). We need to fuse all these sensor data and apply probabilistic methods such as Kalman filters or Extended Kalman Filters to estimate

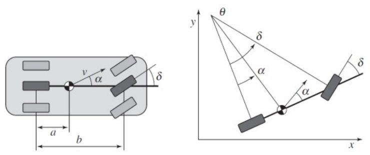

Fig. 1. Non-Linear Kinematic Bicycle Model

pose and orientation of the vehicle. These methods are very convoluted and are out of scope of the MPC project. In our project we will be simulating the fast MPC algorithm in MATLAB and therefore the car would track the trajectory with almost no error and we would know the exact position and orientation of the car.

C. Trajectory Generation

In autonomous vehicles, trajectory is generated by the path planning module, but in our project we will be providing the reference trajectory to be tracked. The reference trajectory is generated by using cubic interpolation splines to connect the initial and final goal via feasible way-points. Time based cubic polynomials are generated for x_{ref} and y_{ref} as shown below. Additionally we also use the non-holonomic constraints and the dynamic model equations to compute coefficients of the cubic polynomial trajectories [2]

$$\begin{aligned} a_0 + a_1 t + a_2 t^2 + a_3 t^3 &= x_{ref}(t) \\ b_0 + b_1 t + b_2 t^2 + b_3 t^3 &= y_{ref}(t) \\ \dot{x}\sin\theta - \dot{y}\cos\theta &= 0 \\ \dot{x}\cos\theta + \dot{y}\sin\theta &= v \end{aligned} \tag{3}$$

IV. Model Predictive Control

Model predictive control(MPC) is an advanced technique of process control. MPC takes care of constrains as well as model dynamics and previously generated control signals while generating the new model input for the system. The main advantage of Model predictive control is that it provides the current output by optimizing the future outputs that can generate the best result in that particular system. The basic block diagram of the MPC is given below.

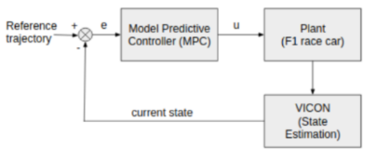

Fig. 2. Block diagram of Nominal MPC

From the above figure, it is easy to understand that MPC takes the reference trajectory and state of the system to generate it's output which can then be fed to plant.The MPC itself contains the model of the plant that helps to predict the future state of the system even before applying the control to plant. In simple terms, using the reference geometry and the plant model, MPC generates the optimal control sequence for the current situation. To understand MPC, it is recommended to understand the terms such as Prediction Horizon, Control Horizon and Receding Horizon[4]. Prediction Horizon is the Number of time step into the future that MPC will consider in each iteration to optimize the control inputs of the plant. Control Horizon is the number of time steps into which the MPC will generate a control state into the future in each iteration. Receding Horizon control depends on the prediction and control horizon which are shifted every time step till we get to the goal. Generally the Prediction horizon is 8 to it

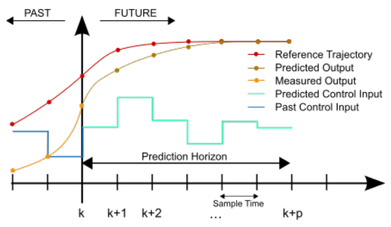

Fig. 3. MPC basic scheme and terminology

times longer than Control horizon, due to the fact that first few control inputs contribute the most to bring the state to the desired reference and by pruning the control horizon, MPC can greatly reduce the computational burden per time step.

The main drawback of the Classical MPC is that it predicts the system based on the current pose at every time step and to optimize the control signals in each time step using the dynamic model of the system is computationally expensive. The main reason for this is because MPC needs to handle the optimization problem using Quadratic Programming(QP). Solving QP using general purpose methods is computationally costly. In general MPC is used in slow dynamic application with sample time in seconds or minutes.

V. Linear MPC Approach

In the numerous Non-linear MPC (NMPC) schemes proposed in the literature [6], one important point to note is that the computation effort necessary is much higher than the linear MPC version. NMPC is a nonlinear programming problem which needs to be solved online is nonconvex and it is difficult to find he global minimum in general. In the linear MPC approach, the fundamental idea employed is the successive linearization approach, which yields a time varying system. The decision variables considered here are the inputs, which are transformed into an optimization problem to be solved at each sampling time in a quadratic programming (QP) problem. Since these are convex in nature, QP problems can be solved easily by algorithms which are numerically robust that lead to solutions that are optimal globally.

We can obtain a linear model of the system dynamics by computing an error model with respect to a reference car. We can expand the state model in the taylor series form around a desired reference point (x_r,u_r) and discard the higher order terms that follow. This is also known as "Jacobian Linearization"

$$\dot{\mathbf{x}} = f(\mathbf{x_r}, \mathbf{u_r}) + \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\Bigg| \tag{4}$$

or,

$$\dot{\mathbf{x}} = f(\mathbf{x_r}, \mathbf{u_r}) + f_{\mathbf{x},\mathbf{r}}(\mathbf{x} - \mathbf{x_r}) + f_{\mathbf{u},\mathbf{r}}(\mathbf{u} - \mathbf{u_r}) \tag{5}$$

where $f_{\mathbf{x},r}$ and $f_{\mathbf{u},r}$ are the jacobians of f with respect to $\mathbf{x}$ and $\mathbf{u}$ respectively. These jacobians are evaluated around the reference point $(\mathbf{x_r}, \mathbf{u_r})$

When we subtract (2) from (5) we get,

$$\dot{\tilde{x}} = f_{\mathbf{x},\mathbf{r}}\tilde{x} + f_{\mathbf{u},\mathbf{r}}\tilde{\mathbf{u}} \tag{6}$$

where, $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x_r}$ is the error with respect to the reference and $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u_r}$ is the error in the control input.

The discrete-time system model is given by the approximation of $\dot{\mathbf{x}}$ using the forward differences:

$$\tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}(k)\tilde{\mathbf{u}}(k) \tag{7}$$

with

$$\mathbf{A}(k) = \begin{bmatrix} 1 & 0 & -v_r(k)\sin\theta_r(k)T \\ 0 & 1 & v_r(k)\cos\theta_r(k)T \\ 0 & 0 & 0 \end{bmatrix} \tag{8}$$

$$\mathbf{B}(k) = \begin{bmatrix} \cos\theta_r(k)T & 0 \\ \sin\theta_r(k)T & 0 \\ \dfrac{\tan\delta(k)T}{L} & \dfrac{v_r\sec^2\delta(k)T}{L} \end{bmatrix} \tag{9}$$

In [7] it is shown that the nonlinear, nonholonomic system (1) is fully controllable, i.e., it can be steered from any initial state to any final state by using finite inputs.

$$\overline{\mathbf{A}}(k) = \begin{bmatrix} \mathbf{A}(k|k) \\ \mathbf{A}(k+1|k)\mathbf{A}(k|k) \\ \vdots \\ \alpha(k,2,0) \\ \alpha(k,1,0) \end{bmatrix} \tag{10}$$

Similarly we can obtain an equation for $\tilde{\mathbf{B}}(k)$

In contrast, we can observe that when the vehicle is not moving (i.e., $v_r = 0$, the linearization about a stationary operating point becomes uncontrollable. As long as the control input $\mathbf{u}$ is not zero, the linearization is controllable.

We can now reframe the optimization problem in a usual quadratic form. Therefore, we can define the following vectors:

$$\overline{\mathbf{x}}(k+1) = \begin{bmatrix} \tilde{\mathbf{x}}(k+1|k) \\ \tilde{\mathbf{x}}(k+2|k) \\ \vdots \\ \tilde{\mathbf{x}}(k+N|k) \end{bmatrix}, \overline{\mathbf{u}}(k) = \begin{bmatrix} \tilde{\mathbf{u}}(k|k) \\ \tilde{\mathbf{u}}(k+1|k) \\ \vdots \\ \tilde{\mathbf{u}}(k+N-1|k) \end{bmatrix}$$

We can now formulate the cost function as follows:

$$\Phi(k) = \overline{\mathbf{x}}^T(k+1)\overline{\mathbf{Q}}\overline{\mathbf{x}}(k+1) + \mathbf{u}^T(k)\overline{\mathbf{R}}\overline{\mathbf{u}}(k) \tag{11}$$

where, $\overline{\mathbf{Q}} = \text{diag}(\mathbf{Q}; \ldots; \mathbf{Q})$ and $\overline{\mathbf{R}} = \text{diag}(\mathbf{R}; \ldots; \mathbf{R})$.

We can now write $\overline{\mathbf{x}}(k+1)$ as,

$$\overline{\mathbf{x}}(k+1) = \overline{\mathbf{A}}(k)\overline{\mathbf{x}}(k|k) + \overline{\mathbf{B}}(k)\overline{\mathbf{u}}(k) \tag{12}$$

where $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are defined in (15) with $\alpha(k,j,l)$ given by:

$$\alpha(k,j,l) = \prod_{i=N-j}^{l} \mathbf{A}(k+i|k) \tag{13}$$

With some algebraic manipulations, the objective function can be rewritten n the standard quadratic form:

$$\overline{\Phi}(k) = \frac{1}{2}\overline{\mathbf{u}}^T(k)\mathbf{H}(k)\overline{\mathbf{u}}(k) + \mathbf{f}^T(k)\overline{\mathbf{u}}(k) \tag{14}$$

where,

$$\begin{aligned} \mathbf{H}(k) &= 2\left(\overline{\mathbf{B}}(k)^T\overline{\mathbf{Q}}\overline{\mathbf{B}}(k) + \overline{\mathbf{R}}\right) \\ \mathbf{f}(k) &= 2\overline{\mathbf{B}}^T(k)\overline{\mathbf{Q}}\overline{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k) \end{aligned} \tag{15}$$

The matrix $\mathbf{H}(k)$ is a Hessian matrix which is always positive definite. The Hessian is the quadratic part of the objective function. The vector $\mathbf{f}$ is the linear part of the objective function. The above objective function is the standard expression of QP optimization problems which has to solved at each time step which gives the optimal control input given by:

$$\tilde{\mathbf{u}}^* = \arg\min_{\mathbf{a}} \left\{ \overline{\Phi}'(k) \right\} \tag{16}$$

An important point to note is that the only the control inputs are used as decision variables.

In the next section, we introduce the Fast MPC as in [5], wherein the QP is rewritten in a compact form. The overall optimization variable is defined as,

$$z = (u(t), x(t+1), \ldots, u(t+T-1), x(t+T)) \in \mathbf{R}^{T(m+n)} \tag{17}$$

and express the QP as,

$$\begin{array}{ll} \text{minimize} & z^T H z + g^T z \\ \text{subject to} & Pz \leq h, \quad Cz = b \end{array} \tag{18}$$

VI. Fast Model Predictive Control

The goal of the Fast MPC[5] is reduce the computation time of MPC while also retaining its accuracy. This will allow it to be used on real world systems that have time steps in seconds. For that several strategies can be applied such as:

- Interior Point Method
- Warm Start
- (very) Early Termination

The main aim of our project was to explore each of these ideas mentioned above in detail and apply them to check their feasibility.

A. Interior Point Method

The problem of model predictive control can be reduced to be an optimization problem by formulating the system as given in [5]. For each iteration, the system finds an optimal control strategy. To find z each time, MPC uses Quadratic Programming approach. Quadratic programming has to take care of inequality constrains present in a system as we need to limit them due to the physical limitations of the vehicle such as maximum and minimum turning angle and maximum and minimum velocity that the vehicle allows.

QP iteratively solves the problem to find the best z as per the constraints. This is the most computationally expensive task. To reduce these computations, use of the following techniques has been proposed in [5].

- Log Primal Barrier Method
- Infeasible start Newton Method

1) **Log Primal Barrier Method**
We use the first approach i.e. the primal barrier method to reduce the computation of Fast MPC. Using this method, we can take care of inequality constraints by adding an appropriate term to cost function.

$$\phi(z) = \sum_{i=1}^{lT+k} -\log\left(h_i - p_i^T z\right) \tag{19}$$

From the above problem formulation, it is clear that, inequality constraints can be taken care of by considering an added cost to the cost function.In simple terms, primal barrier increases overall cost along the foundries exponentially. Thus, we can omit the inequality constraints from the system as we can assume that the systems optimal point wont be outside of the inequality constraints. Thus, the problem becomes a convex optimization problem.

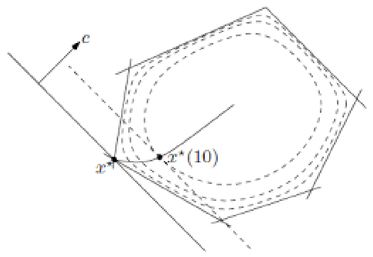

Fig. 4. Simple primal barrier

In this approach, the value of κ is reduced and the solution is found iteratively. As a rule of thumb, κ which is the barrier parameter is reduced by 10 in each iteration till the system converges to the desired optimal state or the maximum number of iterations allowed is reached. These different strategies of deciding the barrier cost to reduce the computation time was explored and the results are presented in subsequent sections.

2) **Newton Infeasible Start**

In the infeasible start Newton method, an initial guess of 'z' is initialized considering only the inequality constraint(P*z ¡ h) . This initial guess should strictly satisfy the inequality constraint but it need not satisfy the equality constraints.

To find the optimal condition from Newton method, we iteratively find "delta z" and "delta v" such that the residual goes to zero and system converges to the desired values.

$$\begin{bmatrix} 2H + \kappa P^T \operatorname{diag}(d)^2 P & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta \nu \end{bmatrix} = - \begin{bmatrix} r_d \\ r_p \end{bmatrix} \tag{20}$$

Primal feasibility (Cz = b) will be achieved when the system converges and residual reaches to zero.

B. Warm Start

In simple MPC, the optimized parameter z is obtained starting from scratch in each iteration. However, for reducing the computation time, the previously computed z can be used as a starting point in the optimization process. To put it simply, the next T time steps are planned while the computation of the current action. However, the previously computed plan can be shifted in time and used as a starting point for the current plan In other words, the previously computed trajectory at time (t-1)sec is used at t sec as a starting point. Suppose at time (t-1) the trajectory is,

$$z = (u(t-1), x(t-1), \ldots, eu(t+T-2), x(t+T-1)) \tag{21}$$

The time step t can be initialized as,

$$z^{\text{init}} = (u(t), x(t+1), \ldots, x(t+T-1), \overline{u}^\star, \overline{x}^\star) \tag{22}$$

We have used warm start in two cases

- In Primal Barrier Method iterations
- Among consecutive frames

In Primal barrier method, we decrease the value of κ per iteration. In each iteration, previously computed z is used for next step. Among consecutive frames also, the "time shifted" z is used that helps to converge to the desired state faster as seen in equations (21) and (22).

VII. Approximate Primal Barrier Method

Primal barrier method reduces the computation time significantly compared to quadratic programming. Yet to achieve real-time computation constraints, different approaches can be utilized to further reduce the computation. Following are some of the techniques that can be used.

A. Fixed Barrier Parameter κ

Instead of decreasing the κ value gradually, κ can be set to a fixed value. Now this value remains the same throughout. This way the computation time per iteration reduces significantly by compromising on the optimality criteria. At the same time, the result satisfies the inequality constraints completely. We tried with different fixed value of κ to and performed experiments which are presented in results section.

B. Fixed Iteration Limit

This is another variation to reduce the computation time by limiting the maximum iterations to solve the optimization problem. In this case, the optimization step will be terminated two cases. One when maximum optimization iterations have been completed or when the system converges to the desired state even before maximum iterations are completed. Fixed iteration and fixed barrier parameter work best when they are used alongside warm start. In this case the κmax can be chosen as low as 2 to 5, greatly reducing the computations.

VIII. Results

The result given below will compare the two methods 1.Fixed Barrier Parameter κ 2.Fixed Iteration Limit in terms of trajectory tracking accuracy as well as the execution time. The velocity is fixed at around 5 m/sec as it should be kept as high as possible for the race car. The length of the horizon is also kept to be 5.

1. Fixed Iteration Limit:

When we fix the value of k and iteratively reduce it using the fixed iteration limit method, we get the following results.

A. Number of iteration = 3

As we can see from the figures below, the trajectory tracked is very similar to the desired trajectory and thus the error in both x and y can be see to converge to zero(fig. 6). The computation time is about 18 Hz. Here the value of κ parameter starts at 1 and is iteratively reduced by a factor of 10 for 3 iterations.

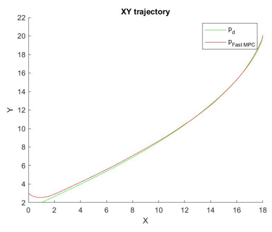

Fig. 5. Trajectory tracked vs Desired Trajectory

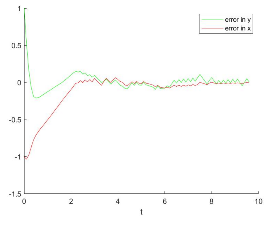

Fig. 6. Error in x and y

The plots for the trajectory tracked in x and trajectory tracked in y are seen below in figures 7 and 8. It can be clearly seen that the trajectory tracked in x converges to the desired trajectory and follows it effectively.

B. Number of iterations = 10

When we increase the number of iterations from 3 to 10,

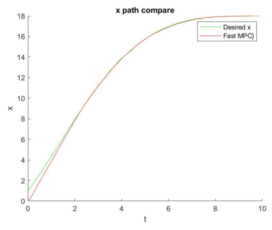

Fig. 7. Trajectory tracked vs Desired Trajectory in x

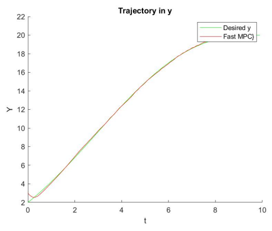

Fig. 8. Trajectory tracked vs Desired Trajectory

there is a considerable increase in the computations required and thus the computation time reduces by a factor of 3 and becomes around 6 Hz. It can also be seen in figure 9 and 10 that there isn't a considerable improvement in the actual results to warrant this time increase.

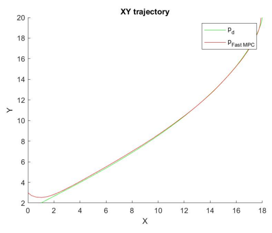

Fig. 9. Trajectory tracked vs Desired Trajectory

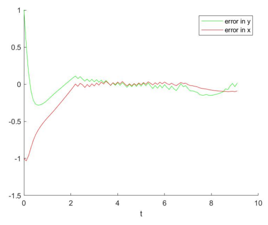

Fig. 10. Error in x and y

2. Fixed Barrier Parameter: When we fix the κ value over the entire computation, it can seen from the figures below that there isn't a considerable change in the performance provided that the value of k is tuned properly. When we fixed the value of k to be low, the performance was quite good and the error converged to zero and also the computation time was about 40 Hz.

A. Value of κ = 1

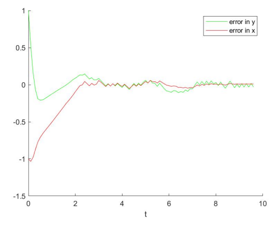

Fig. 11. Error in x and y when κ is fixed = 1

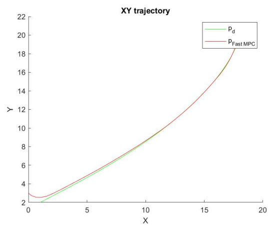

Fig. 12. Desired Trajectory vs Tracked Trajectory when κ is fixed = 1

B. Value of κ = 100

When the value of κ is increased and fixed to 100, the performance degrades a lot and the error takes time to converge to zero. The trajectory tracking performance is also not as good. This further supports the hypothesis of [5] that the value of k should be kept as low as possible. The results can be seen in figures 13 and 14.

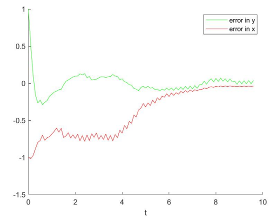

Fig. 13. Error in x and y when κ is fixed = 100

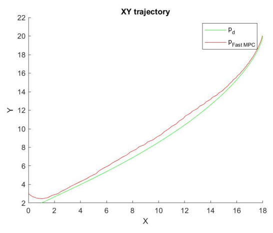

Fig. 14. Desired Trajectory vs Tracked Trajectory when κ is fixed = 100

When we increase the length of the horizon from 5 to 10, the computation time does not change a lot. For the case of fixed value of κ, the difference is as low as 2-3 Hz. The performance doesn't change a lot as well as seen in figures 15 and 16.

To conclude, the fixed barrier parameter(κ) method with a comparatively smaller value of κ seems to work best considering the computation time and the accuracy. However, the fixed number of iterations method also gives decent results in terms of computation time when the value of κ is iteratively reduced only 3-4 times i.e there are 3 iterations. There isn't a considerable increase in computation time when we increase the horizon from 5 to 10.

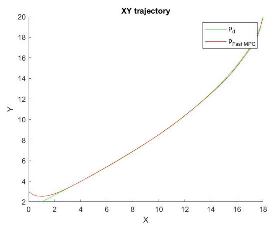

Fig. 15. Desired Trajectory vs Tracked Trajectory when Horizon=10

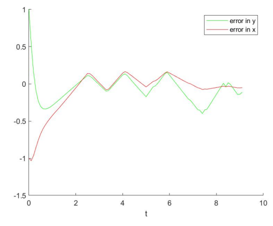

Fig. 16. Error in x and y when Horizon=10

IX. Future Work

The infeasible start newton method can be implemented as it will be useful in situations where a feasible start point is not available. The primal barrier method and the infeasible start method can be compared and whichever is better in terms of the execution time and accuracy can be chosen. However, we can also design a system wherein both of the methods can be present and one can be chosen based on the available conditions. Implementing the code on a Turtlebot or the F1/10 race car in the lab is the next logical step. Robust MPC can also be implemented which will reduce the time required for error minimization as well as allow for tracking of sharp curves.

References

[1] J. Kong, M. Pfeiffer, G. Schildbach, F. Borrelli, Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design, in Proceedings of the IEEE Intelligent Vehicle Symposium, pp. 1094-1099, June 2015.

[2] J. Chen, A. Huynh, Z. Jiang, Z. Wu. Self Driving on 1/10 Racer Car, Worcester Polytechnic Institute. 2018.

[3] Pierre Merriaux, Yohan Dupuis, Rmi Boutteau, Pascal Vasseur, Xavier Savatier. A Study of Vicon System Positioning Performance. Sensors, MDPI, 2017, 17 (7), 10.3390/s17071591.hal-01710399.

[4] Model Predictive Control of Hybrid Systems of Hybrid Systems by Alberto Bemporad; http://cse.lab.imtlucca.it/bemporad/hybrid/school07/pdf/08.Bemporad.pdf.

[5] Yang Wang, Stephen Boyd. Fast Model Predictive Control Using Online Optimization, Proceedings of the 17th World Congress The International Federation of Automatic Control Seoul, Korea, July 6-11, 2008.

[6] Chen, H. e Allgwer, F. (1998). A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability, Automatica 34(10): 12051217.

[7] Bloch, A. M. e McClamroch, N. H. (1989). Control of mechanical systems with classical nonholonomic constraint, Proceedings of the IEEE Conference on Decision and Control, Tampa,FL, USA, pp. 201205.

[8] Khne, Felipe Manoel, Joo Da, Gomes Fetter, Lages. (2019). Mobile robot trajectory tracking using model predictive control.