Controlling A Self Driving Car

Laukik Mujumdar
Yash Mandlik
Gautam Sharma
Chaitanya Kulkarni

Abstract

This project implements and compares path tracking methods for controlling a model of an autonomous vehicle along a predetermined path. We compare the performance of a full state feedback controller, a proportional integral derivative controller and a model predictive controller on a kinematic bicycle model of a vehicle, using just the steering angle as the input. We also develop a pure pursuit controller and test it's performance on the model. Additionally, we develop additional controllers to include the steering angle and the acceleration as inputs to the system.

Contents

1 Introduction

Autonomous vehicle and technology has been a major topic of research and development in the automotive industry in the past decade. Current research on driver assistance systems is based on sensors that measure the internal states of vehicles. These sensors enable the control of vehicle dynamics so that the desired trajectory is followed in the best way possible. This family of sensors and vehicle controllers are called path trackers. Path tracking refers to a vehicle executing a predefined geometric path by applying steering motions that guide the vehicle along the path. The goal of a path tracking controller is to minimize the lateral distance between the vehicle and the defined path, minimize the difference in the vehicle's heading and the defined path's heading and limit steering inputs to smooth motions while maintaining stability.

2 Vehicle model

The motion of the vehicle can be estimated by dynamic and kinematic models. The kinematic models are used to design controllers for low speed maneuvers such as tight parking spaces and in some cases, for motion planning. Dynamic models are used in high speed vehicle motion such as on highways or urban driving.

Simplifying a vehicle model to a kinematic bicycle model is a common approximation for motion planning and simple vehicle analysis. Such a model is included in the following figure -

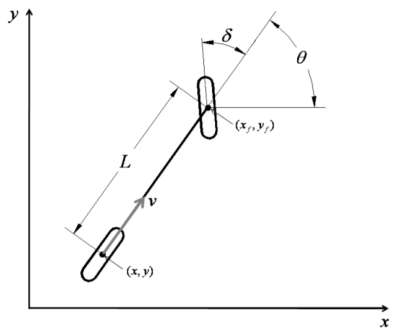


Figure 1: Kinematic bicycle model

The kinematic equations of motion which describe the motion in the global coordinate system, become -

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ (V/L)\tan\delta \end{bmatrix} \tag{1}$$

The longitudinal and lateral motions of the vehicle are determined by the control variables; velocity V and the steering angle δ at the tires. Given the motion inputs (V,δ) the motion trajectories of the vehicle can be simulated[**4**].

On linearizing the above kinematic equations, the state space model of the system is obtained as -

$$A = \begin{bmatrix} 0 & V \\ 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} V \\ V/L \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad D = 0$$

3 Controllers

Using the motion inputs (V,δ), we implemented and compared the path tracking performance of a model predictive controller, a proportional integral derivative controller and a linear quadratic regulator.

3.1 Full State Feedback Controller

Full state feedback is a method employed in feedback control system theory to place the closed-loop poles of a plant in predetermined locations in the s-plane. Placing poles is desirable because the location of the poles corresponds directly to the eigenvalues of the system, which control the characteristics of the response of the system. We implemented a full state feedback controller on our system by linearizing the vehicle model and determining the state space equations of the model. The gain matrix for the controller was determined using using pole placement by Ackermann's method[**3**]. The model is as shown below -

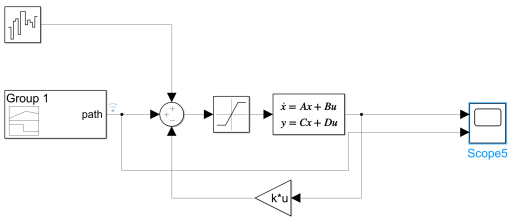
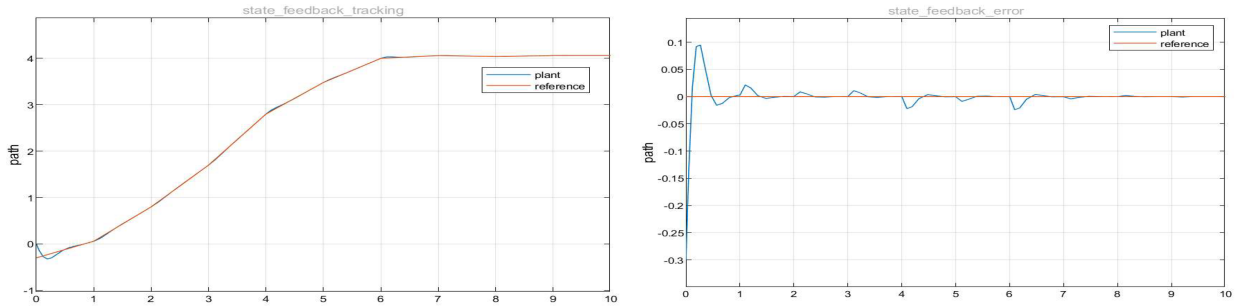


Figure 2: Full state feedback model

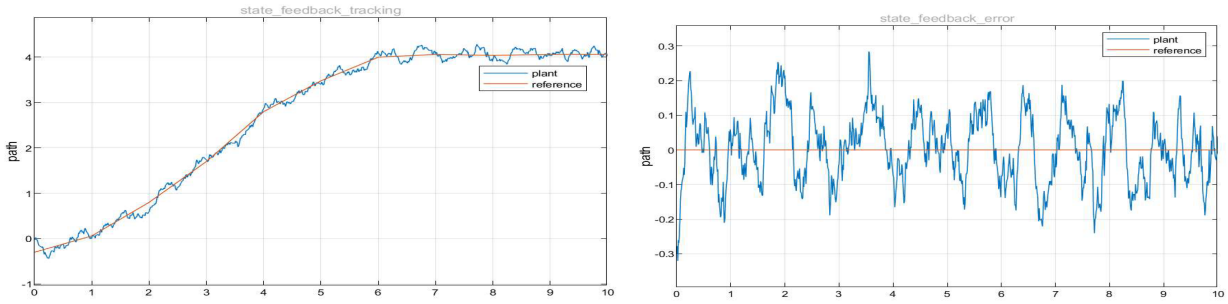
The tracking performance of the feedback controller and the error in the tracking, in the absence of disturbances, is as shown in the figures below -



(a) Path tracking of FSF model (b) Error

Figure 3: Tracking and error of a FSF model

On adding a measured white noise disturbance to the system, the variation in the performance and error is as shown -



(a) Path tracking of FSF model (b) Error

Figure 4: Tracking and error of a FSF model(with external disturbance)

3.2 PID Controller

A proportional–integral–derivative controller (PID controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications. A PID controller continuously calculates an error value *e(t)* as the difference between a desired reference value and a measured variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively)[**2**]. The model using a PID controller is as shown below -

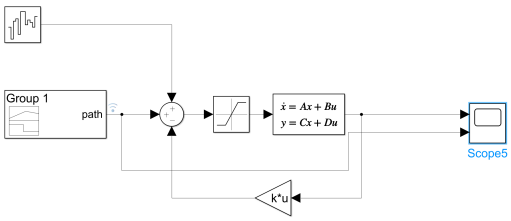
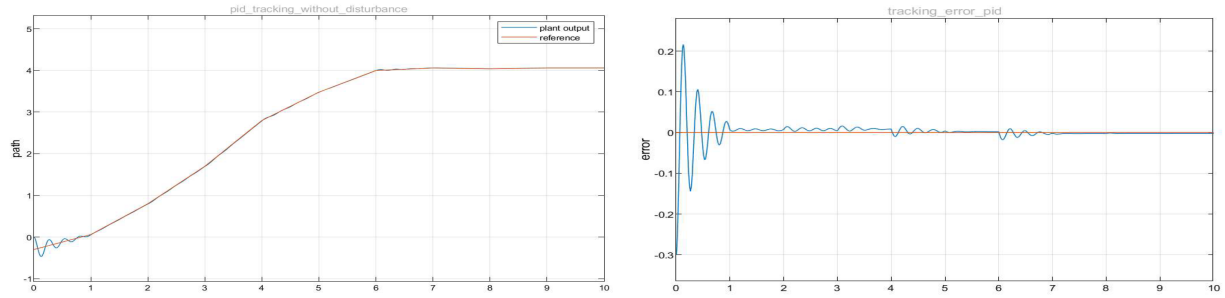


Figure 5: Model implementing the PID controller

The tracking performance of the PID controller and the error in the tracking, in the absence of disturbances, is as shown in the figures below -

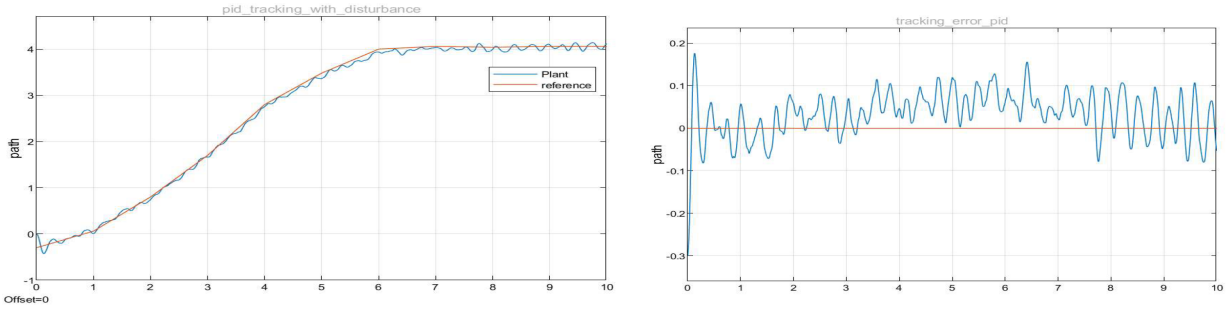


(a) Path tracking of the PID model

(b) Error

Figure 6: Tracking and error of a PID model

On adding a measured white noise disturbance to the system, the variation in the performance and error is as shown -



(a) Path tracking of the PID model

(b) Error

Figure 7: Tracking and error of a PID model

3.3 Model Predictive Controller

Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. MPC is based on iterative, finite-horizon optimization of a plant model. At time t the current plant state is sampled and a cost minimizing control strategy is computed (via a numerical minimization algorithm) for a relatively short time horizon in the future: $[t, t+T]$[**1**]. The model of the MPC implementation is shown below -

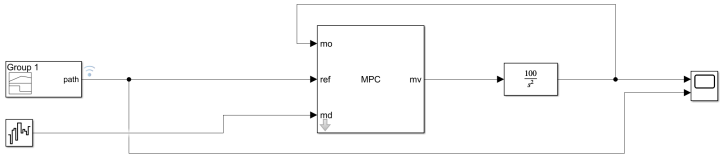
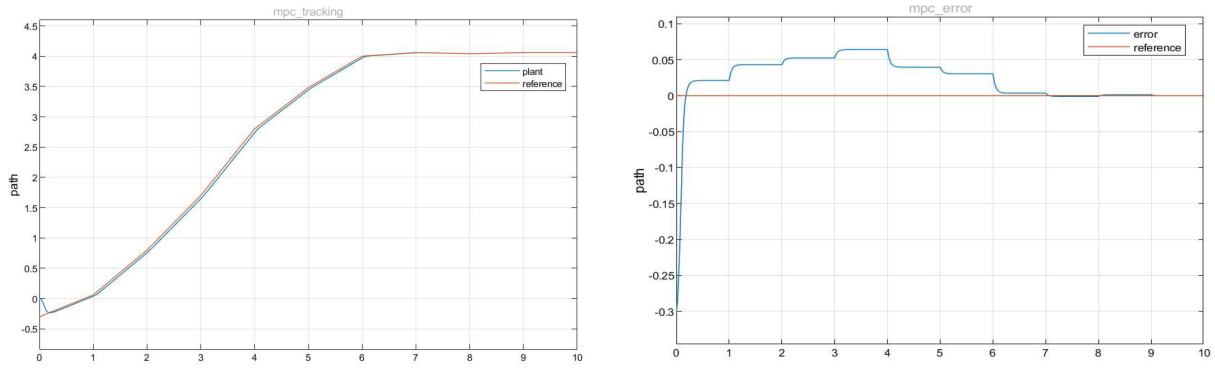


Figure 8: Implementing the Model Predictive Control

The tracking performance of the MPC controller and the error in the tracking, in the absence of disturbances, is as shown in the figures below -

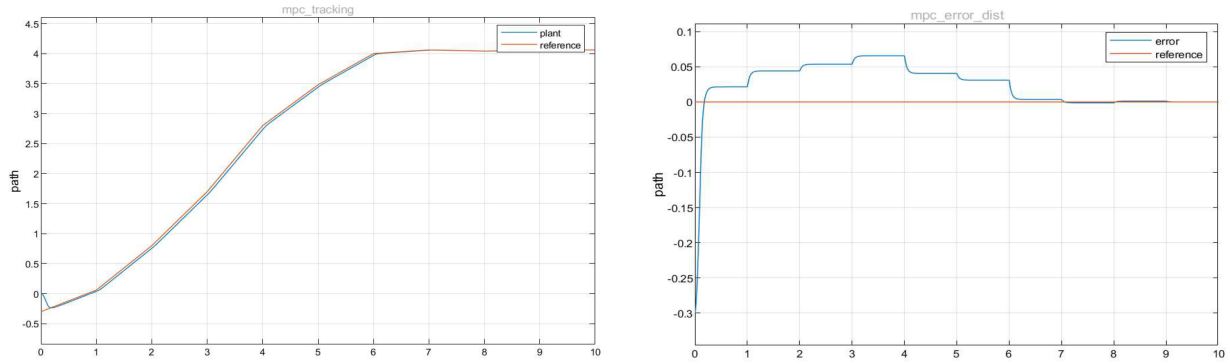


(a) Path tracking of the MPC controller

(b) Error

Figure 9: Tracking and error of MPC model

On adding a measured white noise disturbance to the system, the variation in the performance and error is as shown -



(a) Path tracking of the MPC controller

(b) Error

Figure 10: Tracking and error of MPC model

3.4 Pure Pursuit Controller

The pure pursuit controller is one of the most common controllers used in the domain of robotics to control the lateral trajectory of a vehicle. It neglects the dynamics of the car for simplicity and approaches the tracking problem keeping the kinematics of the car as the basis and has a control effort proportional to the lateral error[**6**].

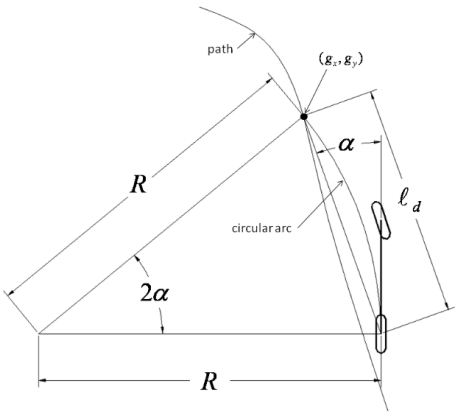


Figure 11: Pure Pursuit model

The steering angle is assumed as the only input in this model. The lateral trajectory tracking is mostly done by using the steering angle as an input and working with the lateral dynamics of the car. The use of longitudinal dynamics in lateral stability is minimal most of the times. We have followed the same approach in this project. We have assumed the steering angle to follow the Ackerman constraints such that

$$\tan(\delta) = L/R \tag{2}$$

The end goal of this approach is to make the rear wheel of the car follow the given arc of the trajectory by manipulating the steering input. Here l_d is the look ahead distance which is the distance from the rear wheel to a goal point (g_x, g_y) that we want the wheel to be at a certain time. This distance can be varied and tuned. Assuming K to be the curvature of the arc we want to follow, law of sines gives us the following results:

$$\frac{l_d}{sin(2\alpha)} = \frac{R}{sin(\pi/2 - \alpha)}$$

$$\frac{l_d}{2sin(\alpha)cos(\alpha)} = \frac{R}{cos(\alpha)}$$

$$\frac{l_d}{sin(\alpha)} = 2R$$

$$K = \frac{2sin(\alpha)}{l_d} sd \tag{3}$$

Using Eq. 2 and 3, the steering input can be written as:

$$\delta(t) = tan^{-1}(\frac{2Lsin(\alpha(t))}{l_d}) \tag{4}$$

where $sin(\alpha) = e_{l_d}/l_d$, e_{l_d} is defined as the distance between heading vector and goal point.
Eq.3 can be manipulated to show the relation:

$$K = e_{l_d} \frac{2}{l_d^2} \tag{5}$$

Eq. 5 establishes a proportional relationship with respect to the error dynamics. If the error increases, K increases and vice-versa. Using Eq. 4 as the control input, and a base lateral trajectory defined, the following is the result of the pure pursuit controller:

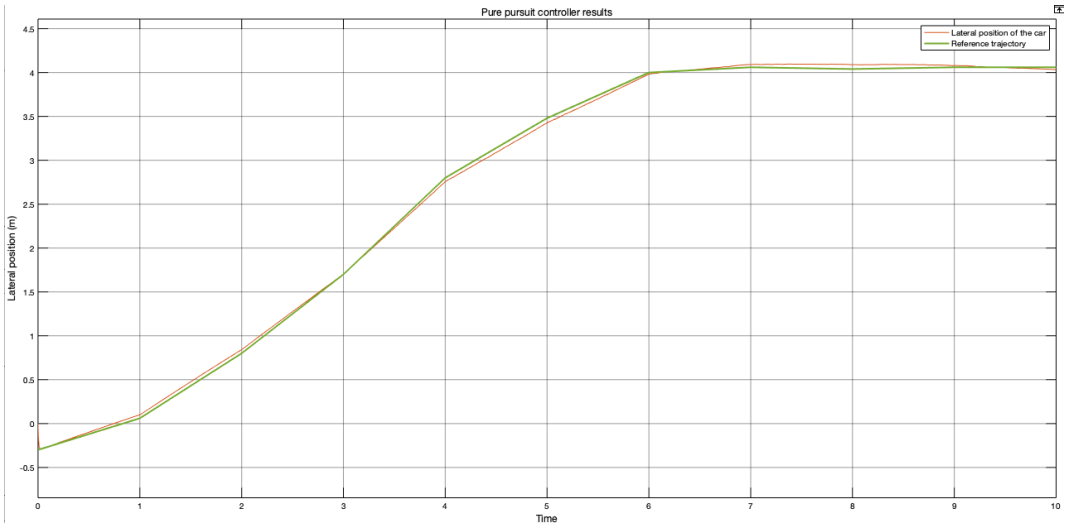


Figure 12: Trajectory tracking using pure pursuit controller

4 Modelling The Gas Pedal

Until now, we assumed the velocity of the car to be constant, and ignored the effect of the gas pedal. We modify the original state space model as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{V} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ U_1 \\ (V/L)U_2 \end{bmatrix}$$

where U_1 represents the gas pedal and U_2 represents the tangent of the steering angle.

4.1 Reference Trajectory

The reference trajectory is defined as a function of time. Only the x and y co-ordinates are defined as functions of time. Using the model structure, we can compute the other states also as functions of time. This is done as follows:

$$\begin{aligned} V(t) &= \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \\ \theta(t) &= \arctan(\frac{\dot{y}(t)}{\dot{x}(t)}) \end{aligned} \tag{6}$$

Also note that a reference trajectory can obey the model equations in presence of some reference input. Thanks to the structure of the model, the reference inputs U_{ref1} and U_{ref2} can be computed as follows:

$$\begin{aligned} U_{ref1} &= \dot{V}(t) \\ U_{ref2} &= \dot{\theta}(t) * \frac{L}{V(t)} \end{aligned} \tag{7}$$

4.2 Linearizing The Model

Consider a point $z_0 = [x_0\ y_0\ V_0\ \theta_0]^T$ where the value of the control input is $U_0 = [0\ 0]^T$. If we linearize the model about this point using the first order Taylor Series expansion, we get the following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{V} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V \\ V_0\theta \\ U_1 \\ (V/L)U_2 \end{bmatrix}$$

Clearly, this can be expressed as a linear state space model as done in the following equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{V} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & V0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ V \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & V_0/L \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{8}$$

Let $z = [x\ y\ V\ \theta]^T$. The above equation can be succinctly expressed as:

$$\begin{aligned} \dot{z} &= Az + Bu \\ A &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & V_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ B &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & V_0/L \end{bmatrix} \end{aligned} \tag{9}$$

The system (A,B) is controllable. Also note that at we have access to all measurements, and hence, the matrix C is identity.

5 Controlling The System

Full state feedback and LQR controllers were used to control the system. However, both of them were based on the assumption that the system trajectory was close to the point of linearization. When this condition was violated, the controllers failed and needed to be modified. With appropriate modifications, the controllers started working again.

5.1 Error Dynamics

Previously, we saw that when only the x and y co-ordinates are given as functions of time, it is possible to determine the full state trajectory as well as the reference inputs as functions of time. Let $x_{ref}(t)$ and $y_{ref}(t)$ be the reference co-ordinates. We obtain $V(t)$, $\theta(t)$, $U_{ref1}(t)$, $U_{ref1}(t)$ from (6) and (7) respectively. Using (8), we can write the evolution of the reference state trajectory as follows:

$$\begin{bmatrix} \dot{x_{ref}} \\ \dot{y_{ref}} \\ \dot{V_{ref}} \\ \dot{\theta_{ref}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & V0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{ref} \\ y_{ref} \\ V_{ref} \\ \theta_{ref} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & V_0/L \end{bmatrix} \begin{bmatrix} U_{ref1} \\ U_{ref2} \end{bmatrix}$$

Hereafter, we shall use $z = [x\ y\ V\ \theta]^T$ to refer to the states. Thus, we can rewrite the above equation as:

$$\dot{z}_{ref} = Az_{ref} + Bu_{ref} \tag{10}$$

We define $e = z_{ref} - z$ to be the error. From (10) and (9), it is easy to see that,

$$\dot{e} = Ae + B\Delta U \tag{11}$$

Where $\Delta U = U_{ref} - U$. If we define $\Delta U = -Ke$ such that $(A - BK)$ has eigenvalues in the left half of the s plane,we will have an exponentially decaying error. Thus, we have:

$$U = U_{ref} + Ke \tag{12}$$

5.2 Full state Feedback

By simply placing the closed loop eigenvalues in the left half plane, it is possible to synthesize a controller gain matrix K. We placed the eigenvalues at $\{-2+j, -2-j, -20, -21.5\}$] so as to have 2 dominant closed loop poles, to elicit an approximate second order response from the fourth order system.However, the linear model fails to track the trajectory as soon as it deviates considerably from the point of linearization.

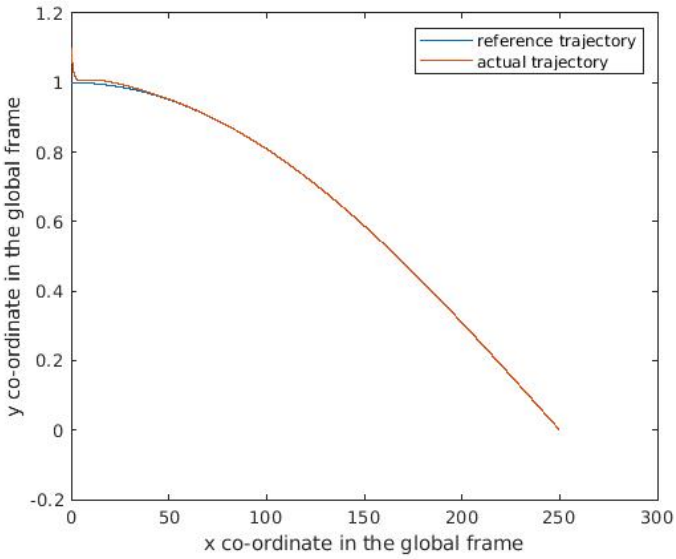


Figure 13: Path Tracking of a sinusoid trajectory using Full State Feedback

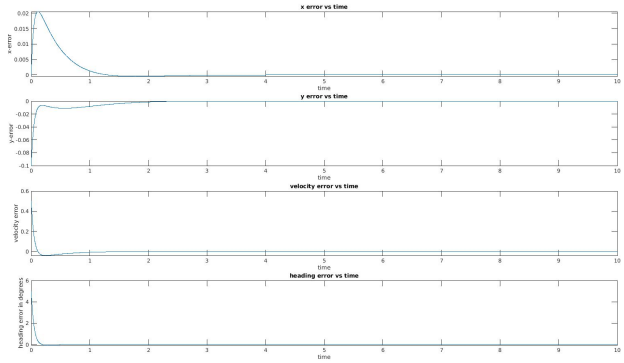


Figure 14: Errors while implementing Full State Feedback

5.3 LQR

Along with a state feedback controller, we also implemented an LQR controller. The process is similar to the former, except that the gain matrix K is replaced by the new gain matrix G, calculated using the system matrices A and B. The state weight matrix and the input weight matrix were taken to be appropriately sized identity matrices respectively. This control mechanism also performed poorly when it went away from the linearization conditions.

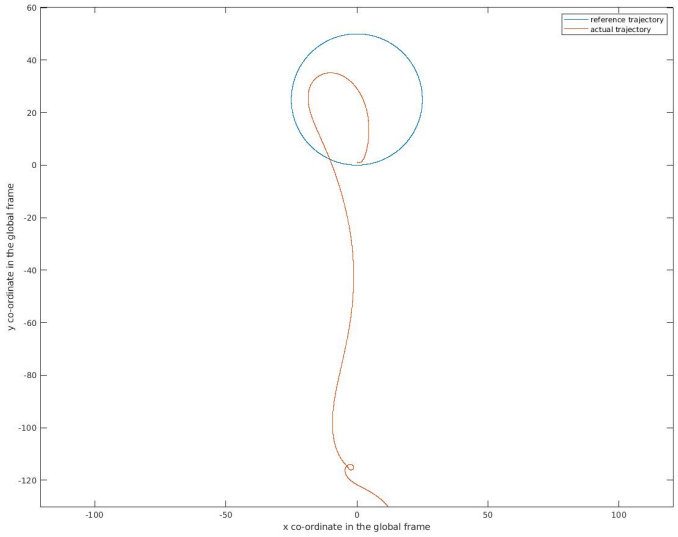


Figure 15: Linear FSF failing to track a circular trajectory

5.4 Constraining The Control Input

We applied constraints to the control input and observed that the trajectory was still being tracked, albeit slightly slowly.

5.5 Extending The Controller to Nonlinear Zones

Previously, we described a linearized model around a fixed point, with zero y-velocity and heading. Herein, we will generalize the linearization to about an arbitrary point $z_t = [x_t \ y_t \ V_t \ \theta_t]$. The control input is taken to be zero. Upon linearization, the following model is obtained :

$$\begin{aligned} \dot{e} &= Ae + Bu \\ A &= \begin{bmatrix} 0 & 0 & cos(\theta_t) & -V_t sin(\theta_t) \\ 0 & 0 & sin(\theta_t) & V_t cos(\theta_t) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ B &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & V_t/L \end{bmatrix} \\ U_t &= U_{ref} + Ke \end{aligned} \tag{13}$$

The matrices A and B are, thus, recomputed at each instant, and the gain matrix K is recomputed by using either FSF or LQR (we are using LQR). This method works on even circular trajectories, where the linear controllers fail. However, since it involves computing the model parameters at each instant, it is slightly more computationally expensive (especially if you use LQR).

5.6 Using a Fixed Gain Controller for Non Linear Control

In the previous method, we linearized the model at each instant to obtain new system matrices each time. In other words, we 'altered our model' to 'keep up' with the deviations from the linearization conditions. Here, we alter our errors instead! First, we define the trajectory frame of reference. At time t, let the reference

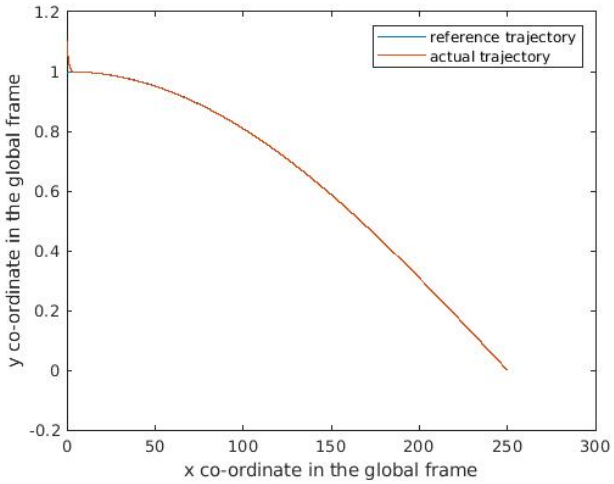


Figure 16: LQR tracking a sinusoid

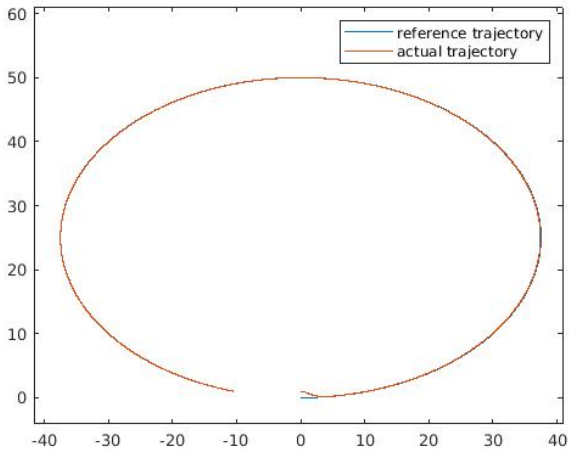


Figure 17: Nonlinear Adaptation of an LQR tracking an ellipse

trajectory lie at point E. The axes x_t and y_t are determined by the shape of the trajectory at E. The error vector is calculated in the body frame, which is used by the controller just as in section 5.2. The calculation is done as follows:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$
$$e = R^T(z_{ref} - z)$$

As evident from the plots below, the controller works with fixed gains for any continuous trajectory in the 2D plane. This also does not increase computational cost, as the error computation is very fast, involving only a matrix multiplication.

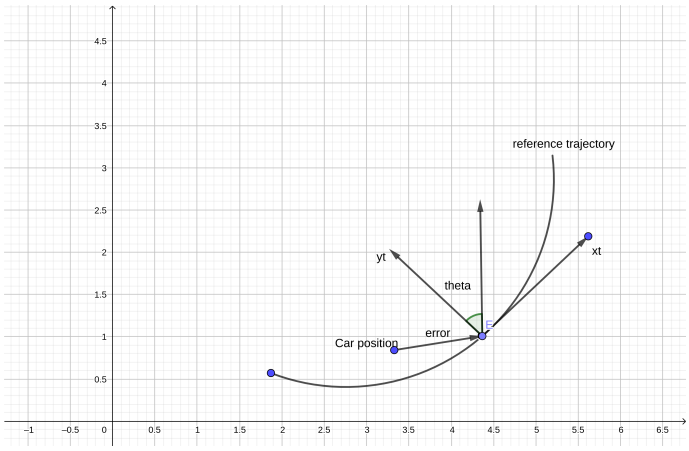


Figure 18: At each instant, the error is expressed in terms of axes x_t and y_t

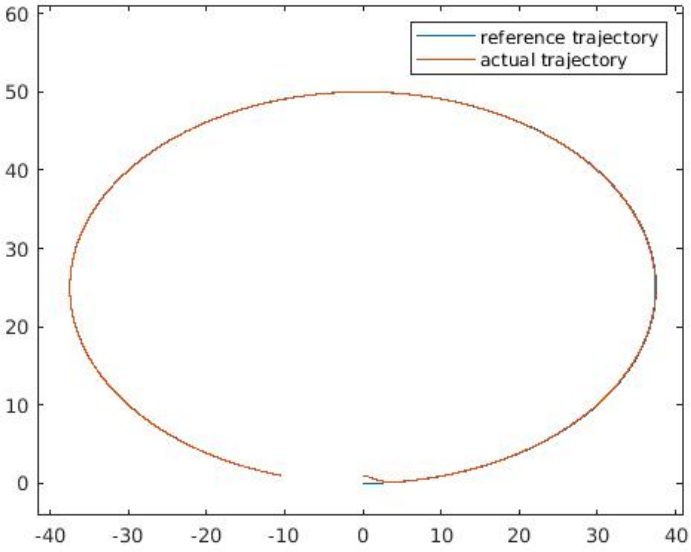


Figure 19: Fixed Gain LQR used as a non-linear controller to track an ellipse

Note: It is rather interesting to note the similarities between the structure of the matrices in (13) and (14).

6 Conclusions and Future Work

6.1 In the second order model

On the second order model, a FSF controller, a PID controller, a MPC controller and a Pure Pursuit controller were applied. The system was linearized about the equilibrium point $(0, 0)$ and the performance and the disturbance rejection properties of the controllers were compared. We observed the following -

- The FSF and PID controllers track the trajectory accurately in the absence of any disturbances, but the accuracy decreases drastically when noise is added to the system.
- The MPC controller is able to track the trajectory well in the presence and absence of disturbances.

6.2 In the 4^{th} order model

On the 4^{th} order system, a fixed gain linear controller was applied. However, the controller failed when it deviated from its point of linearization. To account for that, we used a non linear control, where the gains were recomputed at each instant. That proved to be computationally expensive in the case of LQR control. To solve this problem, we recomputed the errors instead of the gains. The controller still works. However, there is still a lot of room for improvement. Some suggestions are listed:

- The heading angle is given by the sensor and it's value is limited to [0,2π). Thus, there is a discontinuity that is unaccounted for in the model and needs to be addressed. There could be two ways of doing this. First, figure out a way to 'reset' **both** the reference and the vehicle heading. Second, instead of having θ, include both $sin(\theta)$ and $cos(\theta)$ in the state space. Thus the discontinuity completely disappears, and we could proceed similarly to the work done here.
- The reference trajectory may not always start near the vehicle. In such cases, all controllers presented in the model fail. To mitigate this, it might be suitable to interpolate an *interim reference trajectory* that starts close to the system and eventually coalesces with the actual reference trajectory.
- Some measurements might not be available, and might need to be estimated. Also, noise has not been dealt with in this work. Measurement as well as system noise could be incorporated in the simulation, and a state estimator could be implemented.

7 Appendix

7.1 PID Controller

The proportional gain produces an output value that is proportional to the current error value. The integral term accelerates the movement of the process towards a reference and eliminates the residual steady-state error that occurs with a pure proportional controller. Derivative action predicts system behavior and thus improves settling time and stability of the system. These gains are calculated manually, requiring some trial and error, or by using loop tuning methods such as Ziegler-Nichols tuning or by using tuning software.

7.2 Model Predictive Control

Model predictive control (MPC) is an advanced method of process control that is used to control a process while satisfying a set of constraints. MPC is based on iterative, finite-horizon optimization of a plant model. At time t the current plant state is sampled and a cost minimizing control strategy is computed (via a numerical minimization algorithm) for a relatively short time horizon in the future: $[t, t+T]$. An online calculation is used to explore state trajectories that emanate from the current state and find a cost-minimizing control strategy until a time $t+T$. Only the first step of the control strategy is implemented, then the plant state is sampled again and the calculations are repeated starting from the new current state, yielding a new control and new predicted state path. The prediction horizon keeps being shifted forward and for this reason MPC is also called receding horizon control. Although this approach is not optimal, in practice it has given very good results. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly, thus differing from Linear-Quadratic Regulator (LQR). Also MPC has the ability to anticipate future events and can take control actions accordingly. PID controllers do not have this predictive ability.

An example of a non linear cost function for optimization is given by -

$$J = \sum_{i=1}^{N} w_{xi}(r_i - x_i)^2 + \sum_{i=1}^{N} w_{ui}\Delta u_i^2 \tag{15}$$

where -
x_i = controlled variable (measured variable)
r_i = reference variable

u_i = manipulated variable
w_{xi} = weighing coefficient for x_i
w_{ui} = weighing coefficient for u_i

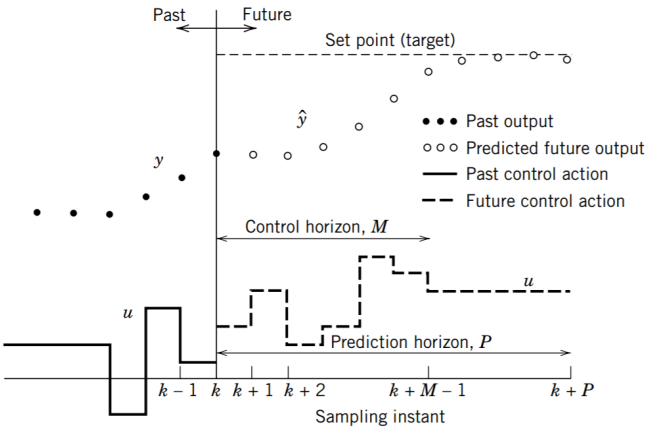


Figure 20: Conceptual diagram for Model Predictive Control

The main differences between MPC and LQR are that LQR optimizes in a fixed time window (horizon) whereas MPC optimizes in a receding time window, and that a new solution is computed often whereas LQR uses the single (optimal) solution for the whole time horizon[**5**]. Therefore, MPC typically solves the optimization problem in smaller time windows than the whole horizon and hence may obtain a suboptimal solution. However because MPC makes no assumptions about linearity, it can handle hard constraints as well as migration of a nonlinear system away from its linearized operating point, both of which are downsides of LQR.

8 Individual Contributions

Member	Contribution
Laukik Mujumdar	Sections 4-6
Gautam Sharma, Yash Mandlik	Sections 3.1-3.3
Gautam Sharma	Section 3.4
Chaitanya Kulkarni	State Observer Analysis

References

[1] Model Predictive Control, "https://en.wikipedia.org/wiki/Model_predictive_control"

[2] PID controller, "https://en.wikipedia.org/wiki/PID_controller"

[3] Full State feedback, "https://en.wikipedia.org/wiki/Full_state_feedback"

[4] A Tutorial on Autonomous Vehicle Steering Controller Design, Simulation and Implementation, "https://arxiv.org/pdf/1803.03758.pdf"

[5] Model Predictive Control, "http://folk.ntnu.no/skoge/vgprosessregulering/papers-pensum/seborg-c20ModelPredictiveControl.pdf"

[6] Automatic Steering Methods for Autonomous Automobile Path Tracking, "https://www.ri.cmu.edu/pub_files/2009/2/Automatic_Steering_Methods_for_Autonomous_Automobile_Path_Tracking.pdf"