SC 626

SYSTEMS AND CONTROL ENGINEERING LABORATORY

# INTRODUCTION TO ROBOTICS SYSTEM TOOLBOX AND KINEMATICS MODELS

Amey Samrat Waghmare
203230013
Systems and Control Engineering,
Indian Institute of Engineering, Bombay

January 20, 2021

# AIM

To study the Robotics System Toolbox of MATLAB and to study the Bicycle Kinematic modelling of a car like robot.

# ROBOTICS SYSTEM TOOLBOX

The Robotics System Toolbox of MATLAB provides tools and algorithms for designing and simulations of Mobile Robots. Algorithms for mapping, path planning, path following and motion control are available. They can be used as a function in MATLAB Script file as well as few of them are available as Simulink Blocks. These functions are explained below.

### 0.0.1 Mapping

For simulation of the robot, a map is required where the robot can travel and we can analyze its behaviour. We can use the toolbox's predefined maps or we can create our own map. We can load the predefined maps using the command *load exampleMaps.mat* .One of the available map is shown in Figure (1).

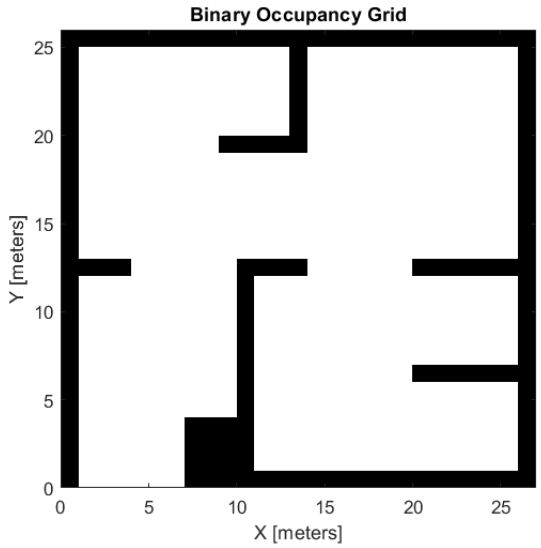


**Figure 1:** simpleMap available in Robotics System Toolbox

Following are the functions available pertaining to Mapping,

1. *binaryOccupancyMap*
   This command is used to create Occupancy Grid. In other words, this command defines

the map for our robot to operate in.

2. *getOccupancy*
   It is used to check if a particular location is a free space or if there are obstacles present at that location. It returns 0 for free space and 1 for occupied space.

3. *inflate*
   It inflates each occupied position on the map by a radius specified. This is done to indicate the dimension of the obstacle present at that location.

Other functions like *move* and *occupancyMatrix* are also available to select a smaller region of the defined map and to store the map as a matrix respectively.

### 0.0.2 Path Planning and Following

Now that we have defined the map in the previous section, we can define the start and end points on this map and create a controller that will make our robot to go from the start location to the end location. Following are the functions available to do the same,

1. *mobileRobotPRM*
   Here,PRM stands for probabilistic road map. *mobileRobotPRM* is used to create a roadmap, which consists a set of possible paths in the map our robot can traverse through taking into account the free and occupied spaces. These set of paths are network graphs whose number of nodes and distance between the nodes can be set as desired to fit the complexity of the map and to create an obstacle free path from start to end location.

2. *findpath*
   Once a roadmap (prm) is defined, the command *findpath* finds an obstacle free path for our robot to follow. It returns a matrix which contains the path in x,y locations to be followed. This is also refered to as waypoints.

3. *controllerPurePursuit*
   This object creates a Controller which is used by a differential drive robot to follow the waypoints. Given the current position of our robot, the controller computes the linear and angular velocity of the robot. This function needs to be called successively in order to traverse the path from start location to end location. One of the property/attribute

of this controller is the *lookAheadDistance*. This is the local goal of our robot. The angular Velocity of the robot is calculated based on this look ahead distance. Varying this attribute have a significant effect on the performance of the algorithm. A larger value results in smoother trajectory of the robot, but can cause robot to cut corners along the path. A lower value can result in oscillations over the trajectory.

### 0.0.3 Kinematic Models

The Robotics System Toolbox also consists of some robot models. We can use any one of the available model for the simulation of our robot. Following are the Kinematics model available to use,

- Unicycle Kinematics Model
- Differential Drive Kinematics Model
- Bicycle Kinematics Model
- Ackermann Kinematics Model

Few of the above kinematics model are explained in upcomming sections.

### 0.0.4 Simulink Blocks

As stated earlier, the Robotics System Toolbox comes with a few Simulink blocks for implementation of the Kinematics of the robot. Following Blocks are available,

1. Pure Pursuit Controller
2. Bicycle Kinematics Model
3. Ackermann Kinematics Model
4. Differential Drive Kinematics Model
5. Unicycle Kinematics Model

# KINEMATICS MODEL

In order for the robot to move, we must apply some forces to this robot. **Dynamics** is the study of motion in which these forces are modelled. **Kinematics** is the study of mathematics of motion without considering the forces that affect the motion. It is based on the geometric relationship of system. Various Kinematics model were stated earlier and few of them are discussed below.

### 0.0.5 Differential Drive Kinematics Model

Many mobile robots use Differential Drive mechanism. It consists of two drive wheels mounted on an common axle. Each of the wheel can be independently driven either in forward or backward direction. When both the wheels turn at the same speed in the same direction, the robot moves straight in that direction. When one wheel turns faster than the other, the robot turns in an arc toward the slower wheel. Figure (2) shows the Differential Drive Model of robot. When the robot is moving in a curve, there is a centre of curve at that moment called as Instantaneous Centre of Curvature.

Here, the control variables are the linear velocities of the right wheel and left wheel. Consider

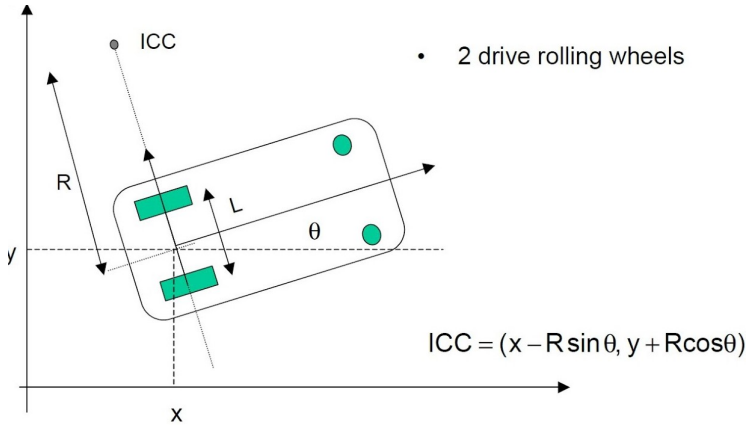


**Figure 2:** Differential Drive Kinematics

the following terminologies,

- R - Instantaneous Curvature Radius of the robot trajectory.
- L - Distance between Left wheel and Right wheel
- $v_r(t)$ - Linear Velocity of Right Wheel.

- $v_l(t)$ - Linear Velocity of Left Wheel.
- $\omega(t)$ Angular velocity of the robot.
- $R - \frac{L}{2}$ - Curvature radius of trajectory of Left Wheel
- $R + \frac{L}{2}$ - Curvature radius of trajectory of Right Wheel

Hence, we can write,

$$\omega(t) = \frac{v_r(t)}{R - \frac{L}{2}}$$

$$\omega(t) = \frac{v_l(t)}{R + \frac{L}{2}}$$

From above two equations,

$$\omega(t) = \frac{v_r(t) - v_l(t)}{L}$$

$$R = \frac{L}{2}\frac{v_r(t) + v_l(t)}{v_r(t) - v_l(t)}$$

and we know that $v(t) = R\omega(t)$, hence

$$v(t) = \frac{1}{2}(v_r(t) + v_l(t))$$

The above equations govern the motion of the robot, where we can take the linear velocities of the Left wheel and Right wheel as an input and correspondingly we can obtain the Angular Velocity and Instantaneous Radius of curvature of robot.

### 0.0.6 Bicycle Kinematics Model

This kinematics model is used to represent a car like model using two wheels. Two front wheels are represented by a single front wheel. Similarly, two rear wheels are represented by a single rear wheel. This model is also called as front wheel steering model.
The Bicycle model is used to model the longitudinal (x), lateral (y) and yaw ($\psi$) motion of the robot. Here, we assume that the robot is lumped mass acting on the centre of mass and we also assume that there is no slip in the longitudinal or lateral direction. Figure (3) shows the Bicycle Kinematics Model.
Consider the following terminologies,

- O - Instantaneous Centre of Rotation.
- R - Circular Radius

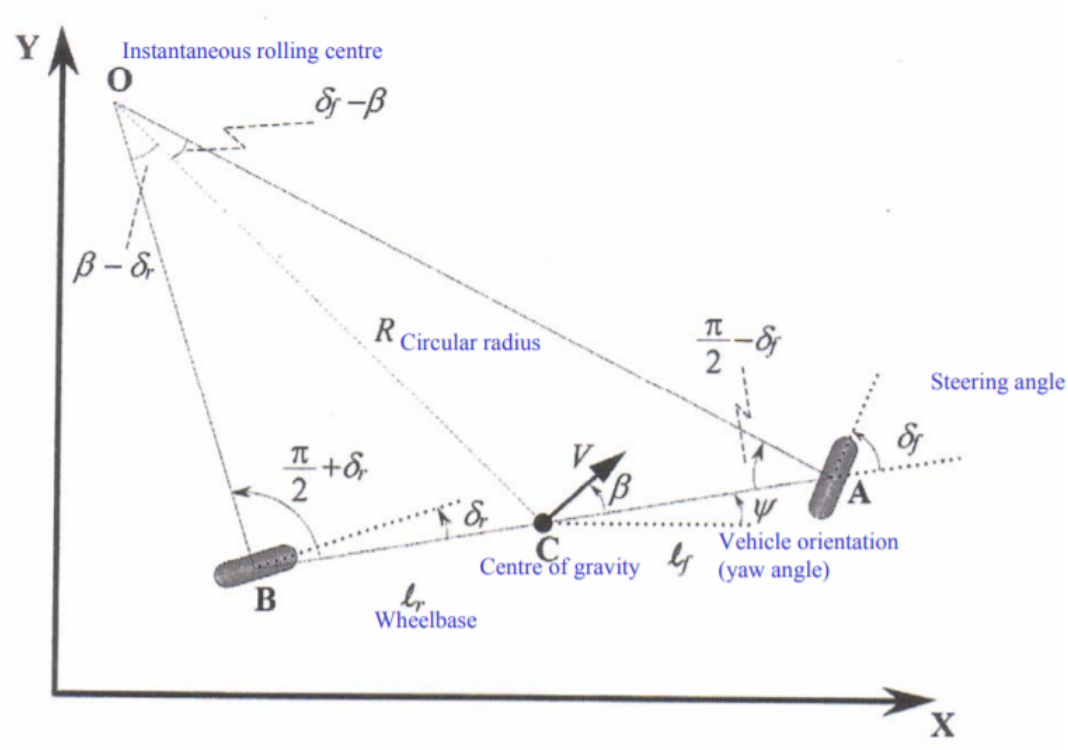


**Figure 3:** Bicycle Kinematics Model

- C - Centre of Gravity
- $l_f$ and $l_r$ are distances of front and rear wheel respectively from C
- v - Vehicle Velocity
- $\delta_f$ - Front wheel Steering angle.
- $\beta$ - Side Slip angle of the Vehicle.

Assuming constant velocity v, from Figure (3), we can write

$$\dot{x}(t) = vcos(\psi(t) + \beta)$$

$$\dot{y}(t) = vsin(\psi(t) + \beta)$$

$$\dot{\psi}(t) = \omega = \frac{v}{R}$$

and from the geometry,

$$\tan(\delta_f) = \frac{l_f + l_r}{\tilde{R}}$$

$$\tan(\beta) = \frac{l_r}{\tilde{R}} = \frac{l_r}{l_f + l_r} tan(\delta_f)$$

$$cos(\beta) = \frac{\tilde{R}}{R}$$

From above equations, we get the following equations that define the kinematics of the robot,

$$\dot{x}(t) = v cos(\psi(t) + \beta) \tag{1}$$

$$\dot{y}(t) = v sin(\psi(t) + \beta) \tag{2}$$

$$\dot{\psi}(t) = \frac{v}{l_f + l_r} tan(\delta_f) cos(\beta) \tag{3}$$

where $\beta = tan^{-1}\left(\frac{l_r}{l_f + l_r} tan(\delta_f)\right)$

This model is available as a Simulink Block in MATLAB. The input to the robot is the vehicle speed, angular velocity and the steering angle. The output is the position of the vehicle.

### 0.0.7 Ackermann Steering Kinematic Model

Although the Differential Drive model are very popular, they suffer from few drawbacks such as its reliance on caster wheel which performs poorly on high speeds. One more drawback is that to drive in a straight line, it requires that both the wheels to rotate at the exact same speed. These drawbacks are mitigated by Ackermann steering model which is driven by single motor and can steer the front wheels. The model is shown in Figure (4). Here, all wheels of the car drive on circles with a common center point, avoiding skid. Consider the following

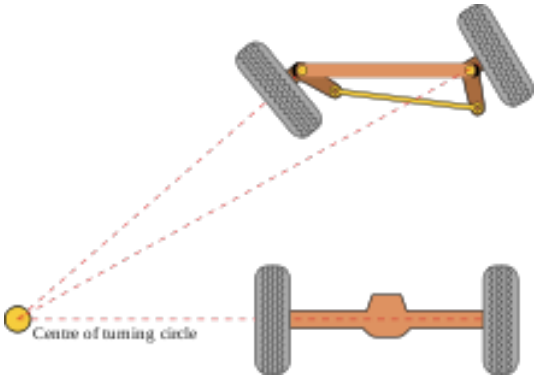


**Figure 4:** Ackermann Steering Kinematics Model

terminologies,

- L - Distance between front and rear axle
- O - Centre of turning circle
- R - Distance between O and Car's longitudinal centre line
- $\phi$ - Steering angle
- $\theta$ - Algle of robot with respect to global frame
- $x_r, y_r, \theta r$ - robot coordinates
- $x_w, y_w, \theta w$ - world or global coordinates
- $\phi_l$ - Angle of front left wheel
- $\phi_r$ - Angle of front right wheel
- l - distance between two front wheels

Then we can write,

$$\dot{x_w} = \dot{x_r} cos(\theta_w)$$

$$\dot{y_w} = \dot{y_r} sin(\theta_w)$$

$$\dot{\theta_w} = \dot{x_r} \frac{tan(\phi)}{L}$$

and as all the wheels of the car drive on circles with common centre point,

$$tan\left(\frac{\pi}{2} - \phi_r\right) = \frac{L}{R - \frac{l}{2}}$$

$$tan\left(\frac{\pi}{2} - \phi_l\right) = \frac{L}{R + \frac{l}{2}}$$

These equations give the angles of the Ackermann Steering model.
The Ackermann Steering Model is also available in Robotics Systems Toolbox as a function and a Simulink Block.

## CONCLUSION

Various functions and algorithms available in the Robotics System Toolbox were studied. Mathematical Kinematics models like Differential Drive, Bicycle and Ackermann's Steering model were developed.

## REFERENCES

1. Robotics System Toolbox, MATLAB User manual.
2. Robust Model Predictive Control for Autonomous Vehicle/Self-Driving Cars
3. Mobile Robot Kinematics
4. Lateral Vehicle Dynamics, Modelling of Autonomous Systems School of Engineering and Informatics University of Sussex Brighton.
5. Advanced Robotics 2: Forward Kinematics of Car-Like Mechanisms

SC 626

SYSTEMS AND CONTROL ENGINEERING LABORATORY

# SIMULATION OF BICYCLE KINEMATICS MODEL USING SIMULINK

Amey Samrat Waghmare
203230013
Systems and Control Engineering,
Indian Institute of Engineering, Bombay

January 27, 2021

# AIM

Simulation of Bicycle Kinematics Model using Simulink to perform following tasks,

- Robot follows straight line path on x-axis and y-axis
- Robot follows straight line path diagonally through origin
- Robot follows a circular path

# BICYCLE KINEMATICS MODEL

The Bicycle kinematics model is used to represent a car like model using two wheels. Two front wheels are represented by a single front wheel. Similarly, two rear wheels are represented by a single rear wheel. This model is also called as front wheel steering model.
The states of the vehicle are x, y positions and the vehicle heading angle $\theta$.The angle $\psi$ is the steering angle. Figure (1) shows the simple Bicycle Kinematics Model.
The equations of Bicycle kinematic model are

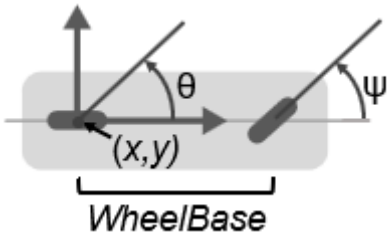


**Figure 1:** Bicycle Kinematics

$$\dot{x}(t) = v cos(\theta(t))$$

$$\dot{y}(t) = v sin(\theta(t))$$

$$\dot{\theta}(t) = \omega = \frac{v}{R}$$

Where, $tan(\psi) = \frac{L}{R}$, L is the Wheelbase and R is the radius of curvature. Hence, the last equation becomes,

$$\dot{\theta} = \frac{v}{L} tan(\psi)$$

The above equations define the kinematics of the robot.

This model is available as a Simulink Block in MATLAB. The input to the robot is the vehicle Linear Speed (v) and Angular Velocity ($\omega$). The output is the position of the vehicle. The MATLAB block is shown in Figure (2).

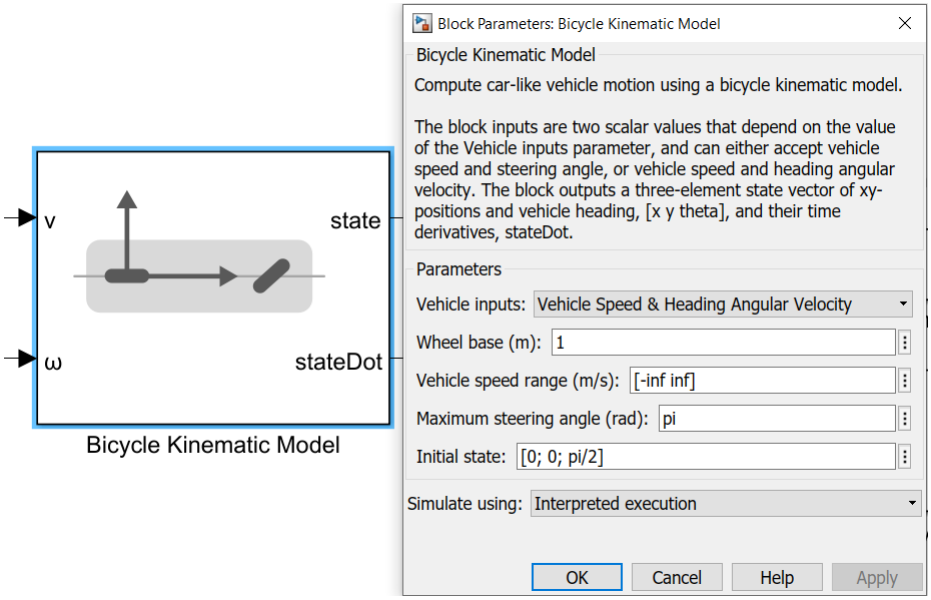


**Figure 2:** Bicycle Simulink Block with parameters

We can see from the Figure (2) that the inputs to this block are Vehicle Speed and Heading Angular Velocity. We can set the Wheel base (L), vechicle speed range and maximum steering angle ($\psi$) as desired through the Block Parameters. Here, we can also specify the initial position of our robot by modifying the Initial state.

## 0.1 Simulink Model

The simulink model for the simulation of vehicle is shown in Figure (3). The Bicycle Kinematics Model Block is explained in previous section. The remaining blocks used in the simulation model are explained below.

### 0.1.1 Block: Constant

This block outputs the constant as specified by the constant value parameter. We have used this block to specify the Vehicle Linear velocity and Angular Velocity as an input to the Bicycle Kinematics Model block.

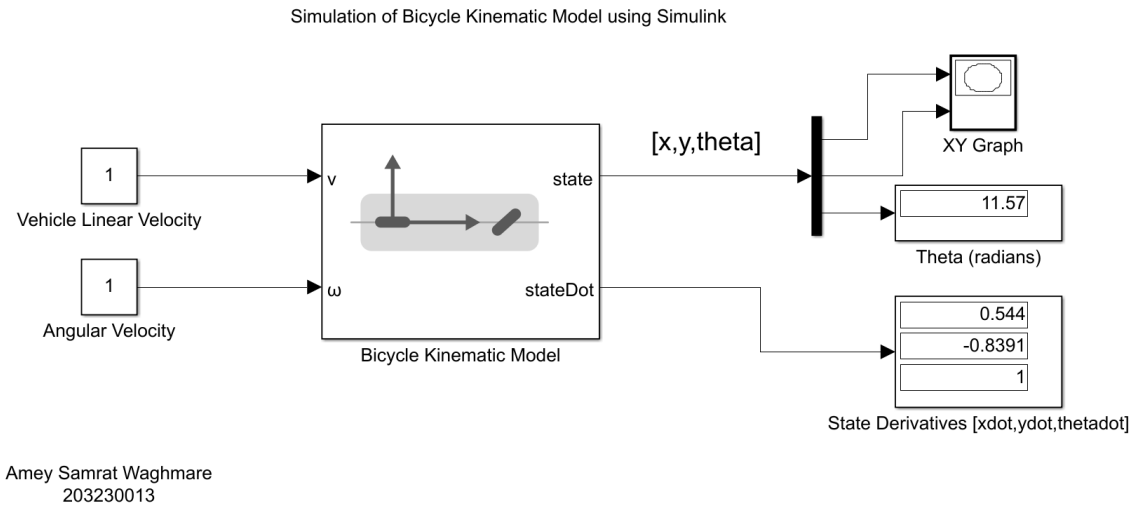


**Figure 3:** Simulink Model

### 0.1.2 Block: Demux

The Demux block splits the input vector signals into scalars. The Bicycle Kinematic Model Block returns the state of the vehicle x,y and $\theta$ in a vector form. Hence we use this block to seperate out these signals.

### 0.1.3 Block: XY Scope

This block takes two inputs X and Y and plots Y with respect to X at each time step of simulation to create a XY plot. We can specify the X and Y limits as parameters for the graph, and any data outside these limits are ignored. We have used this block for Visualizing the path taken by our vehicle by plotting the x and y position of our vehicle.

### 0.1.4 Block: Display

This block is used to display the signal as values. We have used this block to Display the Heading angle $\theta$ and the statedot vector output from Bicycle kinematics model.

## 0.2 SIMULATIONS

The Simulink Model shown in Figure (3) is simulated to achieve certain tasks and the results obtained are explained below,

- **Robot following straight line path on x-axis and y-axis:**
  To make our robot follow a Straight Line path, we must ensure that the Angular velocity must be zero at all the times so that the robot does not take any turn while travelling.We start our robot from Origin and initial heading angle $\theta$ set to 0. We set Linear velocity to 0.5 m/s and Angular velocity to 0 rad/sec so as to make the robot travel straight line path on x-axis. The simulation is run for 10 sec, so that the robot is expected to stop at location x=5 and y=0 after simulation. The obtained result is shown in Figure (4) and we see that the robot travels in a straight line and stops at the location as expected.

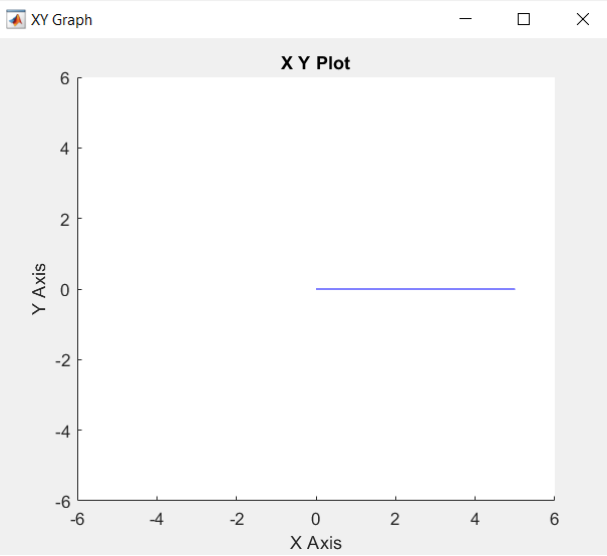


**Figure 4:** Straight Line path on x-axis

Similarly, to make our robot follow straight line path along y-axis, our robot must face in that direction. To achieve this, we change the initial condition heading angle to $90^o$, and give linear velocity of 0.5 m/s and keep angular velocity to zero so that robot follows straight line path. This simulation is shown in Figure (5), and the obtained results are as expected.

- **Robot following straight line path diagonally through origin:**
  To make our robot follow straight line path diagonally and passing through origin, we need that our robot have a heading of $45^o$. Once again we change the initial heading to

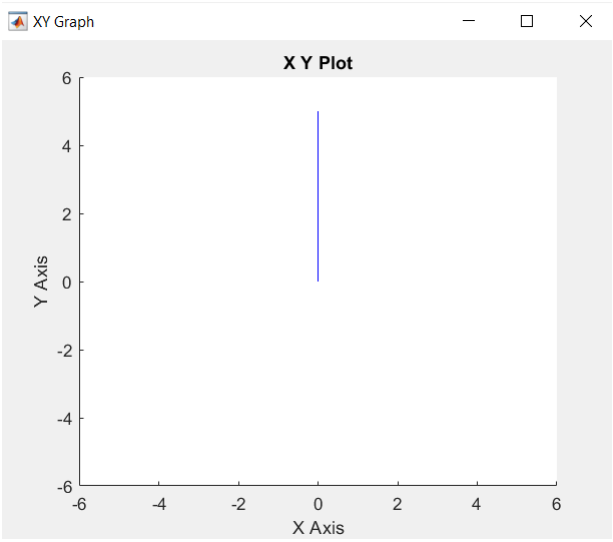


**Figure 5:** Straight Line path along y-axis

$45^o$ and keep the angular velocity to 0. Figure (6) shows the simulation result. We see that the robot starts at origin and passes through points (2,2) and (4,4) meaning that the robot is travelling diagonally and passing through origin as desired.

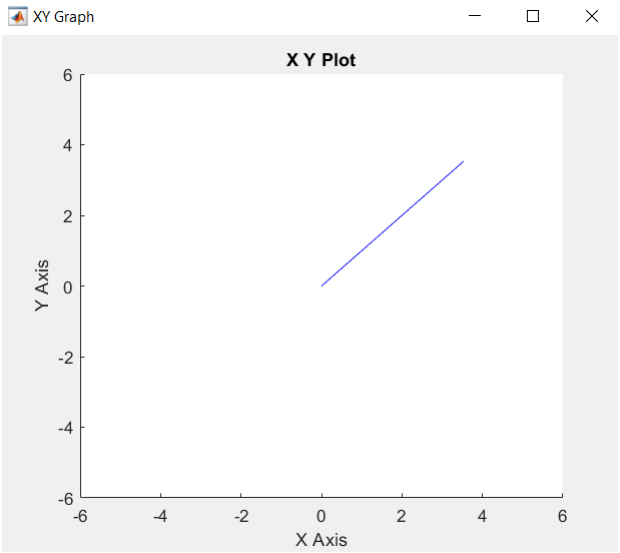


**Figure 6:** Robot path diagonal and passing through origin

- **Robot following a Circular path:**
  To make our robot follow a circular path, it is important that our robot rotates or takes a turn in order to be in a circular path. Hence we must apply the Angular velocity input along with the Linear velocity. We know the relationship $v = r\omega$. So for a given

constant Linear velocity and the desired circular path radius we can determine the Angular velocity so that the robot follows the circular path of desired radius r.
We set Linear Velocity $v = 0.5m/s$ and let our desired radius be $r = 1m$, then the corresponding Angular velocity comes out to be $\omega = 0.5rad/sec$. So our robot should start from origin, travel in a circular path of radius 2m and then return back to the origin. The center of the circular path should be (0,1). After simulation, the obtained result is shown in Figure (7). We can clearly see that the obtained result is as desired by us as discussed earlier.

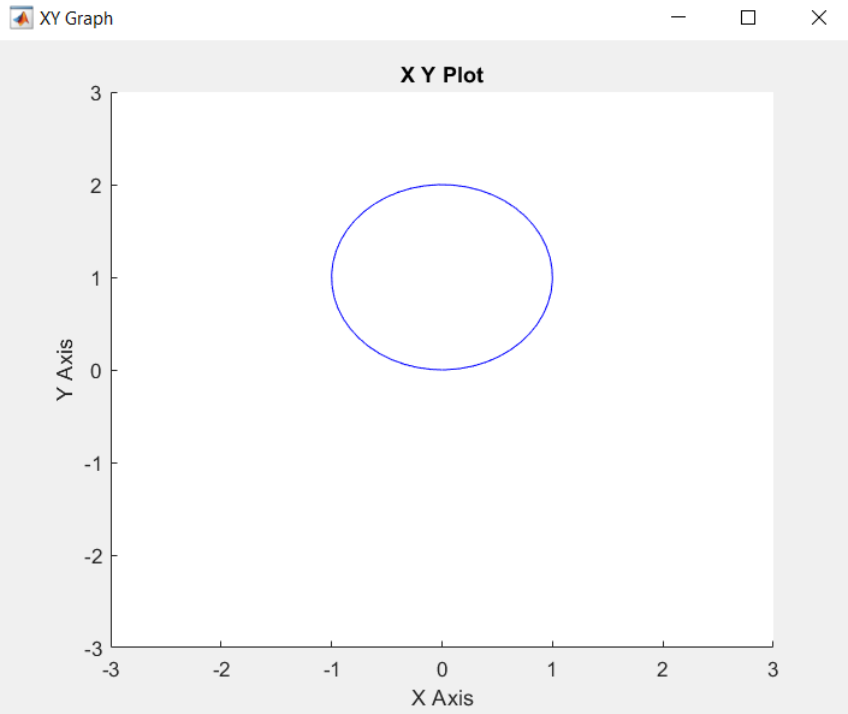


**Figure 7:** Robot travelling in Circular path of radius 1m

Now suppose we want that our robot follows a circular path of radius $r = 1.5m/s$ and have a linear velocity of $v = 2m/s$. Then the angular velocity comes out to be $\omega = 1.33rad/sec$. Suppose we also want that our robot starts from location (2,3), so we also change the initial conditions of x and y to 2 and 3 respectively. The obtained Circular path will then have its center at (2,4.5). Figure (8) shows the corresponding simulation result. We can see that as desired, the robot starts from (2,3) point and returns back following a circular path. We also note that the radius of this circular path is $r = 1.5m$ as desired. This circular path is centered at (2,4.5) as expected.

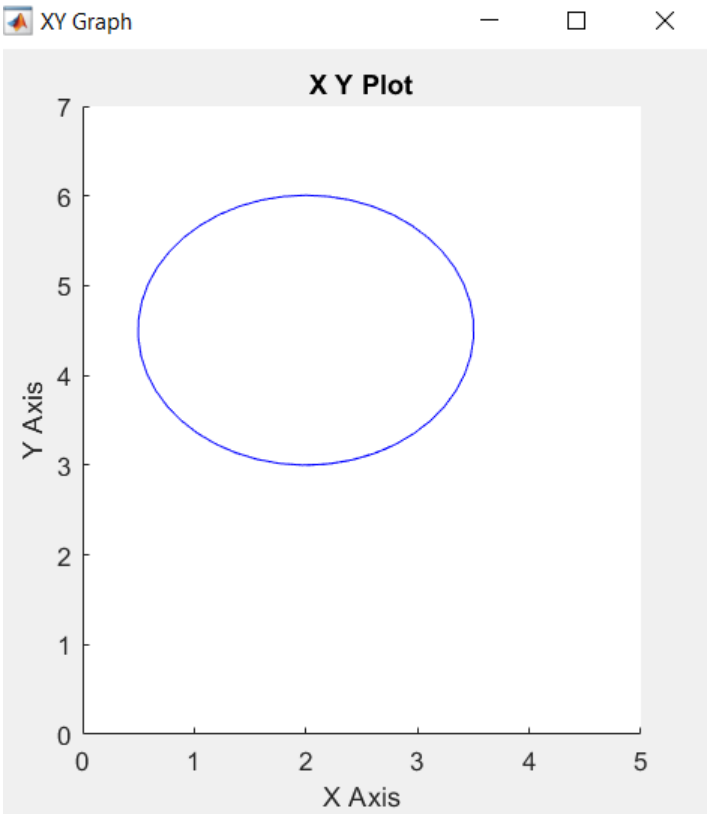


**Figure 8:** Robot following Circular Path of radius 1.5m

## CONCLUSION

The Bicycle Kinematics model was studied. To simulate this model in Simulink, Bicycle Kinematics Model block from Robotics System Toolbox was used. Correspondingly the robot was simulated to follow Straight line and Circular trajectory and it was observed that it followed the desired trajectory.

SC 626

SYSTEMS AND CONTROL ENGINEERING LABORATORY

# PATH TRACKING FOR BICYCLE KINEMATICS MODEL

Amey Samrat Waghmare
203230013
Systems and Control Engineering,
Indian Institute of Engineering, Bombay

March 10, 2021

## AIM

To simulate the Bicycle Kinematics Model in closed loop for path tracking.

## LISSAJOUS CURVES

Lissajous Curves are formed by combining two mutually perpendicular simple harmonic motions. They are the graph of a system of parametric equations,

$$x = m\ sin(at)$$

$$y = n\ sin(bt + \delta)$$

Various curves like Circle, Ellipse, Parabola, etc. can be generated by varying the parameters m,n,a,b and $\delta$. These curves can be used as a **reference** for our robot to follow. Figure (1) shows few Lissajous Figure.

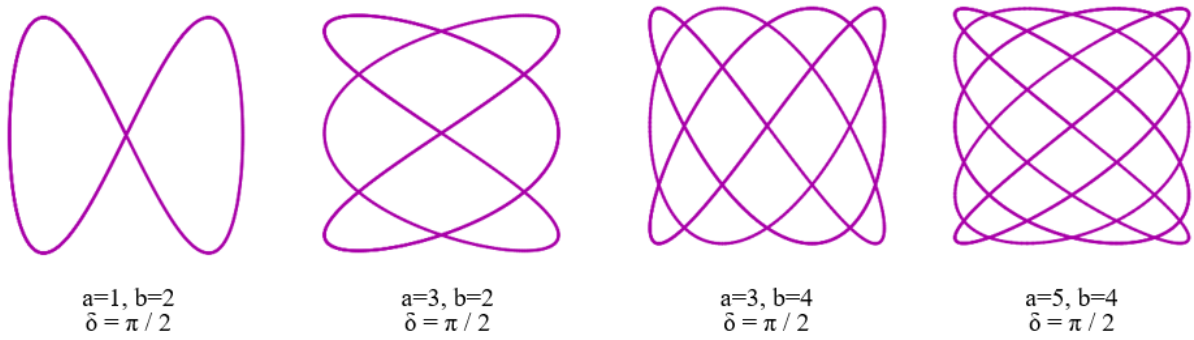


**Figure 1:** Lissajous Figure

To generate ellipse pattern, we can set the parameters as $m \neq n$ and $\delta = \frac{\pi}{2}$. Another way is to set $m = n$ and set $\delta = \frac{\pi}{4}$, which will basically skew the axes of ellipse. These Lissajous curves are shown in Figure (2).
Note:- Frequency ratio $\frac{a}{b}$ plays an important role while generating Lissajous curves. When this ratio is 1, curve is elliptical as shown in above figure. When this ratio is 2, we get parabolic curves. Other ratios produce more complicated curves.

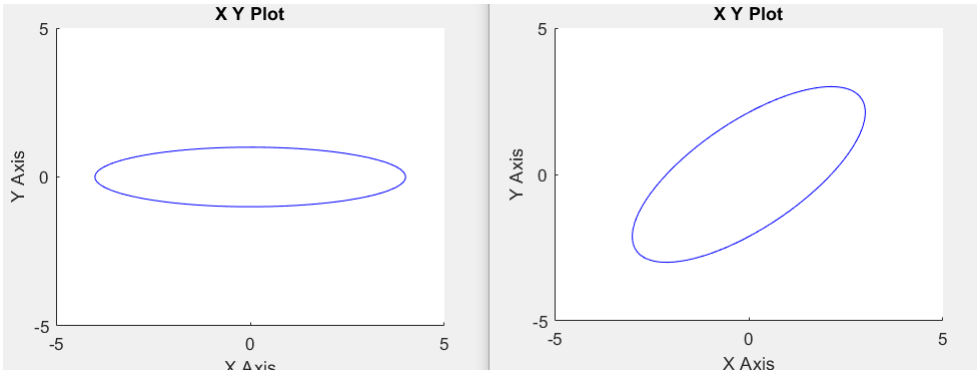


**Figure 2:** Lissajous Curves a) m=4,n=1, $\delta = \frac{\pi}{2}$ , b) m=n=3, $\delta = \frac{\pi}{4}$

# SIMULATIONS

### 0.0.1 Simulink Model

The robot is modelled in Bicycle Kinematics Model and is supposed to track a path generated by Lissajous curve. We will use Elliptical path as reference for our robot to follow. To ensure that the robot tracks this path, we have used P and PI Controllers. The Simulink Model is shown in Figure (3).

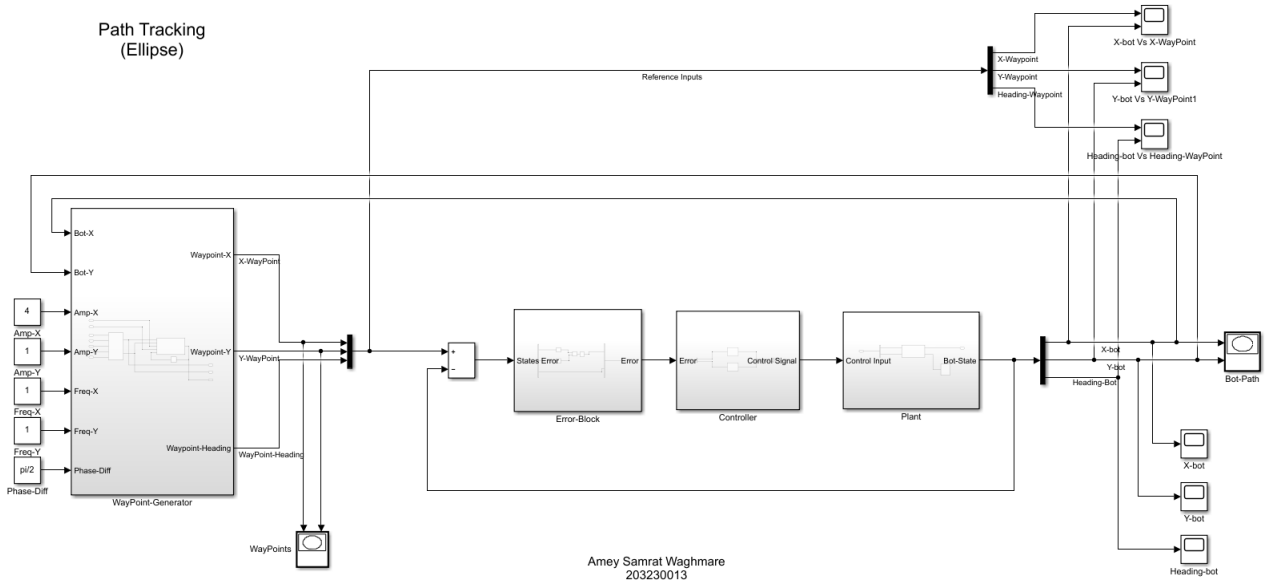


**Figure 3:** Simulink Model

To generate the Lissajous curve, a MATLAB Function block is used and their parameters are set to generate the curves as shown in Figure (2). This gives the x and y coordinate set-points for our robot to follow.

The position error is calculated as the Euclidean distance between the present position of the robot and the desired setpoint. The Linear Velocity input of the Bicycle model is then fed with this error, so that farther the robot is from setpoint, higher will be the error.
The direction in which the robot should face is determined from the setpoint coordinates and the actual robot position. The robot should rotate in this direction and move forward for path tracking. Hence, this direction is compared with actual $\theta$ of the robot and then fed as error to the Steering angle of the robot.

### 0.0.2 Controllers

To ensure that the robot follows the path determined by setpoint, P and PI Controllers are implemented. These Controllers are implemented using Gain and Integrator blocks of Simulink. Two Controllers are implemented which are used to control the Linear Velocity and Steering Angle of the robot. The Control inputs generated by these controllers are fed to the Bicycle Kinematics model. The Gains are analytically chosen to make the robot follow the desired path.

### 0.0.3 Simulation Results

It is Desired that our robot follows the elliptical setpoints shown in Figure (2), and the parameters to generate them are also indicated there. The Bicycle Model's Wheelbase is set to 0.1m and Maximum Steering Angle is $\frac{\pi}{2.5}$radians. The robot starts at origin in XY Plane, with heading angle set to 0.

- Proportional Controller:
  The Proportional Controller Gains for Linear Velocity and Steering angle are set to 15 and 2 respectively. Figure (4) shows the path followed by robot for this configuration. The x,y and heading $\theta$ of the robot which are the states of the system are compared with desired values and are shown in Figure (5). We see that although the robot starts from origin (which is not on the ellipse), it slowly follows the desired trajectory. All the states also closely follow the desired waypoints.

- PI Controller:
  The Controller gains for Velocity Control are set as P = 20, I = 0.5 and that for Steering

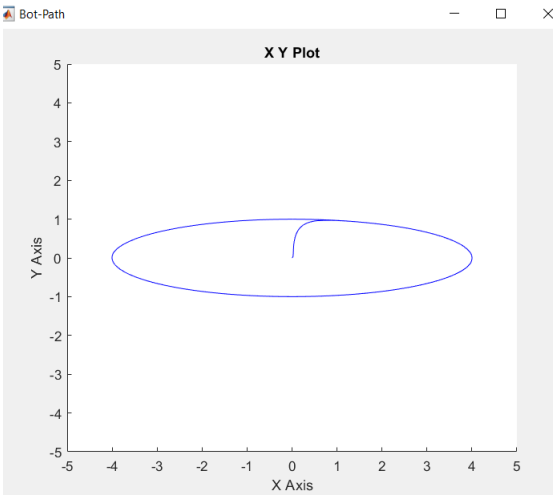


**Figure 4:** Robot Path Tracking for Proportional Controller

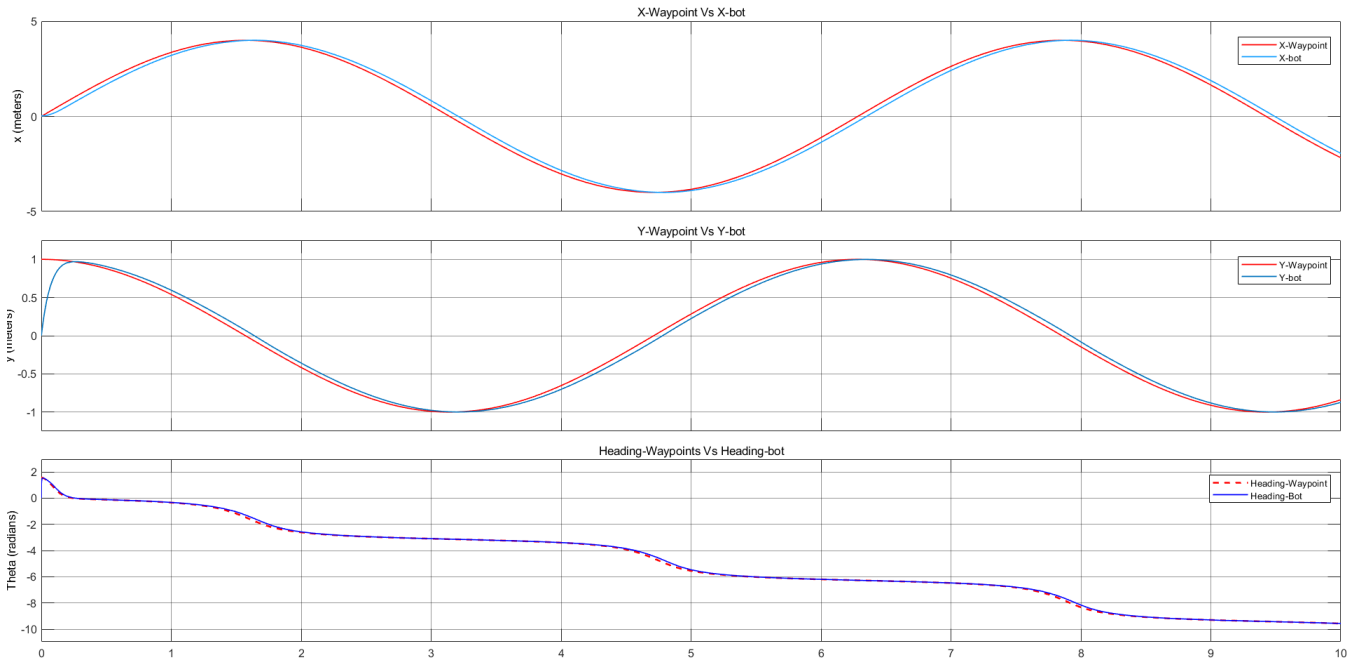


**Figure 5:** States of the robot with P Controller

Angle Controller are set as P = 4, I = 2. The path tracked by the robot is shown in Figure (6). It follows the elliptical path. The comparison of states is shown in Figure (7).

The comparison in terms of Mean Squared error between the waypoints and the actual path followed by robot is shown in the table below.

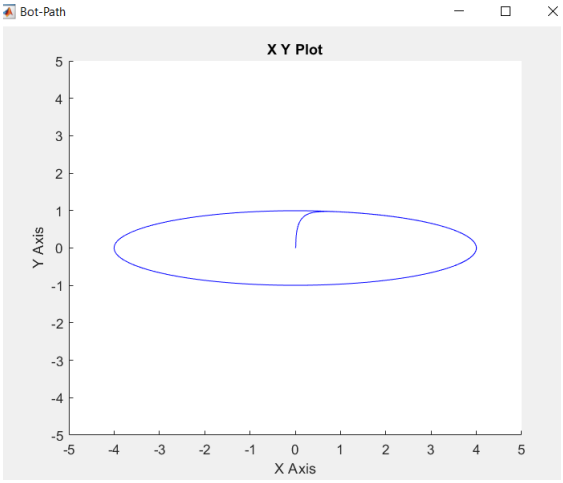


**Figure 6:** Robot Path Tracking for PI Controller

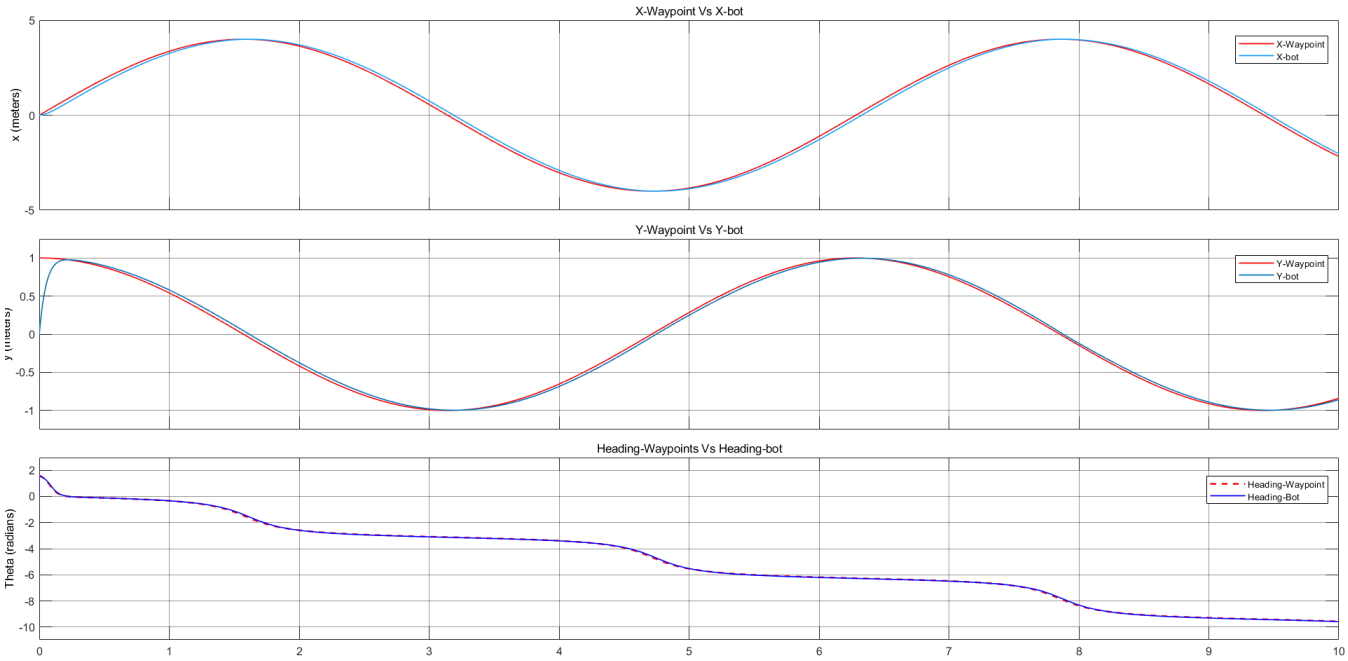


**Figure 7:** States of the robot with PI Controller

| Comparison for Elliptical Path | | |
|---|---|---|
| - | P Controller ($P_1$,$P_2$) = (15,2) | PI Controller ($P_1$,$I_1$) = (20,0.5), ($P_2$,$I_2$) = (4,2) |
| $MSE_X$ | 0.1892 | 0.1339 |
| $MSE_Y$ | 0.2848 | 0.2231 |
| $MSE_\theta$ | 0.2801 | 0.1990 |

The simulations for the skewed elliptical curve as shown in Figure (2) were also done and the

results are explained below.

- Proportional Controller:
  The Proportional Controller Gains for Linear Velocity and Steering angle are set to 10 and 1.5 respectively. Figure (8) shows the path followed by robot for this configuration. The corresponding state trajectories are shown in Figure (9).

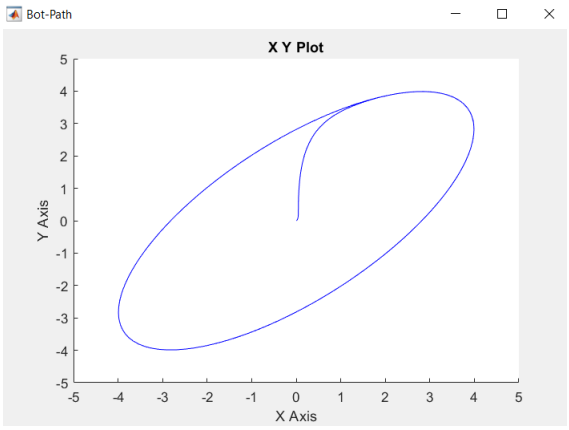


**Figure 8:** Robot Path Tracking for Proportional Controller

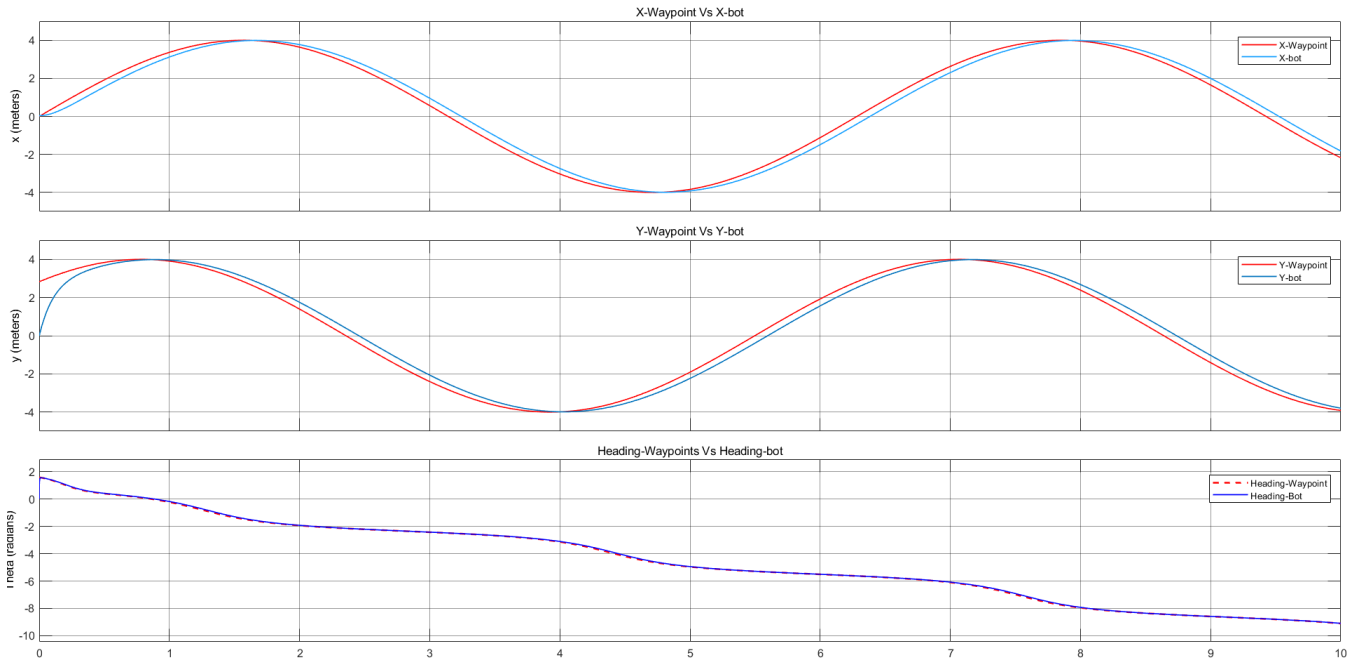


**Figure 9:** States of the robot with P Controller

- PI Controller:
  The Controller gains for Velocity Control are set as P = 10, I = 1 and that for Steering Angle Controller are set as P = 5, I = 4. The path tracked by the robot is shown in Figure (10). The corresponding state trajectories are shown in Figure (11). We see that the robot closely follows the waypoints given to it.

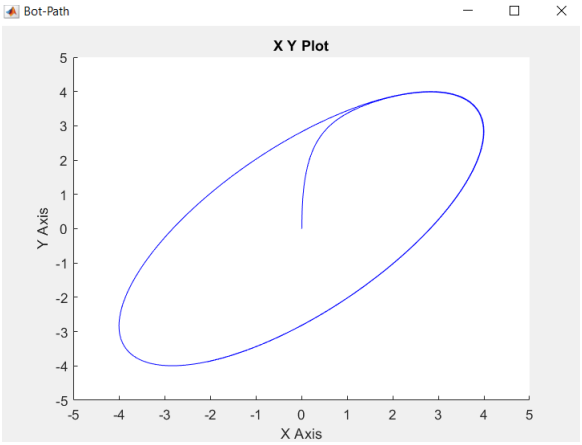


**Figure 10:** Robot Path Tracking for PI Controller

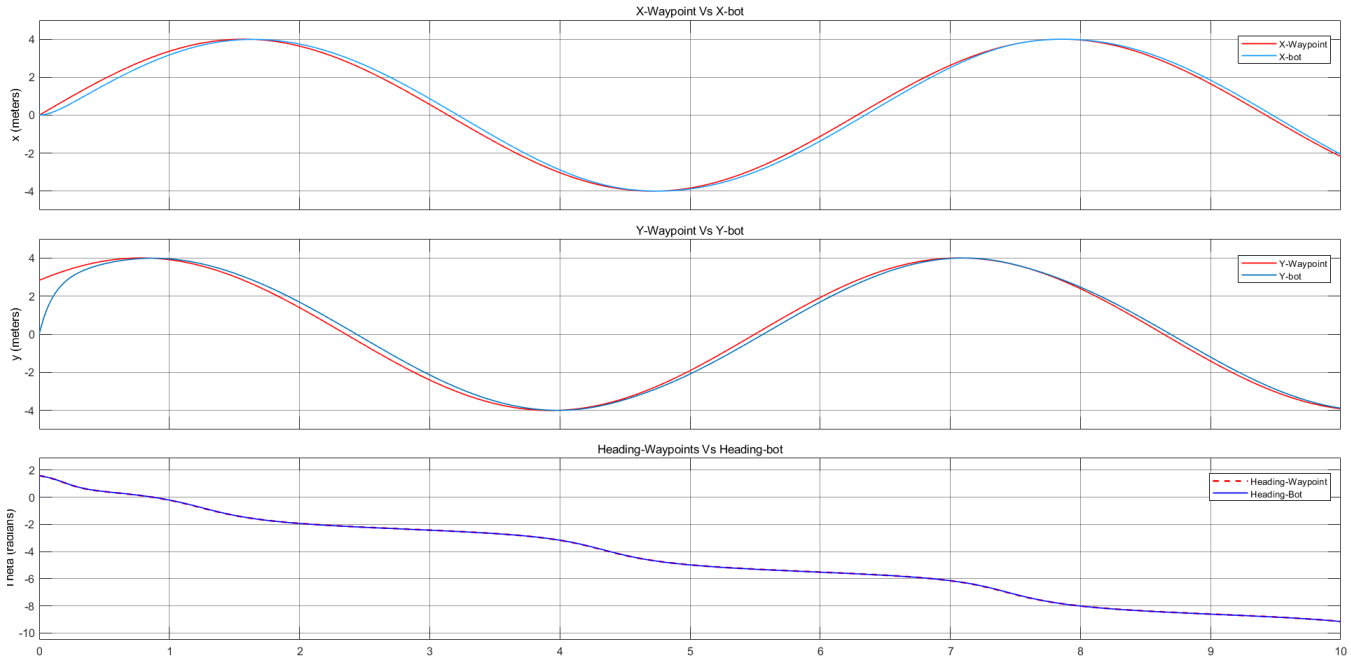


**Figure 11:** States of the robot with PI Controller

The comparison in terms of Mean Squared error between the waypoints and the actual path followed by robot is shown in the table below.

| Comparison for Skewed Elliptical Path | | |
|---|---|---|
| - | P Controller ($P_1$,$P_2$) = (10,1.5) | PI Controller ($P_1$,$I_1$) = (10,1), ($P_2$,$I_2$) = (5,4) |
| $MSE_X$ | 0.1918 | 0.1959 |
| $MSE_Y$ | 0.7887 | 0.6471 |
| $MSE_\theta$ | 0.2342 | 0.1494 |

## CONCLUSION

The Bicycle Kinematics Model was simulated in Closed loop using P and PI Controllers for tracking of elliptical path. The controllers are important as they give weightage to how quickly the robot corrects the direction to which it is facing and continues to follow the desired path. It was observed that the robot fairly follows the path and the desired trajectory is obtained in all the cases. We observe that even the proportional controllers are sufficient for controlling as we obtain very good tracking response.

SC 626

SYSTEMS AND CONTROL ENGINEERING LABORATORY

# TRACKING OF REFERENCE VEHICLE WITH SAME KINEMATICS

Amey Samrat Waghmare
203230013
Systems and Control Engineering,
Indian Institute of Engineering, Bombay

March 10, 2021

## AIM

To Track a Car like Reference vehicle using a Controlled Vehicle having same Kinematics model.

## TRACKING : KINEMATICS MODEL

In the previous week, path tracking for a reference (lissajous) path was implemented. In this week, we are interested to track a reference vehicle which has the kinematics same as our actual or controlled robot. In controls literature, this problem is associated with asymptotically stabilizing the reference trajectory. In other words, we have to determine a Feedback Control law that asymptotically stabilizes the tracking error. Consider the following models for Actual or Controlled Vehicle and Reference Vehicle with reference to Ackerman model,

Kinematics of Controlled Vehicle,

$$\dot{x} = u_1 cos(\theta)$$
$$\dot{y} = u_1 sin(\theta)$$
$$\dot{\theta} = \frac{u_1}{L} tan(\phi)$$
$$\dot{\phi} = u_2$$

Kinematics of the Reference Vehicle,

$$\dot{x_r} = u_{1r} cos(\theta_r)$$
$$\dot{y_r} = u_{1r} sin(\theta_r)$$
$$\dot{\theta_r} = \frac{u_{1r}}{L} tan(\phi_r)$$
$$\dot{\phi_r} = u_{2r}$$

We can express the error in tracking position with respect to frame of reference vehicle as,

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = \begin{bmatrix} cos(\theta_r) & sin(\theta_r) \\ -sin(\theta_r) & cos(\theta_r) \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix}$$

and $\theta_e = \theta - \theta_r$, $\phi_e = \phi - \phi_r$
Taking the time derivative of above equations, we get a new system as follows,

$$\dot{x_e} = \Big(\frac{u_{1r}}{L} tan(\phi_r)\Big) y_e + u_1 cos(\theta_e) - u_{1r}$$
$$\dot{y_e} = -\Big(\frac{u_{1r}}{L} tan(\phi_r)\Big) x_e + u_1 sin(\theta_e)$$
$$\dot{\theta_e} = \frac{u_1}{L} tan(\phi) - \frac{u_{1r}}{L} tan(\phi_r)$$
$$\dot{\phi e} = u_1 - u_{2r}$$

We transform the above system to following coordinates,

$$z_1 = x_e$$
$$z_2 = y_e$$
$$z_3 = tan(\theta_e)$$
$$z_4 = \frac{tan(\phi) - cos(\theta_e) tan(\phi_r)}{L cos^3(\theta_e)}$$

and introduce new input variables as,

$$w_1 = u_1 cos(\theta_e) - u_{1r}$$
$$w_2 = k_2 y_e + \Big(\frac{3tan(\phi)}{cos(\theta_e) - 2tan(\phi_r)}\Big) \frac{sin(\theta_e)}{Lcos^3(\theta_e)} \dot{\theta}_e - \frac{u_{2r}}{Lcos^2(\phi_r) cos^2(\theta_e)} + \frac{u_2}{Lcos^2(\phi) cos^3(\theta)}$$

Hence, our system becomes,

$$\dot{z_1} = \Big(\frac{u_{1r}}{L} tan(\phi_r)\Big) z_2 + w_1$$
$$\dot{z_2} = -\Big(\frac{u_{1r}}{L} tan(\phi_r)\Big) z_1 + u_{1r} z_3 + w_1 z_3$$
$$\dot{z_3} = -k_2 u_{1r} z_2 + u_{1r} z_4 + w_1 \Big(z_4 - k_2 z_2 + (1 + z_3^2) \frac{tan(\phi_r)}{L}\Big)$$
$$\dot{z_4} = w_2$$

The control law that makes the above system Globally Asymptotically Stable is

$$w_1 = -k_1 |u_{1r}| \Big[ z_1 + \frac{z_3}{k_2} \Big( z_4 + (1 + z_3^2) \frac{tan(\phi_r)}{L} \Big) \Big]$$
$$w_2 = -k_3 u_{1r} z_3 - k_4 |u_{1r}| z_4$$

This input makes the system globally asymptotically stable when i) $u_{1r}$ is bounded differen-

tiable function and its derivative is also bounded and doesnot tend to 0 as t tends to infinity, ii) $|\phi_r|$ is smaller than or equal to $\frac{\pi}{2}$.

In above equations, $k_1, k_2, k_3, k_4$ are the controller gains, which can be obtained by Linearizing the above system. The linearized system is of the form $\dot{\eta} = A\eta$, with $\eta = (x_e, y_e, \theta_e, \phi_e / L)^T$ and,

$$A = \begin{bmatrix} k_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -k_2 k_4 & -k_3 & -k_4 \end{bmatrix}$$

The characteristics equation is $(\lambda + k_1)(\lambda^3 + k_4\lambda^2 + k_3\lambda + k_2 k_4 \lambda) = 0$. The gain values must be selected such that the poles of the system lie in the left half of s-plane for stability.

## SIMULINK MODEL

The Simulink model for the simulation of vehicles is shown in Figure (1). Ackerman model is used for modelling both reference vehicle and controlled vehicle. Few function blocks are used which are explained below

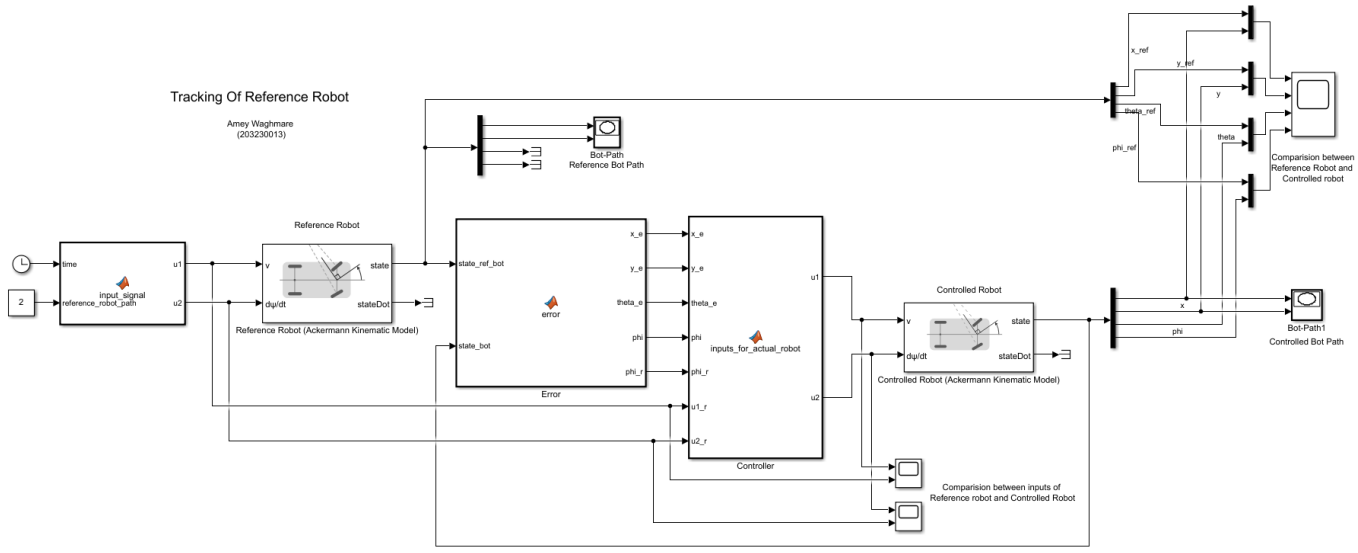


**Figure 1:** Simulink Model

### *input signal*

It is employed to generate the Linear velocity and Steering angle angular velocity for the Reference robot. Input 0 gives straight line path and Input 1 gives Circular path

### *error*

It calculates the error in state variables between Reference vehicle and Controlled vehicle as explained in previous section.

### *inputs for actual robot*

This function converts the system into z variables, determines the control law $w_1, w_2$ and then reverts back into original variables and compute the Control law $u_1$ and $u_2$ to be given to the Controlled vehicle. The controller gains $k_1, k_2, k_3, k_4$ are set in this block and can be varied as per requirements. The outputs of this block, Linear velocity and Steering angle angular velocity are directly fed to as inputs to the Controlled Vehicle.

## SIMULATIONS

Simulations are done for various paths of reference vehicle as explained previously. Depending upon the path, the controller gains are also changed. Following subsections explains the simulation results for straight line and circular path

### Straight Line path for Reference Vehicle

Here the reference vehicle follows a straight line path and the Controlled vehicle is supposed to follow the reference robot. The reference vehicle starts at origin and the controlled variable starts from initial location (-2,-3) in (x,y). The controller gains are analytically chosen as $k_1 = 1, k_2 = 1, k_3 = 1$ and $k_4 = 1$. One more simulation is done using values $k_1 = 1, k_2 = 1, k_3 = 0.5$ and $k_4 = 0.5$. Figure (2) and Figure (3) shows the XY plot of the robots and the states of the vehicles respectively. It is evident that the controller asymptotically stabilizes the error and hence the error between controlled vehicle and reference vehicle is greatly reduced. The comparison of both the controllers is shown in the table below.

| - | $k_1 = 1, k_2 = 1, k_3 = 1, k_4 = 1$ | $k_1 = 1, k_2 = 1, k_3 = 0.5, k_4 = 0.5$ |
|---|---|---|
| $MSE_X$ | 0.7078 | 0.6766 |
| $MSE_Y$ | 1.2397 | 1.2562 |
| $MSE_\theta$ | 0.2252 | 0.1882 |
| $MSE_\phi$ | 0.1570 | 0.1267 |

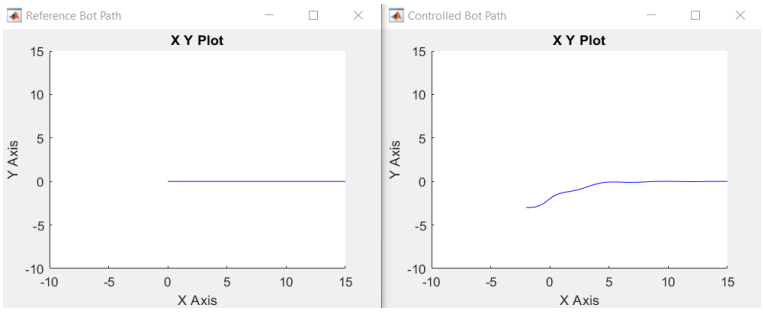


**Figure 2:** Vehicle Tracking for Straight Line Path

The comparison of the inputs to reference vehicle and controlled robot is shown in Figure(4).

## Circular path for Reference Vehicle

Here the reference vehicle follows a circular path and the Controlled vehicle is supposed to follow the reference robot. The reference vehicle starts at origin and the controlled variable starts from initial location (-2,-3) in (x,y). The controller gains are analytically chosen as $k_1 = 5, k_2 = 5, k_3 = 15$ and $k_4 = 9$. Other values are chosen as $k_1 = 5, k_2 = 2, k_3 = 10$ and $k_4 = 5$. Figure (5) and Figure (6) shows the XY plot of the robots and the states of the vehicles respectively. It is evident that the controller asymptotically stabilizes the error and hence the error between controlled vehicle and reference vehicle is greatly reduced.

| - | $k_1 = 5, k_2 = 5, k_3 = 15, k_4 = 9$ | $k_1 = 5, k_2 = 2, k_3 = 10, k_4 = 5$ |
|---|---|---|
| $MSE_X$ | 0.7100 | 0.6929 |
| $MSE_Y$ | 1.2503 | 1.2627 |
| $MSE_\theta$ | 0.3936 | 0.3276 |
| $MSE_\phi$ | 0.3044 | 0.2643 |

The comparison of the inputs to reference vehicle and controlled robot is shown in Figure(7).

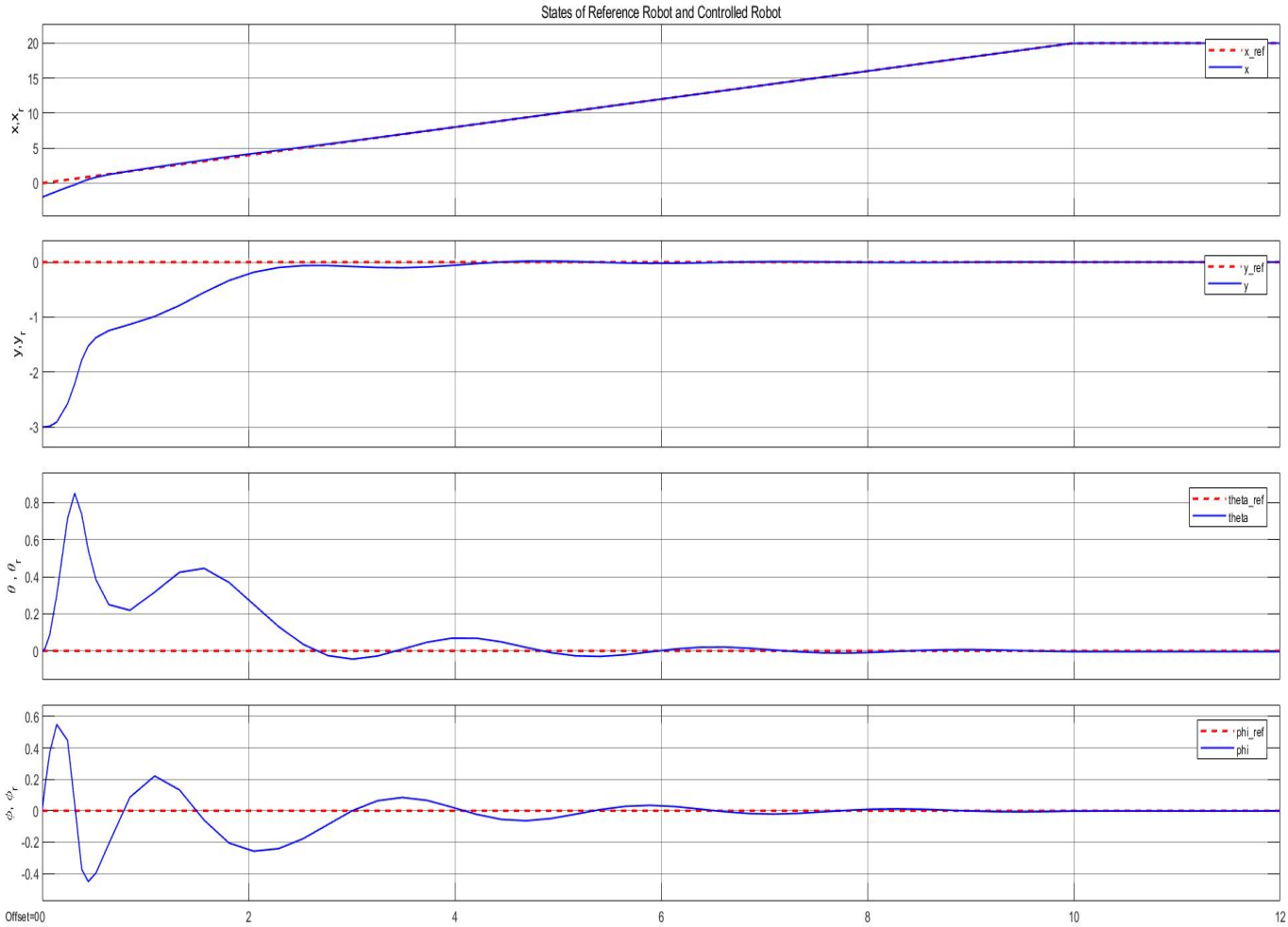


**Figure 3:** States of Reference vehicle and Controlled Vehicle for Straight Line path

This nonlinear controller is also compared with the PI controller implemented previously for Circular Path and the results are given in the table below,

| - | PI ($P_1$,$I_1$) = (4,0.5),($P_2$,$I_2$) = (2,1) | $k_1 = 5, k_2 = 2, k_3 = 10, k_4 = 5$ |
|---|---|---|
| $MSE_X$ | 0.7338 | 0.6929 |
| $MSE_Y$ | 1.6879 | 1.2627 |
| $MSE_\theta$ | 0.2075 | 0.3276 |

## OBSERVATIONS

- A feedback controller is implemented for tracking a reference vehicle.
- The nonlinear system (for error) is linearized to obtain the controller gains $k_1, k_2, k_3$ and

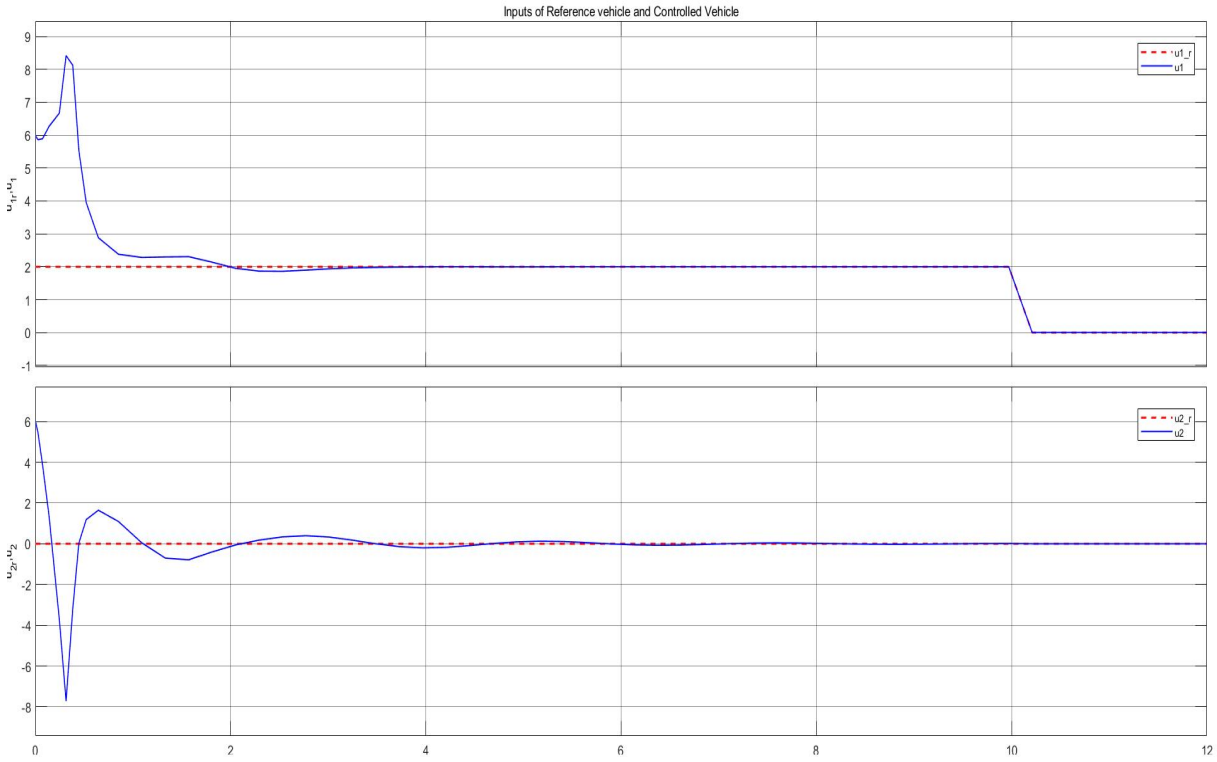


**Figure 4:** Inputs to Reference Vehicle and Controlled Vehicle for Straight Line path

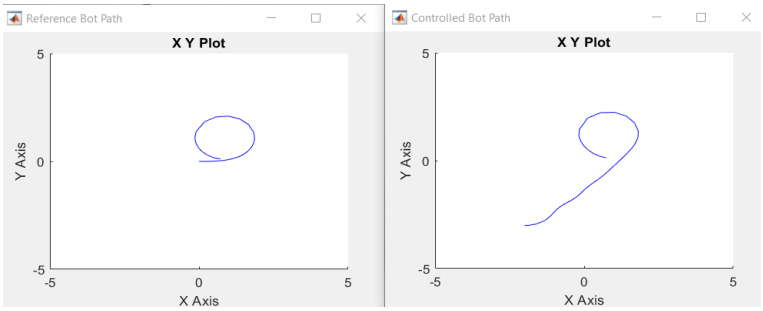


**Figure 5:** Vehicle Tracking for Circular Path

$k_4$.

- The gains are chosen such that the system is stable and for this the gain values must be positive integers so that the poles lie in the left half of s plane.
- These gains influence the error between the states of the Reference vehicle and Actual vehicle.
- Ensuring above points for the gains renders the (error) system Globally Asymptotically Stable. Hence error between reference vehicle and controlled vehicle tends to 0 as t

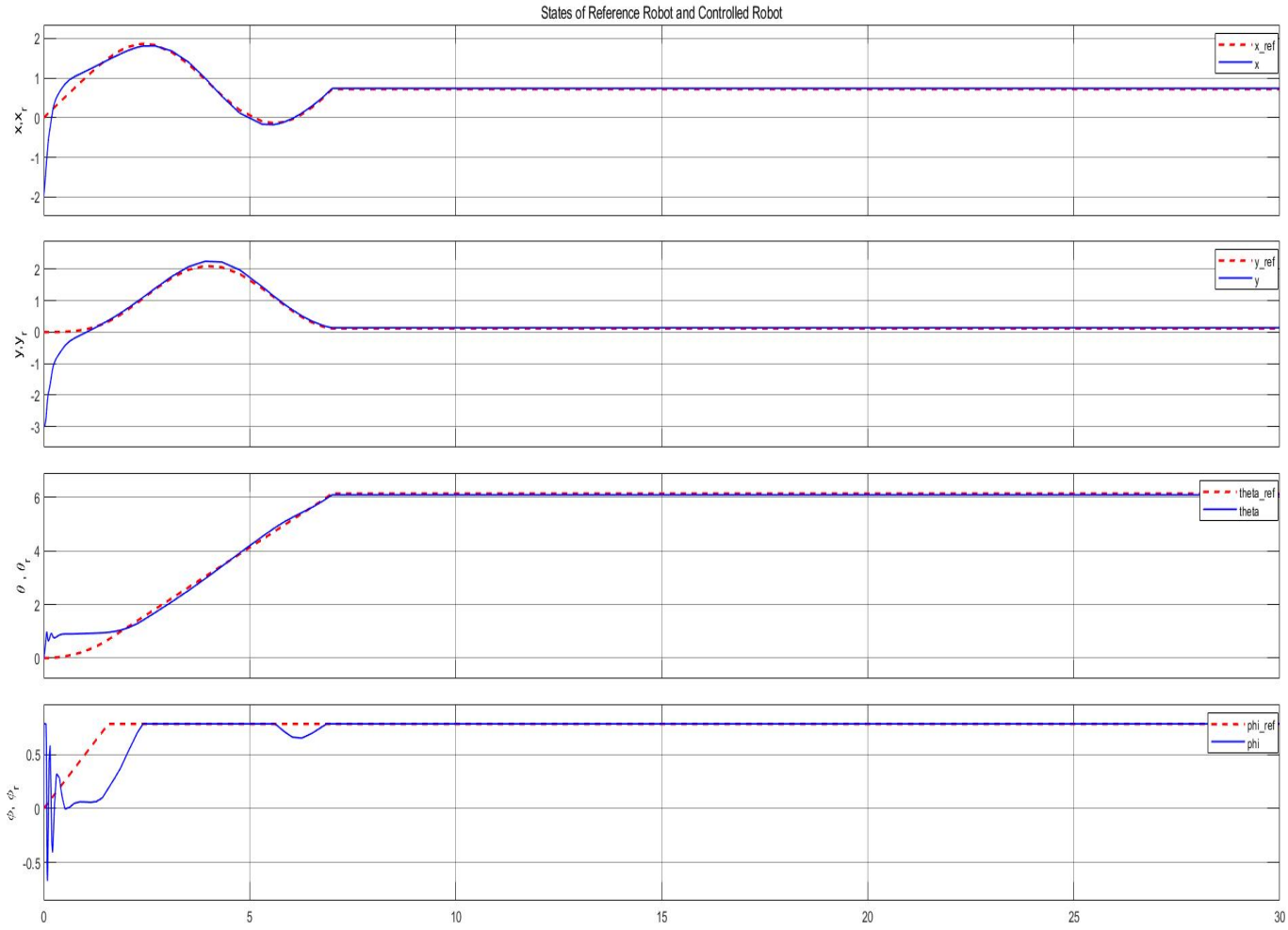


**Figure 6:** States of Reference vehicle and Controlled Vehicle for Circular path

tends to infinity.

- By simulation, it was verified that the Controlled vehicle actually tracks the reference vehicle and after some time follows the exact path followed by reference vehicle.

## CONCLUSION

A feedback controller was implemented to track a reference vehicle travelling in a straight line and circular path. The controller asymptotically stabilizes the error and hence whatever be the initial conditions of the controlled vehicle, it eventually tracks the reference vehicle.

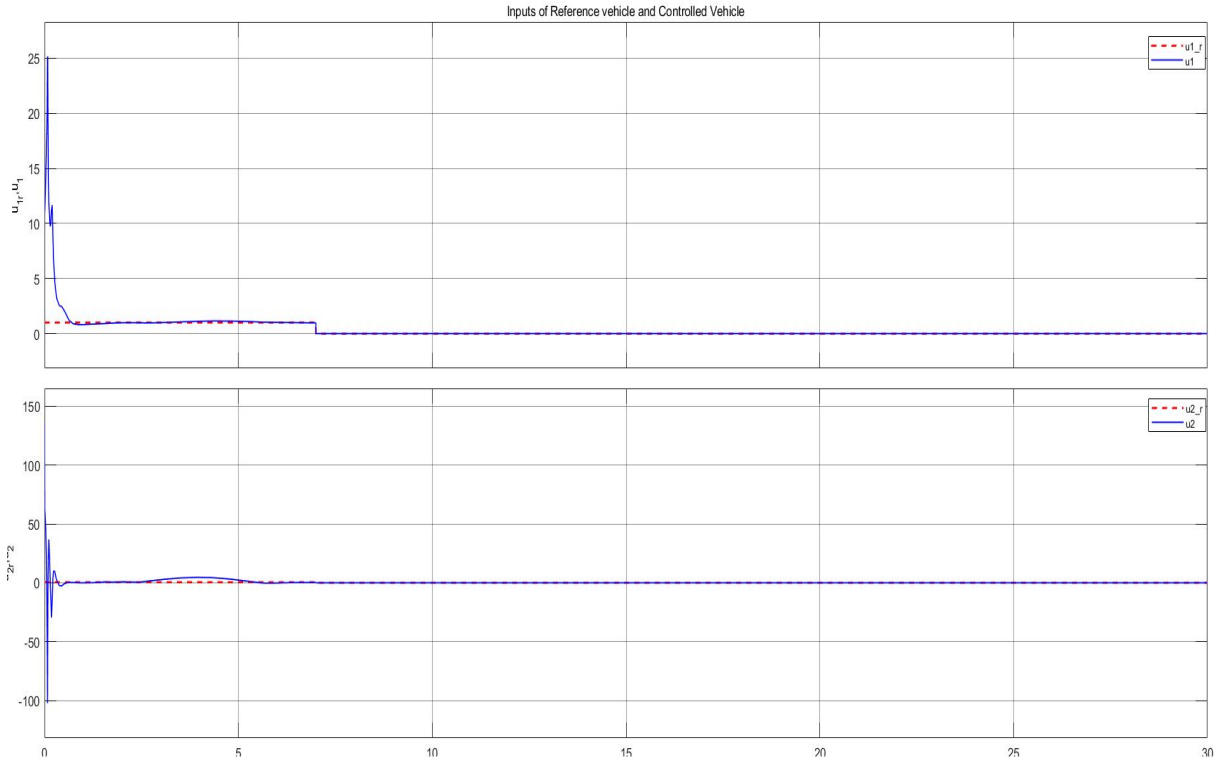


**Figure 7:** Inputs to Reference Vehicle and Controlled Vehicle for Circular path