SC 626

SYSTEMS AND CONTROL ENGINEERING LABORATORY.

Final Report

Sayan Ray

213230011

Department of Systems and Control.

Indian Institute of Engineering, Bombay

December 18, 2024

Contents

0.1 Lab 1: Intro to ROS and Differential drive robots.

In this section we are going to explore Differential Drive Robots and its working mechanism. Then we will cover **Robotic Operating System** (ROS in short), some important concepts and the python libraries which will be used code the robot in for simulation in **GAZEBO** and then we are going to have a look at the robot we are going to work on known as the Turtle-bot.

0.1.1 Aims

Therefore in short in this section we will :

- Learn about Differential Drive Robots, its Kinematics and related concepts.
- Learn about the ROS concepts and the Python Libraries that we are going to use.

0.1.2 ROS Framework

Robot Operating System or ROS is an open-source, meta-operating system for any robot.
[*A meta-operating system is an operating system built with tools, hardware and libraries for obtaining, building, writing, and running code across multiple computers.This allows different processes (nodes) to communicate with each other at run-time. This means a huge amount of functionality plus an Operating System.*]

ROS by usage of the nodes (pre-existing codes already known to function without error) these exist in the form of packages and can imported anytime.

With ROS we can implement the code in any modern programming language for our case its Python. we can code our desired robot as long it has all the referenced hardware. Paired up with the simulator known as GAZEBO a popular robot simulation software, we can even test the code prepared in ROS.

0.1.3 ROS concepts used in this Project

Here we'll get to know some Computation Graph Level concepts of ROS that we are going to use in our codes:

Nodes: Nodes are simply the processes that perform computation. So one can imagine a robot control system would usually comprise of many nodes. For example say we have a rover so there would be a node that controls the wheel motor, another node is responsible for localization then another node is responsible for path planning and so on. Now a ROS node is coded with the use of a ROS client library, such as roscpp or rospy.

Messages: So, messages can be said to be analogous to data. the standard data types are integer, floating point, boolean, and so on, apart from that arrays of these primitive data types are also supported.

Nodes exchange these messages.

Topic: Now these Messages are exchanged via some transport system using the publish and subscribe commands.

A node sends out a message by publishing it to a given topic. The command for this is **"pub"**. The topic just labels a name that will be used to identify the data/message. If a node is to import the data it will subscribe to the required and proper topic. The command for this is **"sub"**.

Now there may be multiple publishers and subscribers for a single topic also a single node may publish or subscribe to multiple topics.

0.1.4 Python Packages and Libraries used

Python has many packages and libraries that we may need and use, as navigation etc. for a robot requires such. Primarily we have two that needs mention :

The rospy library.

The Odometry Package.

The rospy is a Python library for ROS enabling Python programmers to quickly interface with ROS related services and parameters. Library rospy has been designed such that it favors development time instead run-time performance, so that algorithms can be quickly designed and tested within ROS.

The Odometry package is imported to facilitate exchange or usage of of position related information, say between two nodes. Generally we find use for these in the domains of SLAM (Simultaneous Localization And Mapping) and robot navigation.

0.1.5 Odometry

Odometry is formed from the Greek words odos (meaning "route") and metron (meaning "measure"). In other words simply we can say that Odometry is the practice of using the data from the motion sensors and using it to estimate the change in position over time. It is used by some legged or wheeled robots to estimate their position relative to a starting location. We will be using this for navigating the robot we will work upon.

0.1.6 Differential Drive Robot

A differential drive system is a two-wheeled drive system with one independent actuator(a motor) for each wheel as shown in Figure 1. In the front there may be a wheel may be a flywheel or two wheels, for balancing the robot. Now, the reason why it is called differential is because the robot motion is dictated by the difference of the direction of rotations and velocities of the two independent wheels, or to be more

precise the two actuators which can be controlled by a code written by us.
So, say if the robot wants to move straight, then both wheels turn with the same speed in the same direction (forward or reverse).
Now, for a turn towards the Right - the Right wheel turns Slower than the Left wheel
and similarly for a Left turn - the Right wheel turns Faster than the Left wheel.

And lastly if the robot is to simply rotate on its spot, Both its wheels turn at the same speed but in the opposite directions (one clockwise then other is anti-clockwise). All these have been illustrated in the Figure 2.

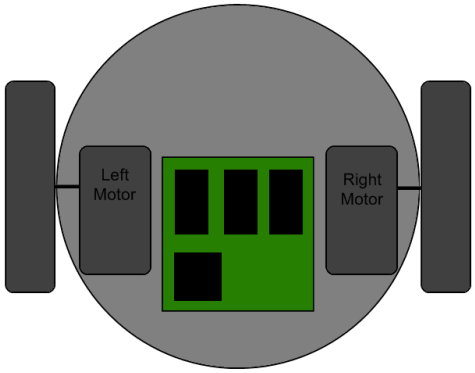


Figure 1: The Differnetial Drive Robot

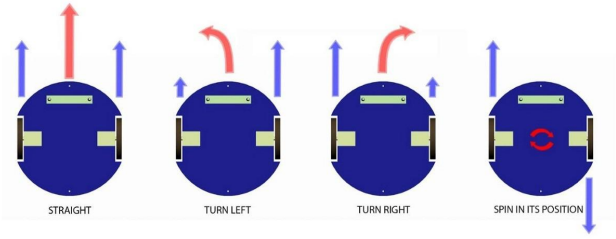


Figure 2: Direction of the robot as a result of relative motion of the wheels

If both the wheels are turned at equal speed, the robot follows a straight line. If the left wheel rotates faster than the right wheel, the robot turns towards right and similarly we can turn the robot to the left by increasing the speed of the right wheel. If both the wheels turn at equal speeds, but in opposite direction then there is a pure rotational motion of the robot. Therefore, we can steer the robot just by varying the speeds of the drive wheels.

0.1.7 DIFFERENTIAL DRIVE KINEMATICS

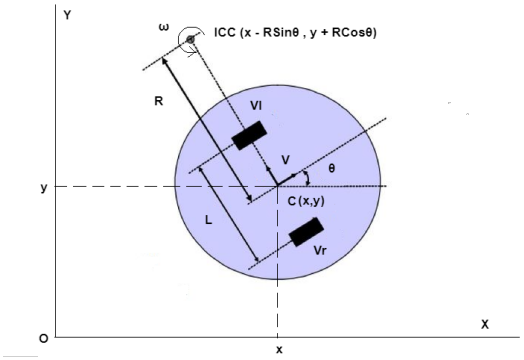


Figure 3: Kinematic Frame of reference

Here we will see about the mathematical aspects about the motion of the Differential Drive robot. Here the parameter are -

• ICC : Its the Instantaneous Center of Curvature. Depending on the the velocity and the direction of the wheels, its position changes. In the figure its towards to the left of the robot as its turning to the left (Vr more than Vl).

• C : The centre of the robot, given by the midpoint between the two actuated wheels.

• R : Its the Instantaneous Curvature Radius of the robot's trajectory. Visually its the distance between the Robot's center C and the ICC.

• L : Its the separation length between Left wheel and Right wheel.

• Vr : Its the linear velocity of the Right Wheel. Its value depends upon the rotation of the actuator connected to it.

• Vl : Its the linear velocity of the Left Wheel. Its value similarly depends upon the rotation of the actuator connected to it.

• : Its the angular velocity of the robot at the ICC.

• V : Its the velocity of the robot.

Now the radius of curvature of the

• Right wheel is: R + L/2

• Left wheel is: R - L/2

We need an expression for Angular velocity, Instantaneous radius and Velocity

Now, the angular velocity can be given as:

$$\omega(t) = \frac{V_l(t)}{R - L/2} \quad or \quad V_l(t) = (R - L/2)\omega(t)$$

$$\omega(t) = \frac{V_r(t)}{R + L/2} \quad or \quad V_l(t) = (R + L/2)\omega(t)$$

By subtracting V_r and V_l we get

$$V_r(t) - V_l(t) = L * \omega(t)$$

By adding V_r and V_l we get

$$V_r(t) + V_l(t) = 2 * R * \omega(t)$$

By using two equations and eliminating $\omega(t)$ we get:

$$R = \frac{L}{2} * \left(\frac{V_r(t) + V_l(t)}{V_r(t) V_l(t)} \right)$$

Since $V(t) = R\omega(t)$ we can say that : $V(t) = \frac{1}{2}(V_r + V_l)$

0.1.8 The Turtlebot

The Turtlebot is a family of low cost, easy to setup and assemble robots, using an open source software. In this project the simulator GAZEBO uses the model of Turtlebot 2. The TurtleBot kit consists of a mobile base, 3D Sensors, laptop computer, and the TurtleBot mounting hardware kit. From their website we get to know the specifications for Turtlebot 2. It uses a YUJIN Kobuki base, a 2200 mAh battery pack, a Kinect sensor.

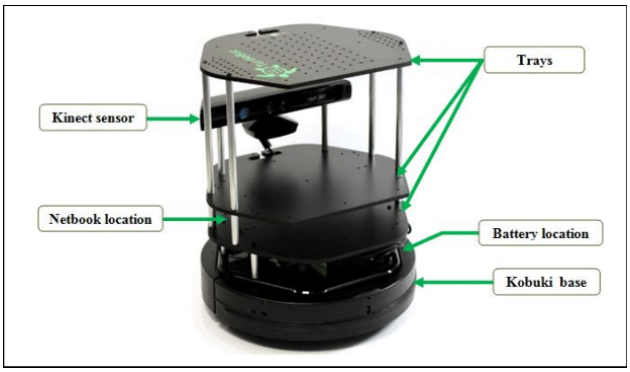


Figure 4: Turtlebot

0.1.9 References

1. Wikipedia page for Odometry. [https://en.wikipedia.org/wiki/Odometry]
2. Syscon talks on ROS.[https://sites.google.com/view/syscontalks/workshops/ros-workshop?authuser=0]

3. Coloumbia university notes. [http://www.cs.columbia.edu/allen/F17/NOTES/icckinematics.pdf]
4. Offical ROS wiki page.

0.2 Lab 2: Open Loop Control

In this section we are going to explore Open Loop Control of the turtle-bot in an open world simulation in Gazebo. We will be inspecting 3 straight line movements along x-axis, along y-axis and at a point diagonal to the axes in the plane. After that we will inspect the performance by traversing a triangle.

0.2.1 AIM

Therefore in the section we will :

- Inspect the movement along straight lines.
- Inspect the movement along a triangle.

0.2.2 Straight Line Trajectory

So the way it is programmed is that the robot is going to be initially at the Origin (Default). Now when first the robot will rotate along the z axis with some predefined speed until it is facing the goal point direction. Once it does it is going to move forward by facilitating both wheels moving forward at the same speed at a pre-determined speed until it is within an acceptable range of the goal point.

In short there are 2 modes of movements:

1. Rotation on the spot, to align towards the the goal. Its done by making R = 0 and magnitude of V_r and V_l same but opposite in sense of rotation.
2. Forward movement towards the goal. Its done by making the radius $R = \infty$ and magnitude of V_r and V_l same but with same sense of rotation.

Now, we will be seeing:

The path traced out by the simulated robot in gazebo shown in Orange. The variation of X and Y coordinates with time.The Orange graphs in the X and Y vs time plots indicate the goal point and the Blue indicate the path actually traversed.

Along X axis

We are going to move the robot from Origin to the point (10,0) on the X axis.

The sampling rate is 4 Hz, and the linear speed is 0.5 units/sec and angular speed is 0.3 units/sec. The path traversed. The variation of the X and Y coordinate with time is:

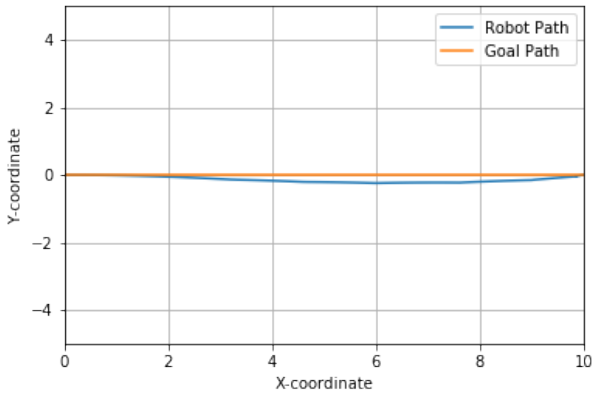
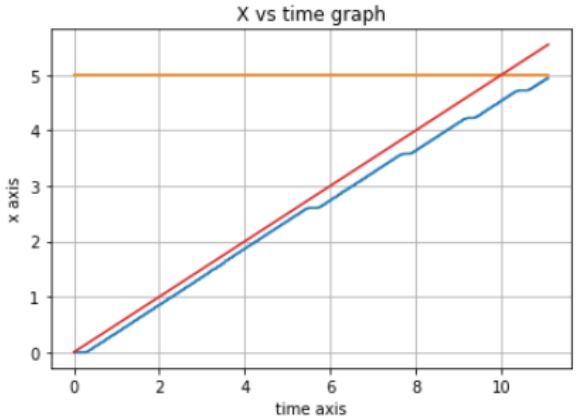
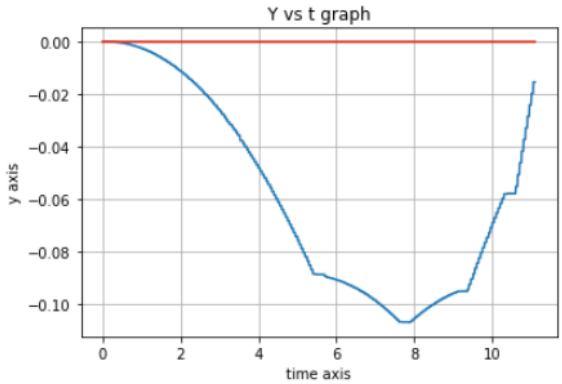


Figure 5: Trajectory of the robot along X-axis from Origin to (10,0)





Along Y axis

We are going to move the robot from Origin to the point (0,10) on the Y axis.

The sampling rate is 4 Hz, and the linear speed is 0.5 units/sec and angular speed is 0.3 units/sec. The path traversed. The variation of the X and Y coordinate with time is:

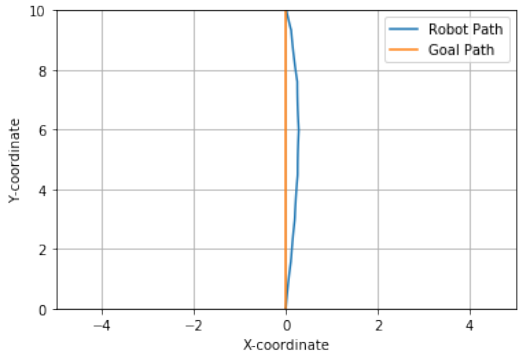
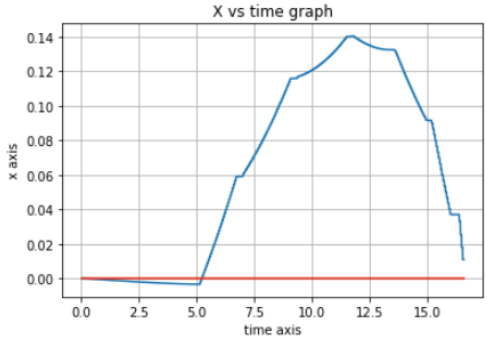
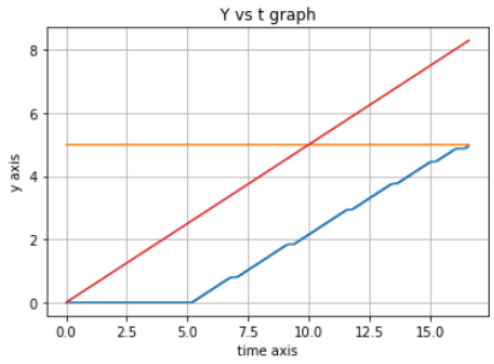


Figure 6: Trajectory of the robot along X-axis from Origin to (0,10)





Diagonally w.r.t. axes along a line joining Origin to the point (4,8)

We are going to move the robot from Origin to the point (4,8) The sampling rate is 100 Hz, and the linear speed is 0.5 units/sec and angular speed is 0.3 units/sec. The path traversed.

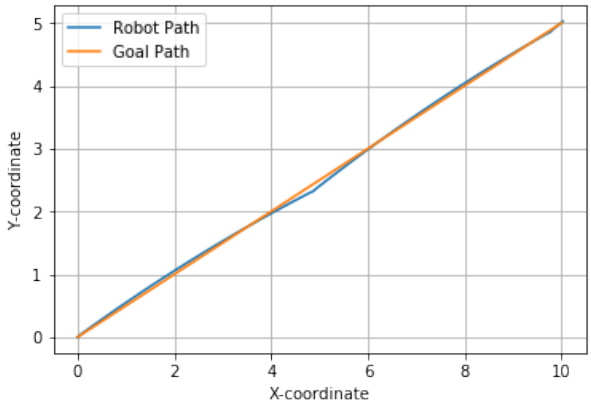
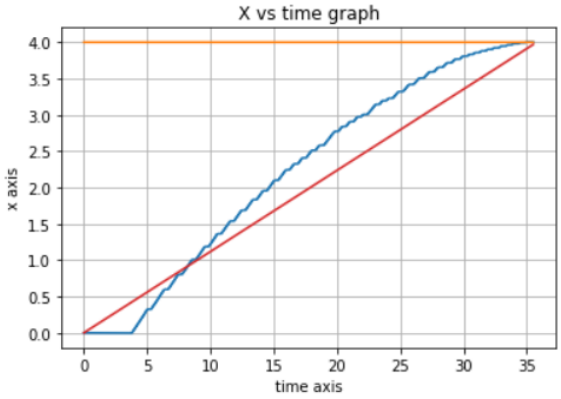
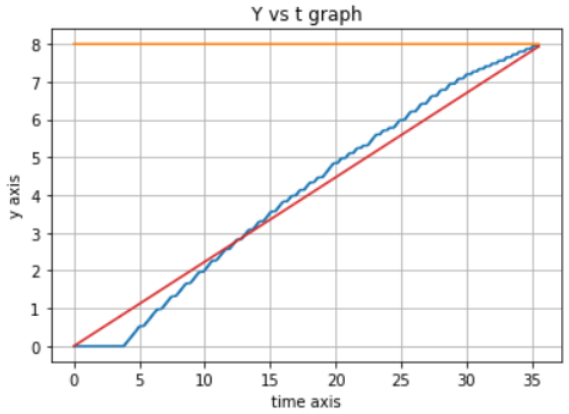


Figure 7: Trajectory of the robot along X-axis from Origin to (10,0)

The variation of the X and Y coordinate with time is:





0.2.3 Triangular Trajectory

So the way to do this was to simply change the goal points as soon as we reached a certain acceptable range of the goal point. And by doing that the robot can trace out any polygon (for our case a triangle). The robot is coded to move from origin to (0,5) then to (5,0) then finally back to (0,0). The simulation of the same is available at -

`https://drive.google.com/file/d/1qKb1lY8e8Fe9SwGLytUHLfSoekTMoQ9A/view?usp=sharing`

The path traversed: The variation of the X and Y coordinate with time will be:

The path traversed.

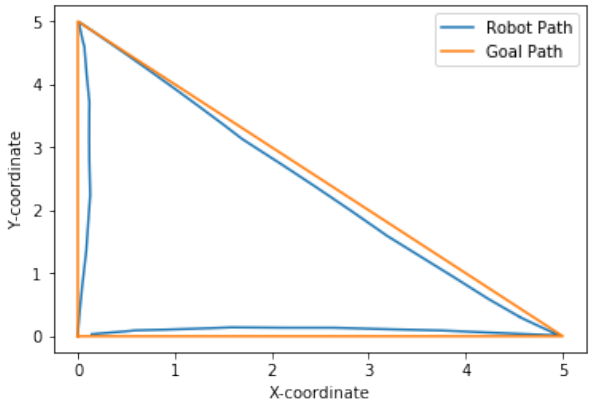
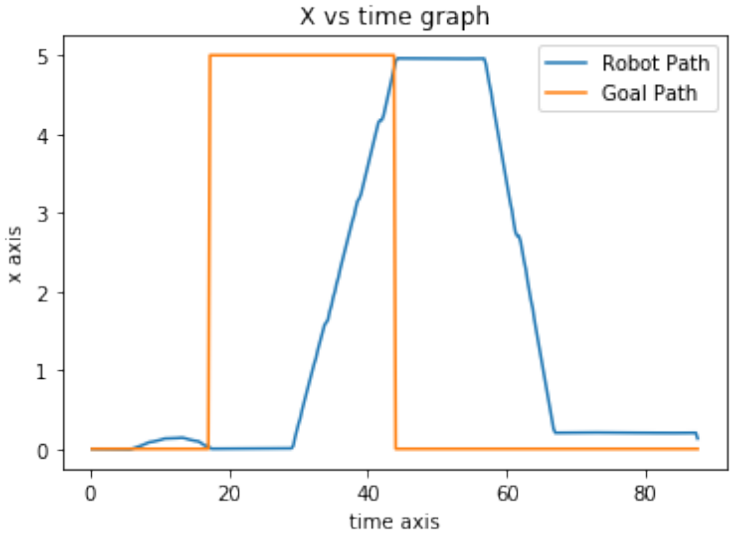
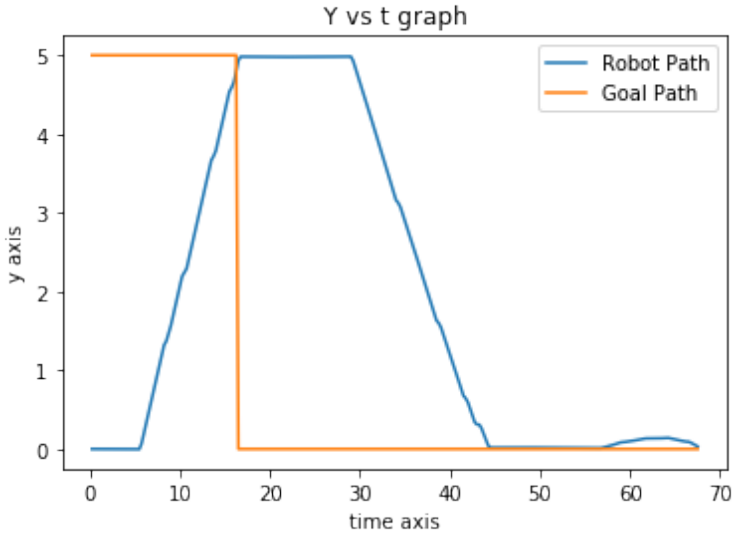


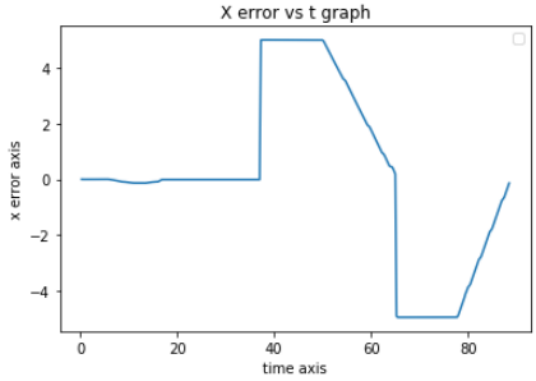
Figure 8: Trajectory of the robot along X-axis from Origin to (10,0)

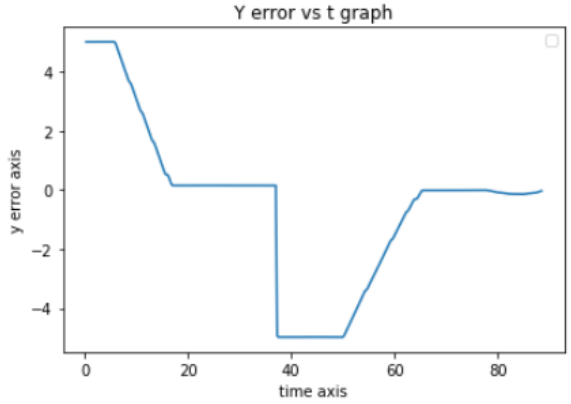
The variation of the X and Y coordinate with time is:

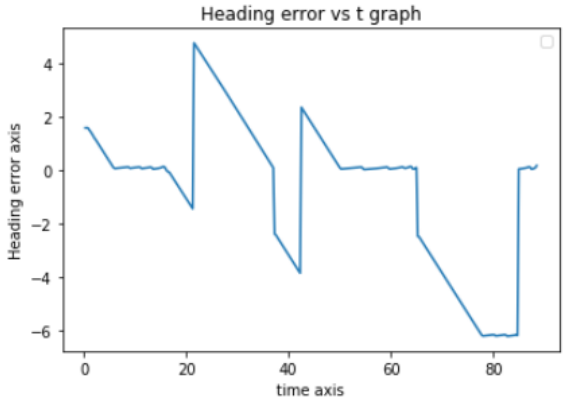




The variation of the X and Y and Heading error with time is:







0.2.4 Conclusions:

Straight Line

The robot deviates some amount from the original trajectory, but its small and acceptable as well. The deviations range to some values less than 0.25 units. Except that it is performing well.

Triangle

It is quite evident that the robot deviates some amount from the original trajectory, but on close inspection one can say that its small and acceptable. Here too, at the end the deviations range to some values less than 0.25 units.
Except that the robot is tracing the required triangle and is performing well.

0.2.5 References

1. Tutorials at the website The Constructs [https://www.theconstructsim.com/]
2. Offical ROS wiki page.
3. Python documentation available at Geeksforgeeks website

0.3 Lab 3: PID Control

In this section we are going to use a differential drive robot to trace an infinity curve using Lissajous equations.

0.3.1 Aims

Therefore in the report we will :

- Learn about Lissajous curves and the requirements to make an infinity curve.
- Implement the equations to trace out an infinity curve.
- Use a PID control scheme and tuning them to get best performance

0.3.2 Lissajous Figures

These figures of Complex Harmonic Motion and are obtained when 2 SHM motions(or systems or equations) which are spatially perpendicular to each other are superimposed. Say,

$$x(t) = A\ Sin(at + \delta) \quad and \quad y(t) = B\ Sin(bt)$$

The appearance of the figure is highly sensitive to the ratio of $\frac{a}{b}$ and the overall magnitude of the shape is shown by $\frac{A}{B}$ and lastly the δ.
For our case we'll be using a:b = 2:1 with $\delta = 0$ and A = B = 3 units.

0.3.3 PID control

In Matalab and Simulink, PID is implemented using the inbuilt PID packages, but in our case (for Gazebo) we have to calculate the input in terms of errors and implement the PID control on the robot.

$$u(t) = k_p\ e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt}$$

where $u(t)$ is the control / manipulated variable, $e(t)$ is the value of the error given by e(t) = r(t) y(t) (reference - current), k_p k_i and k_d are the proportional, integral and derivative gains respectively.

In discrete time we need to discretize this so instead of this equation we simply have to use difference equations. Thus by differentiating both side and using Back Difference we get the discrete time equations like:

$$\ddot{u}(t) = K_p\ddot{e}(t) + K_i e(t) + K_d\ddot{e}(t) \quad \textit{then by backward difference},$$

$$\frac{u(t_k) - u(t_{k-1})}{t} = K_p \frac{e(t_k) - e(t_{k-1})}{t} + K_i e(t_k) + K_d \frac{e(t_k) - 2e(t_{k-1}) + e(t_{k-2}))}{t^2} \quad (t = \textit{sampling interval})$$

Then for the k^{th} instant and multiplying by t both sides,

$$u(t_k) = u(t_{k-1}) + K_p\ (e(t_k) - e(t_{k-1})) + K_i\ t\ e(t_k) + K_d \frac{e(t_k) - 2e(t_{k-1}) + e(t_{k-2}))}{t}$$

And that is how we will build our code. (t = 0.2 for our case)

0.3.4 Navigation

So we have out equations now all we need is the driving equation to be incorporated. First we set the goal points as the parametric equations and then we that to find the error of distance or displacement. Then using the coordinates we find the angle to the goal point and then subtract the current heading to get the angle error. It will look like this:

$$e1(t) = ((y_{goal} \quad y_{current})^2 + (x_{goal} \quad x_{current})^2)^{\frac{1}{2}}$$

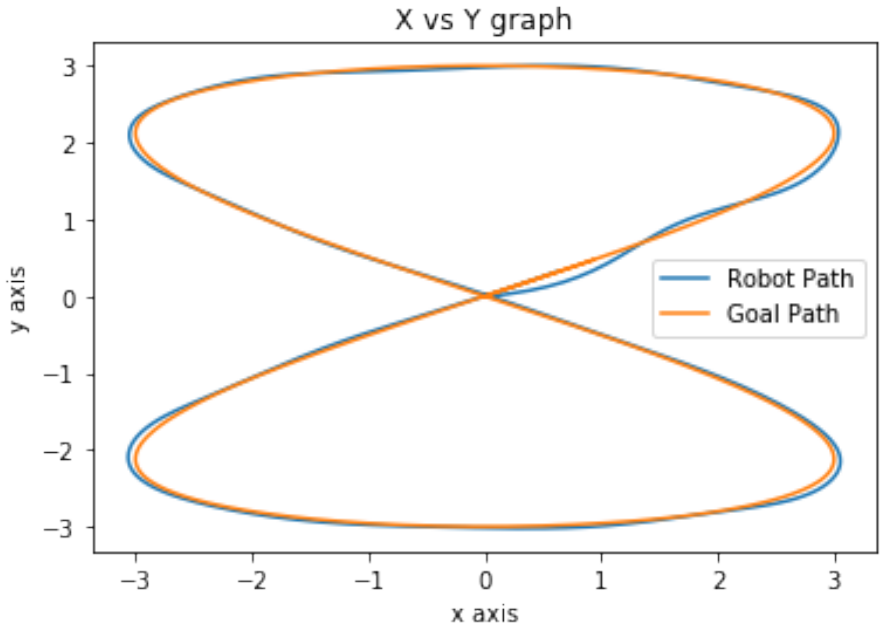
$$e2(t) = tan^{-1} \frac{(y_{goal} \quad y_{current})}{(x_{goal} \quad x_{current})} - \quad \theta_{current}$$

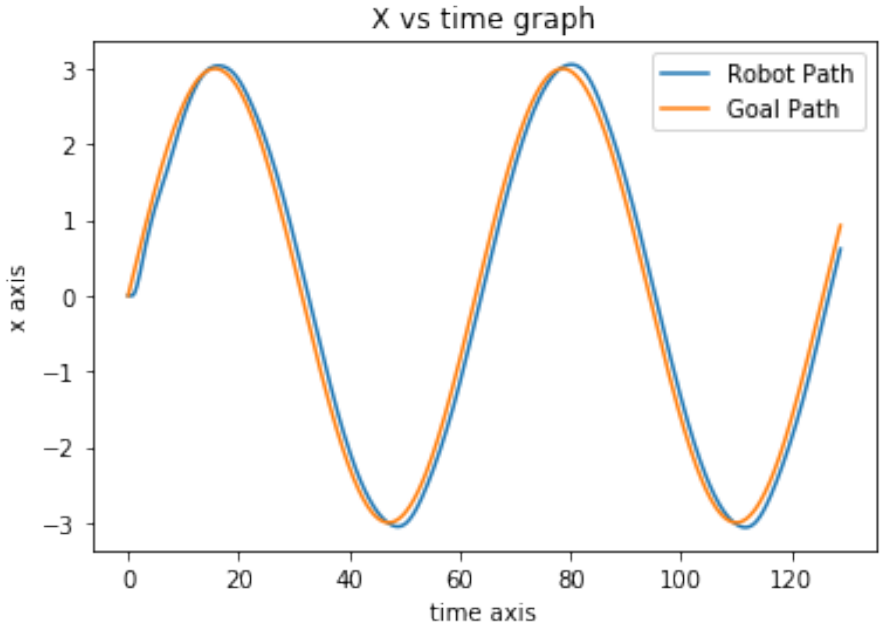
We will be storing these values in an external list so that we can refer to the past values

0.3.5 Results

Using P

Now a good result was seen using a P controller. Blue is Goal trajectory and Orange is Traversed trajectory. The graphs showing the behaviour of the bot is shown below for the values of $K_p \ = 2$ and $K_i = K_d \ = 0$.




X vs time graph
Robot Path
Goal Path
x axis
time axis
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120

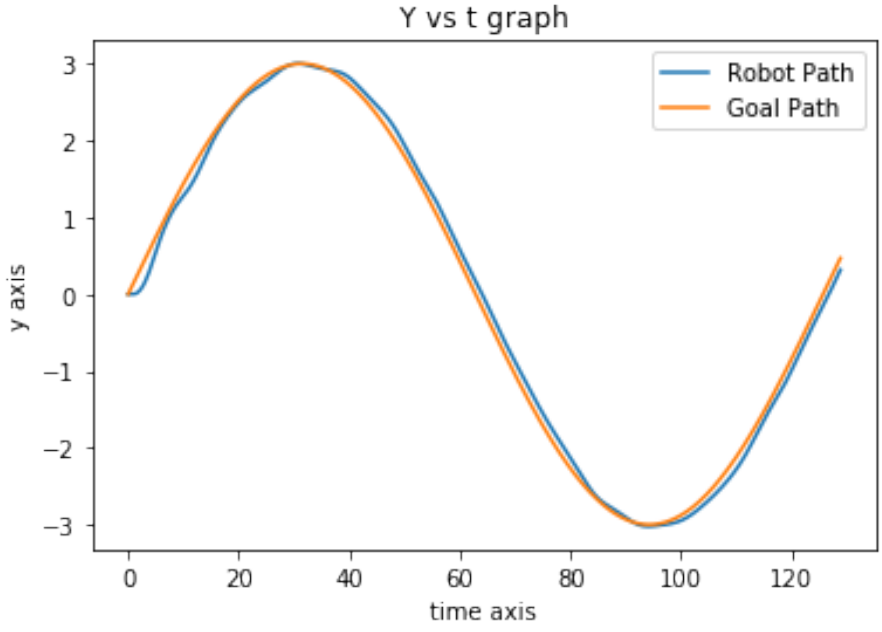
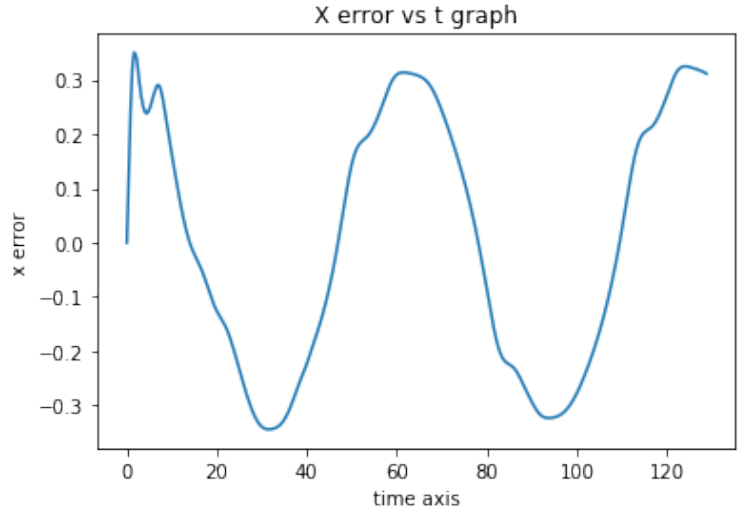
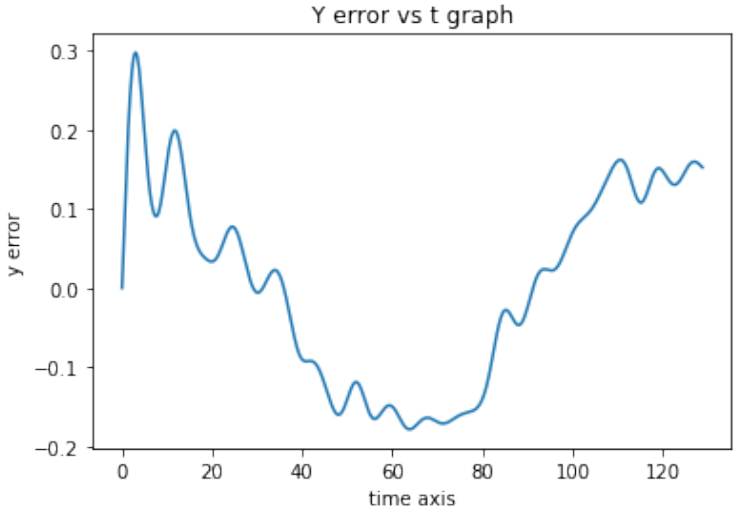
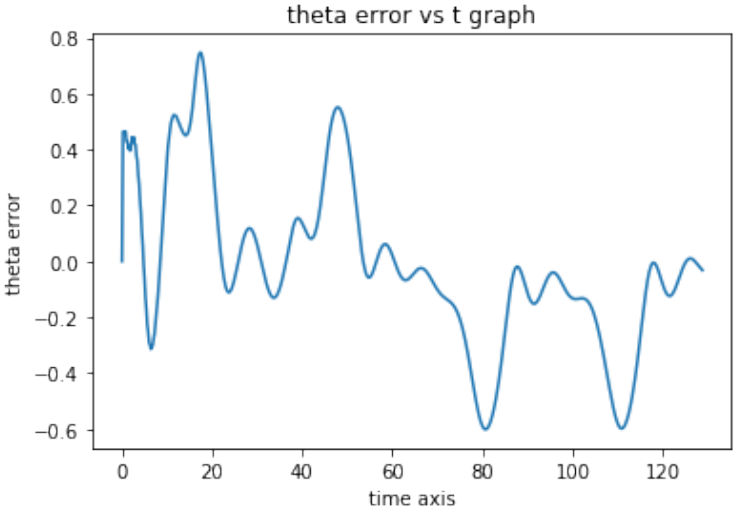

Y vs t graph
Robot Path
Goal Path
y axis
time axis
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120

Figure 9:

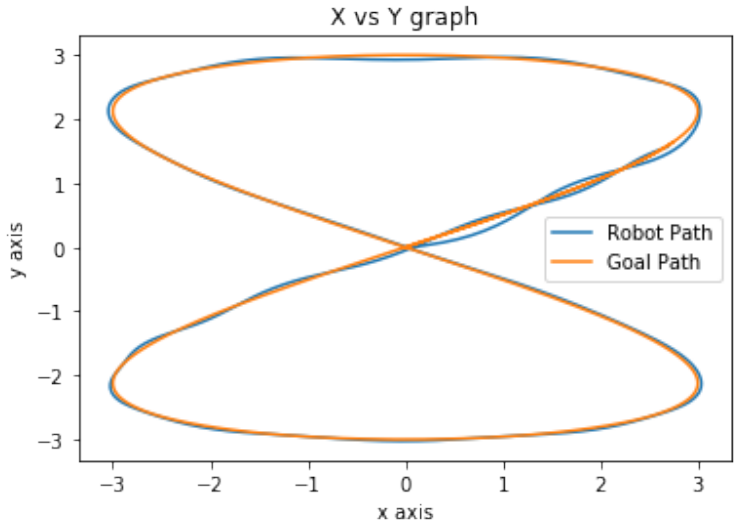

X error vs t graph
x error
time axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
20
40
60
80
100
120

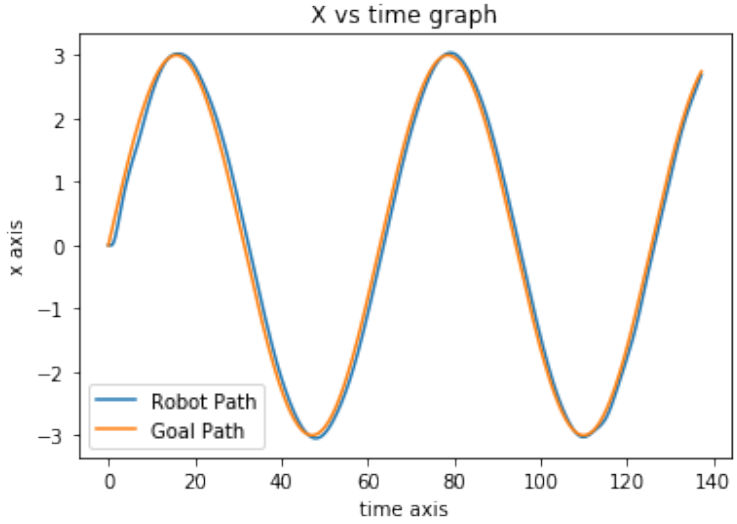

Y error vs t graph
y error
time axis
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
120

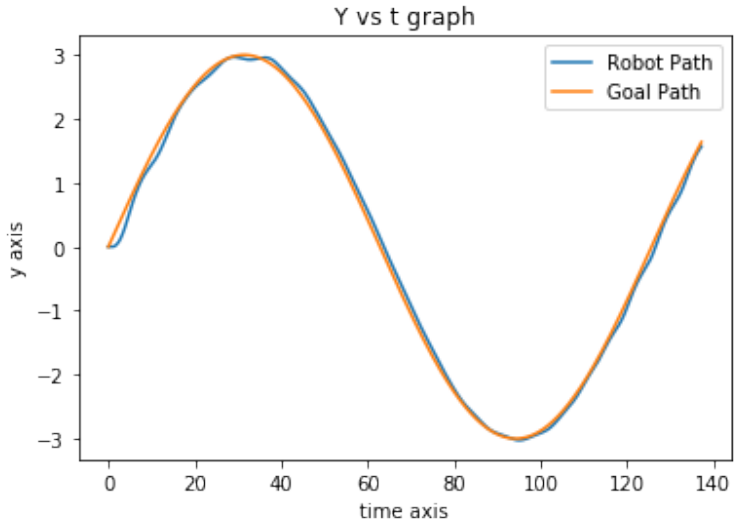

theta error vs t graph
theta error
time axis
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
120

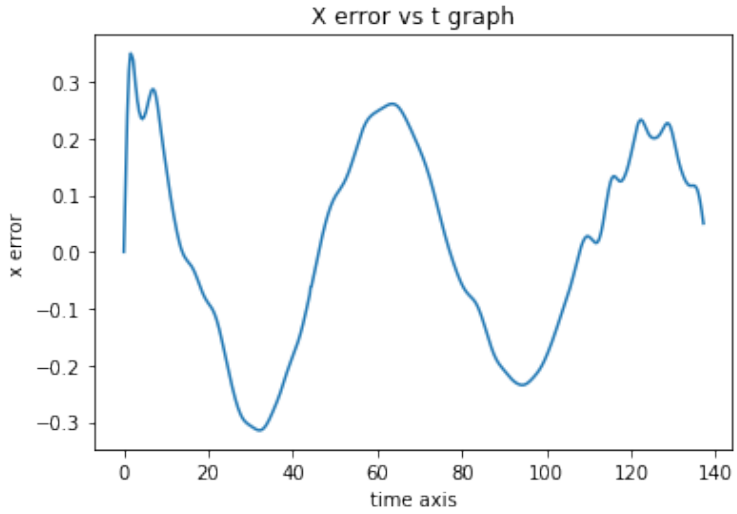
Using PI

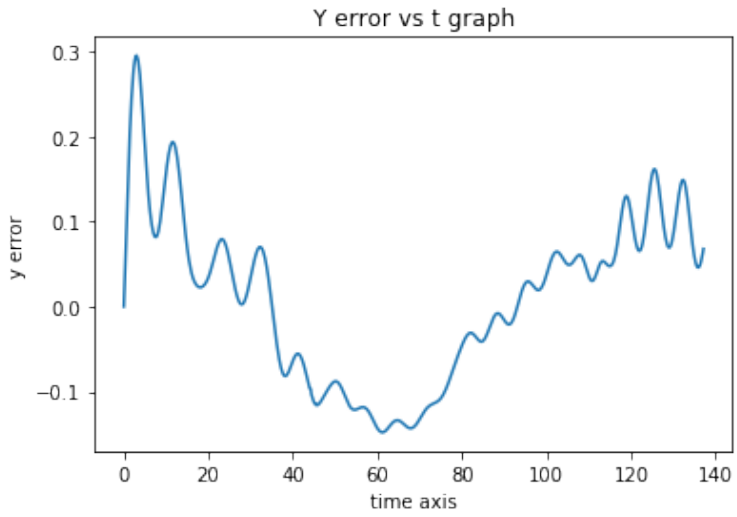
Now a better result was seen using a PI controller. Blue is Goal trajectory and Orange is Traversed trajectory. The behaviour of the bot is shown below for the values of $K_p = 2$ and $K_i = 0.01$ and $K_d = 0$.

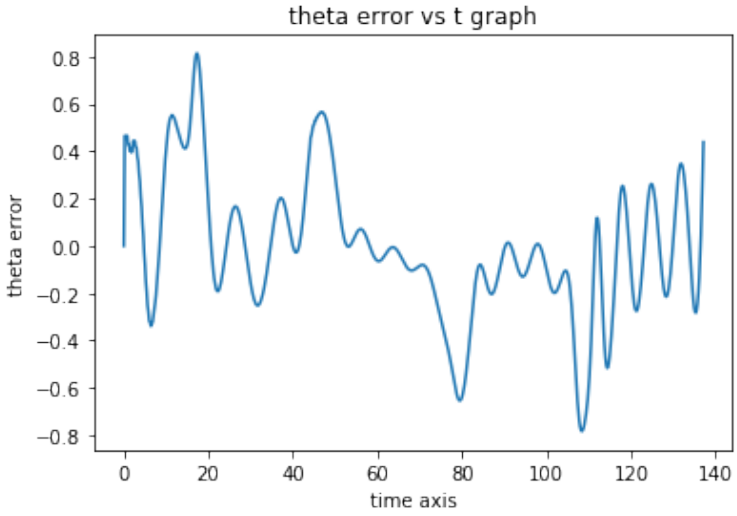





Y vs t graph
Robot Path
Goal Path
y axis
time axis
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120
140

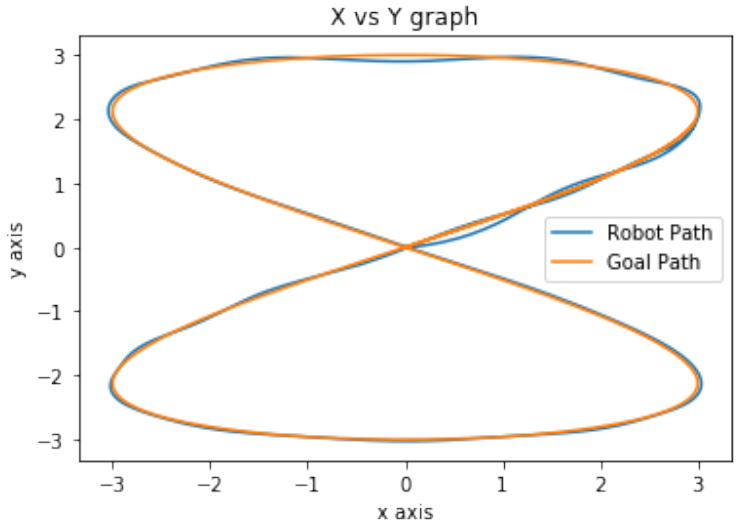

X error vs t graph
x error
time axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
20
40
60
80
100
120
140

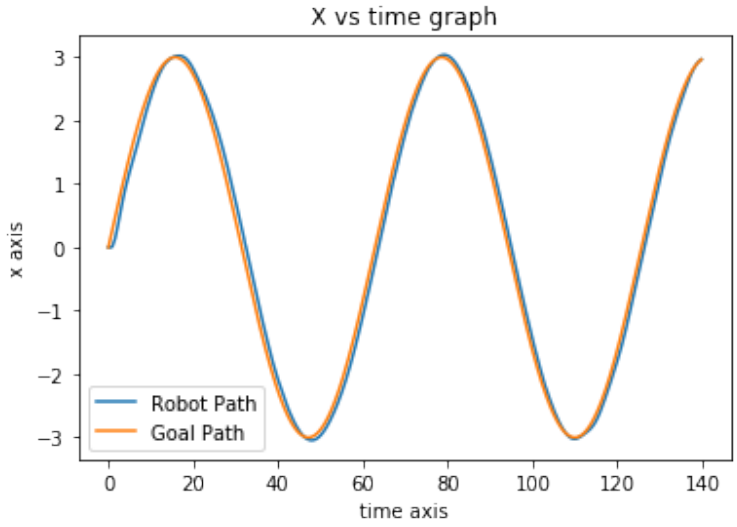

Y error vs t graph
y error
time axis
0.3
0.2
0.1
0.0
−0.1
0
20
40
60
80
100
120
140

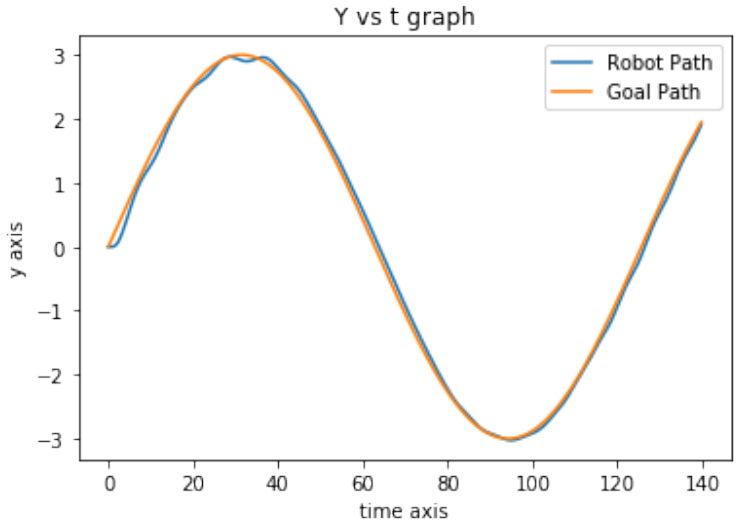

theta error vs t graph
theta error
time axis
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
20
40
60
80
100
120
140

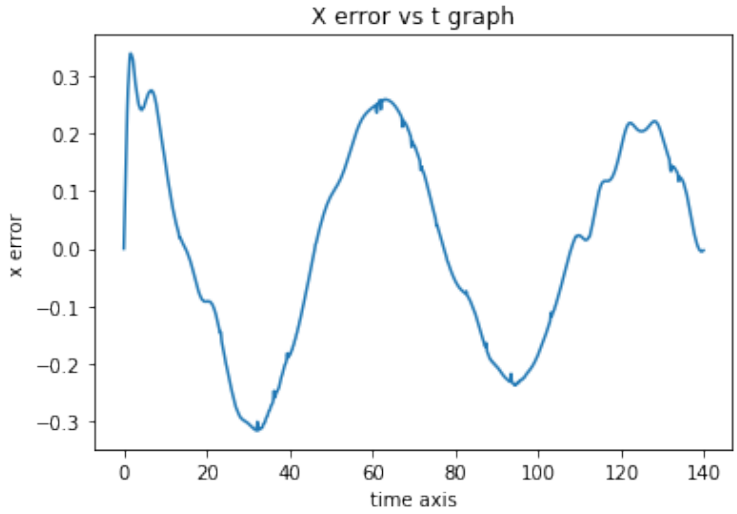
Using PID

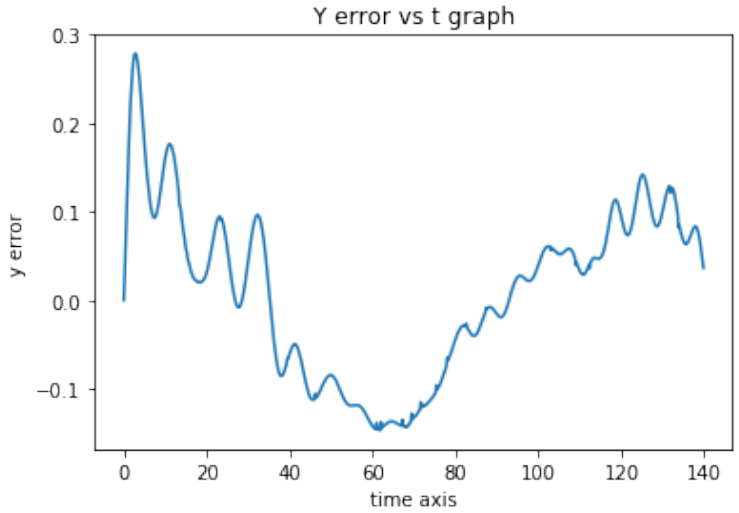
Now a better result was seen using a PID controller. Blue is Goal trajectory and Orange is Traversed trajectory. The graphs showing the behaviour for the values of $K_p = 2$ and $K_i = 0.01$ and $K_d = 0.1$.

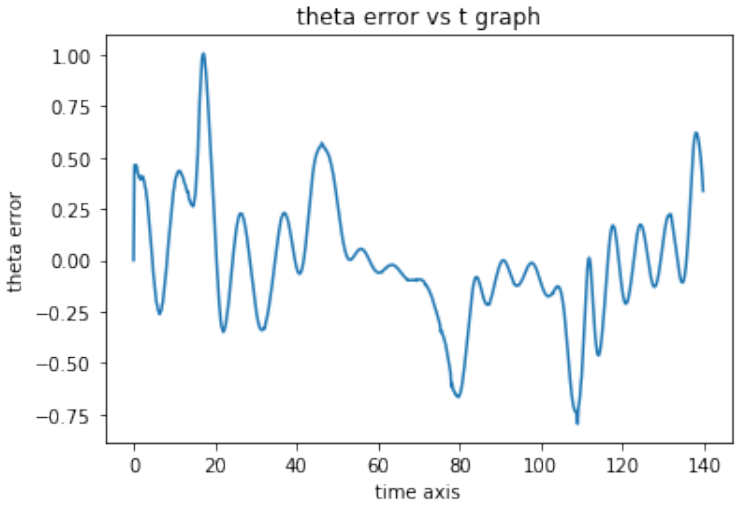





Y vs t graph
Robot Path
Goal Path
y axis
time axis
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
120
140


X error vs t graph
x error
time axis
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
20
40
60
80
100
120
140


Y error vs t graph
y error
time axis
0.3
0.2
0.1
0.0
−0.1
0
20
40
60
80
100
120
140


theta error vs t graph
theta error
time axis
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100
120
140

0.3.6 Conclusion

Controller	dist. RMSE (metres)	heading RMSE (radians)	Time to finish (sec)
P ($K_p = 2$)	0.2665	0.2784	140
PI ($K_p = 2$ $K_i = 0.01$)	0.2053	0.3098	125
PID ($K_p = 2$ $K_i = 0.01$ $K_d = 0.1$)	0.2014	0.2968	120

So we can see from the graphs and metrics, PID is performing better than the other two, in terms of both error and time taken, although PID is a bit shaky on its path than PI. P controller follows the path satisfactorily as well. But, PI and PID both perform well in these situations. Video links of the
P controller : `https://drive.google.com/file/d/1tHsc4G6hc5d9IwI4pemQQeNqjiDGrMwA/view?usp=sharing`
PI controller : `https://drive.google.com/file/d/13i4cL8prrQ2GpbUpM4BIz481mM3sDip3/view?usp=sharing`
PID controller : `https://drive.google.com/file/d/1yiAf2mAUKLO0J36eHZNtJSImyq8e8zZv/view?usp=sharing`

0.3.7 References

1. Wikipedia page for Lissajous: https://en.wikipedia.org/wiki/Lissajouscurve
2. Python documentations.

0.4 Lab 4: Non-Linear Control

In this report we are going to make a differential drive robot to trace an infinity curve using Lissajous equations using a Non-Linear controller as depicted in the page 40 of "Surveillance and Monitoring Strategies for Autonomous Mobile Agents" - A thesis by Mr. Aseem Vivek Borkar.

0.4.1 Aims

Therefore in the report we will :

- Learn about Lissajous curves and the requirements to make an infinity curve.
- Implement the Non - Linear equations to trace out an infinity curve.
- Tune the gains to them to get best performance

0.4.2 Lissajous Figures

These figures of Complex Harmonic Motion and are obtained when 2 SHM motions(or systems or

equations) which are spatially perpendicular to each other are superimposed. Say,

$$x(t) = A\ Sin(at + \delta) \quad and \quad y(t) = B\ Sin(bt)$$

The appearance of the figure is highly sensitive to the ratio of $\frac{a}{b}$ and the overall magnitude of the shape is shown by $\frac{A}{B}$ and lastly the δ.

For our case we'll be using a:b = 2:1 with $\delta = 0$ and A = B = 3 units.

In discrete time the equations will look like this:

$$x(t) = A\ Sin(\frac{ak}{N} + \delta) \quad and \quad y(t) = B\ Sin(\frac{bk}{N})$$

N is a discretizing factor, chosen such that the robot can follow up with the change of trajectory.

For Our case we use the values of $\frac{a}{N} = 0.1$ and $\frac{b}{N} = 0.05$

0.4.3 The Non-Linear controller

Let us see what the controller looks like mathematically.

Control law for linear velocity:

$$V(t) = k_v\ D\ Cos(e_\alpha)$$

Control law for angular velocity:

$$\omega(t) = k_\alpha\ e_\alpha\ +\ \dot{\alpha}(t)$$

Where,

x_p *and* y_p *are Reference or goal coordinates* x *and* y *are Current coordinates* K_v is the is the controller gain for linear velocity

K_α is the controller gain for angular velocity

D is the distance error = $((x_p - x)^2 + (y_p - y)^2)^{\frac{1}{2}}$

α is the heading rate = $tan^{-1}\left(\frac{Y_{goal}}{X_{goal}}\right)$.

e_α is the heading error = $\alpha^c - \alpha^r$

α^c is the goal or target heading angle and α^r is the current heading angle.

Now the α^c is calculated as:

$$\alpha^c = tan^{-1}\left(\frac{y_p - y}{x_p - x}\right)$$

The quantity of $\dot{\alpha}$ will be:

$$\dot{\alpha} = \frac{d}{dt}(tan^{-1}\left(\frac{Y_p}{X_p}\right))$$

$$\dot{\alpha} = \frac{x_p \dot{y_p} - y_p \dot{x_p}}{(x_p^2 + y_p^2)}$$

All these are in the continuous time. We nee to create the equivalent discrete equations in order to implement the equations in our code. We simply have to use difference equations. Thus using it we get the equations like this:

$$V(k) = k_v \; D(k) \; Cos(e_\alpha(k))$$

$$\omega(t) = k_\alpha \; e_\alpha(k) \; + \; \frac{x_p(y_p(k) - y_p(k-1)) - y_p(x_p(k) - x_p(k-1))}{h \; (x_p^2 + y_p^2)}$$

SO, using these formulas we tune the parameters K_v and K_α manually to get the best performance:

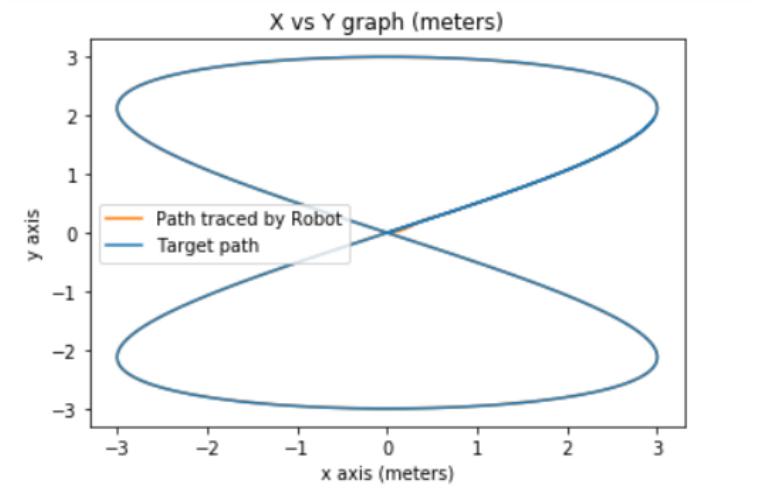
0.4.4 Observations

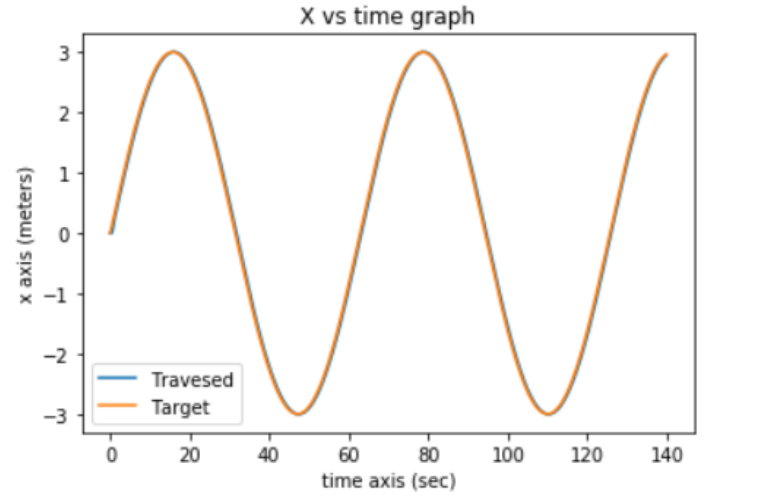
K_α	K_v	Dist. RMSE (meters)	Heading RMSE (radians)	Time to finish (sec)
1	1	0.2589	0.1126	170
2	1	0.2949	0.0600	163
5	1	0.2580	0.0380	187
1	2	0.1298	0.1171	224
1	5	0.0529	0.1520	204
2	2	0.1500	0.0625	195
3	3	0.0805	0.0491	180

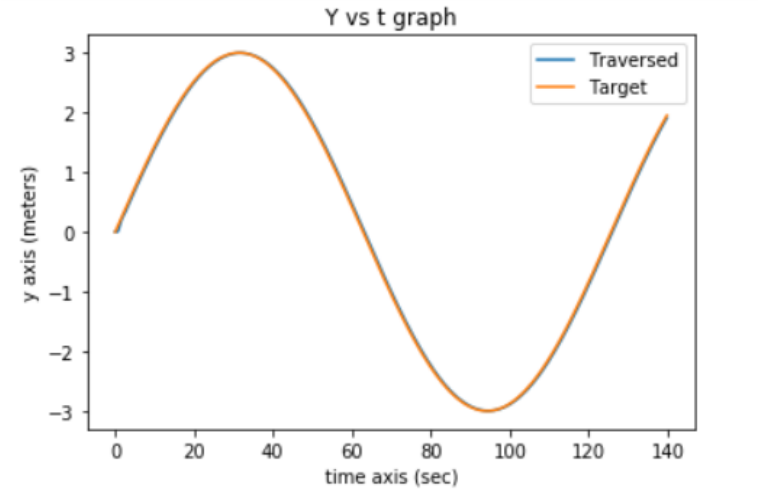
We see a decrease in the Distance RMSE and Heading RMSE for increase in K_α and K_v values respectively. In order to reduce the error I increased it further, to find out that high K_v values made the robot take more time to adjust to the trajectory, but the overall time taken to complete is lowered after it adjusts to the trajectory. Increasing K_α made the robot take more time but the adjustment to the trajectory was better and smooth.

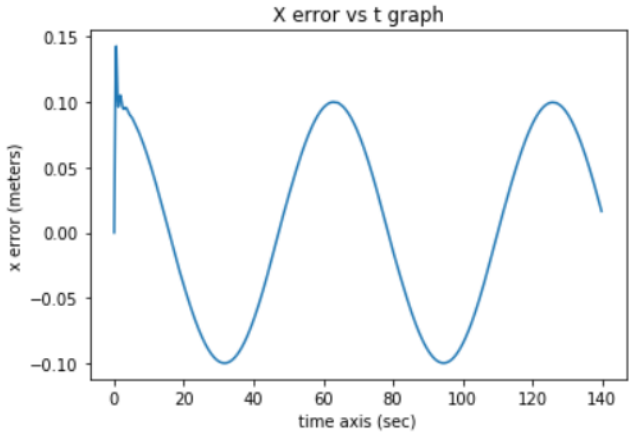
As far as the error is concerned there was always a small offset for lower gain values which owing to the nature of the reference was also sinusoidal.

At last I chose the pair of gains $K_\alpha = 3$ and $K_v = 3$ as increasing beyond this does decrease the error but the time to settle to the trajectory increases. The results are as follows: (Blue is Goal trajectory and Orange is Traversed trajectory.)

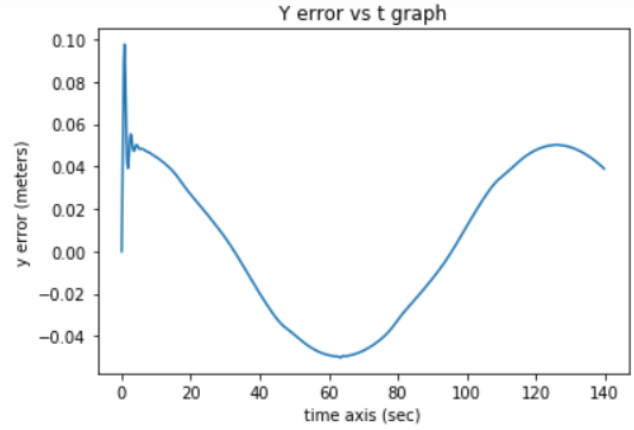

X vs Y graph (meters)
Path traced by Robot
Target path
y axis
x axis (meters)


X vs time graph
Travesed
Target
x axis (meters)
time axis (sec)

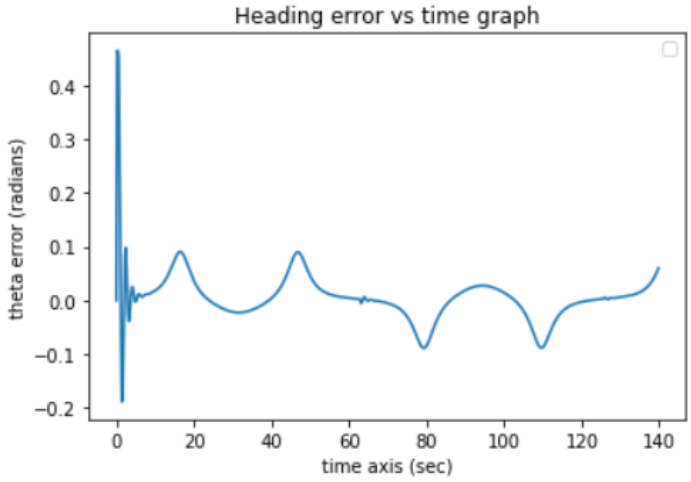

Y vs t graph
Traversed
Target
y axis (meters)
time axis (sec)

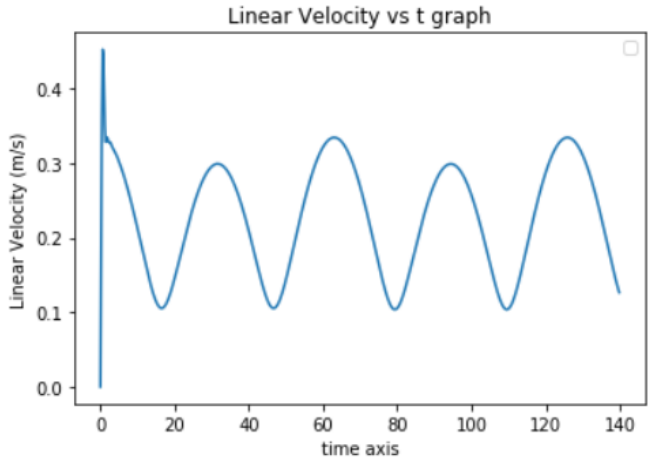


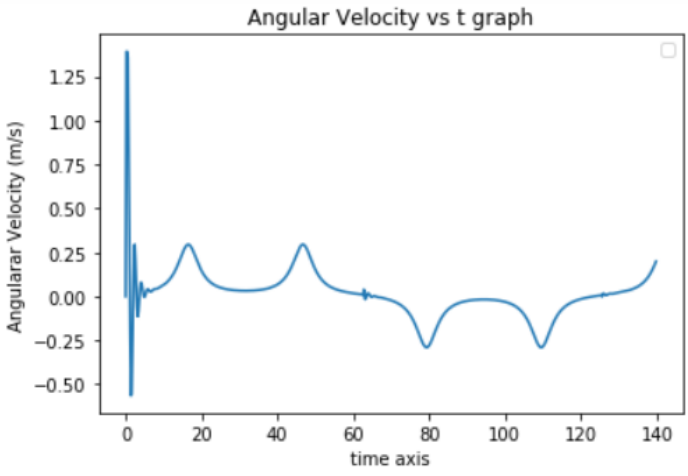
('The RMSE of x error is (in meters) :', '0.0709791363439')



('The RMSE of y error is (in meters) :', '0.0368629504138')




Linear Velocity vs t graph
Linear Velocity (m/s)
0.0
0.1
0.2
0.3
0.4
0
20
40
60
80
100
120
140
time axis


Angular Velocity vs t graph
Angularar Velocity (m/s)
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
0
20
40
60
80
100
120
140
time axis

0.4.5 Conclusion

Controller	Dist. RMSE (meters)	Heading RMSE (radians)	Time to finish (sec)
P	0.2665	0.2784	140
PI	0.2053	0.3098	125
PID	0.2014	0.2968	120

This was the performance metrics of a PID. Judging by the values we can clearly find Non-Linear controllers perform better than that of PID controllers. It even traces the trajectory indistinguishably, which is a major improvement in comparison to PID controllers. Although it takes a few seconds more to complete the same trajectory, which can be attributed to the fact that it starts a bit slow, but once it sets itself it takes almost the same time.

Video link of the non-linear controller :

`https://drive.google.com/file/d/1CTA5GOq_Rzg4x8mSyWLDOODm-aZpfRpR/view?usp=sharing`

0.4.6 References

1. Wikipedia page for Lissajous : https://en.wikipedia.org/wiki/Lissajous$_curve$

2.Surveillance and Monitoring Strategies for Autonomous Mobile Agents by Mr. Aseem Vivek Borkar.

3.*Python documentations.*

0.5 Lab 5: Implementing a Follower robot.

In this report we are going to implement a differential drive robot to follow a reference robot which will be tracing the an infinity curve and another Lissajous equations using a Non-Linear controller as depicted in the page 809 of "Springer Handbook of robotics" by Bruno Siciliano and Oussama Khatib. The reference robot will be using the Non Linear controller given on Page 40 of "Surveillance and Monitoring Strategies for Autonomous Mobile Agents" - A thesis by Mr. Aseem Vivek Borkar.

0.5.1 Aims

Therefore in the report we will :

- Implement the Non - Linear equations to trace out an infinity curve and another Lissajous figure by a reference which is followed by a follower robot.

- Tune the gains to them to get best performance

0.5.2 Lissajous Figures

These figures of Complex Harmonic Motion and are obtained when 2 SHM motions(or systems or equations) which are spatially perpendicular to each other are superimposed. Say,

$$x(t) = A\ Sin(at + \delta) \quad and \quad y(t) = B\ Sin(bt)$$

The appearance of the figure is highly sensitive to the ratio of $\frac{a}{b}$ and the overall magnitude of the shape is shown by $\frac{A}{B}$ and lastly the δ.

For our case we'll be using a:b = 2:1 and a:b = 3:2 with $\delta = 0$ and A = B = 3 units.

In discrete time the equations will look like this:

$$x(t) = A\ Sin(\frac{ak}{N} + \delta) \quad and \quad y(t) = B\ Sin(\frac{bk}{N})$$

N is a discretizing factor, chosen such that the robot can follow up with the change of trajectory.

For Our case we use the values of $\frac{a}{N} = 0.1$ and $\frac{b}{N} = 0.05$ for a:b = 2:1

and values of $\frac{a}{N} = 0.15$ and $\frac{b}{N} = 0.1$ for a:b = 3:2.

0.5.3 The Non-Linear controllers

The reference robot's controller looks like mathematically.

Control law for linear velocity:

$$V(t) = k_v\ D\ Cos(e_\alpha)$$

Control law for angular velocity:

$$\omega(k) = k_\alpha\ e_\alpha\ +\ \dot{\alpha}(t)$$

Where,

x_p *and* y_p *are Reference or goal coordinates* x *and* y *are Current coordinates* K_v is the is the controller gain for linear velocity

K_α is the controller gain for angular velocity

D is the distance error = $((x_p - x)^2 + (y_p - y)^2)^{\frac{1}{2}}$

α is the heading rate = $tan^{-1}\left(\frac{Y_{goal}}{X_{goal}}\right)$.

e_α is the heading error = $\alpha^c - \alpha^r$

α^c is the goal or target heading angle and α^r is the current heading angle.

Now the α^c is calculated as:

$$\alpha^c = tan^{-1}\left(\frac{y_p - y}{x_p - x}\right)$$

The quantity of $\dot{\alpha}$ will be:

$$\dot{\alpha} = \frac{d}{dt}(tan^{-1}\left(\frac{Y_p}{X_p}\right))$$

$$\dot{\alpha} = \frac{x_p\dot{y_p} - y_p\dot{x_p}}{(x_p^2 + y_p^2)}$$

All these are in the continuous time. We nee to create the equivalent discrete equations in order to implement the equations in our code. We simply have to use difference equations. Thus using it we get the equations like this:

$$V(k) = k_v\ D(k)\ Cos(e_\alpha(k))$$

$$\omega(t) = k_\alpha\ e_\alpha(k)\ +\ \frac{x_p(y_p(k) - y_p(k-1)) - y_p(x_p(k) - x_p(k-1))}{h\ (x_p^2 + y_p^2)}$$

So, using these formulas we tune the parameters K_v and K_α manually to get the best performance and got them for $K_v = 3$ and $K_\alpha = 3$.

Now the Controller for the follower or tracking robot

These are linearized near the origin. Control law for linear velocity:

$$V_{tra}(t) = V_{ref}(t)\ -\ K_1|V_{ref}(t)|Z_1$$

Control law for angular velocity:

$$\omega_{tra}(t) = \omega_{ref}(t)\ -\ K_2\ V_{ref}(t)\ Z_2\ -\ K_3|V_{ref}(t)|Z_3$$

Where,

K_1,K_2 and K_3 is the is the controller gain for linear velocity

$Z_1 = X_e = x_{curr} - x_{goal}$

$Z_2 = Y_e = y_{curr} - y_{goal}$

$Z_3 = tan(\theta_e)$ where θ_e is the Heading error

$W_1 = V_{tra}\ cos(\theta_e)\ -\ V_{ref}$

$W_2 = \frac{\omega_{tra}\ -\ \omega_{ref}}{cos^2(\theta_e)}$

The control law is given by W_1 and W_2 but we can linearize it to a more simpler version as given above. Since near the origin: $W_1 \approx V_{tra}(k) - V_{ref}(k)$ and $W_2 \approx \omega_{tra}(k) - \omega_{ref}(k)$ and

In discrete time, it looks like this

$$V_{tra}(k) = V_{ref}(k)\ -\ K_1|V_{ref}(k)|Z_1$$

$$\omega_{tra}(k) = \omega_{ref}(k) \; - \; K_2 \; V_{ref}(k) \; Z_2 \; - \; K_3 |V_{ref}(k)| Z_3 \quad for \; k \in I$$

0.5.4 Observations

I have tuned the Infinity curve first and used the same configuration for the other Lissajous curve.

K_1	K_2	K_3	Dist. RMSE (meters)	Heading RMSE (radians)	Time to finish (sec)
0.01	0.01	0.01	0.079962	0.04897	126
0.1	0.1	0.1	0.079803	0.04935	126
1	1	1	0.079864	0.04864	126
2	2	2	0.083605	0.04063	126
1	2	2	0.079550	0.04030	126
1	2	1	0.079830	0.04901	126
0.5	2	2	0.07953	0.04627	126
0.5	2.5	2	0.08010	0.05011	126
1	2	2.5	0.07983	0.04670	126

For the second Lissajous curve:

K_1	K_2	K_3	Dist. RMSE (meters)	Heading RMSE (radians)	Time to finish (sec)
1	2	2	0.13088	0.07424	128

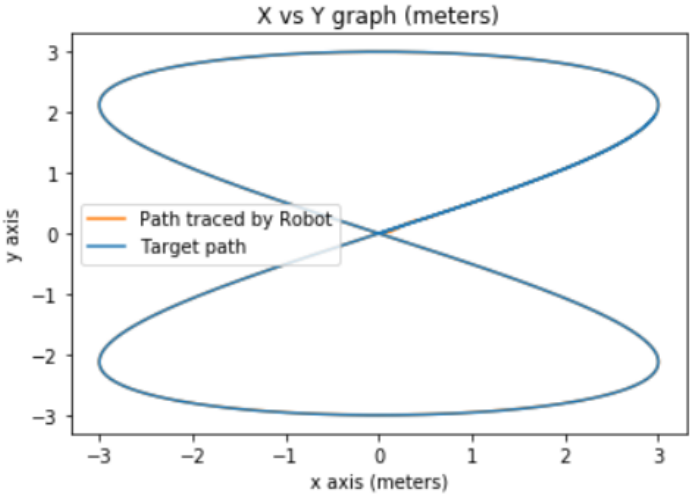
We see a decrease in the Distance RMSE and Heading RMSE for increase in K_1 and K_2 and K_3 values respectively for all from 0.01 to 1. Then on increasing the all the values to 2 there is a decreased in Heading RMSE, but Distance RMSE has increased, thus I specifically reduced K_1. And finally i got the best pair of values from my entire observation list. I still experimented with many more values and finally found that this is the best candidate possible.

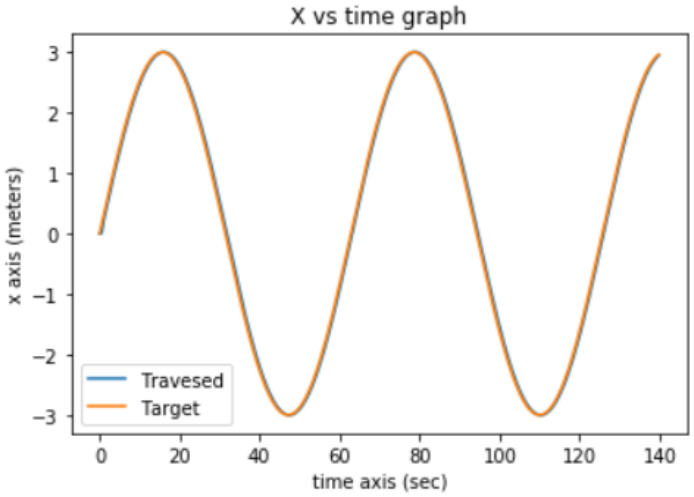
The overall time taken to complete is almost the same every time. This is good since it means that the robot is tracking the reference robot perfectly and the gains only affect the behaviour of the robot in terms of settling to the reference robot trajectory.

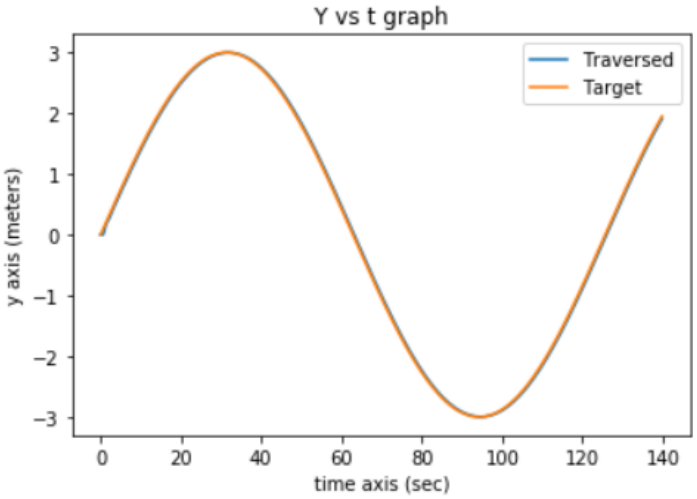
As far as the error is concerned there was always a small offset for lower gain values which owing to the nature of the reference robot was also sinusoidal.

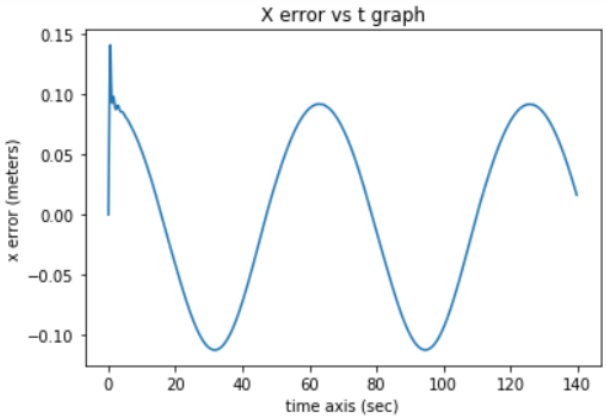
At last I chose the pair of gains $K_\alpha = 3$ and $K_v = 3$ and K_1 and K_2 and K_3 as 1, 2, 2 :

For the Infinity curve.

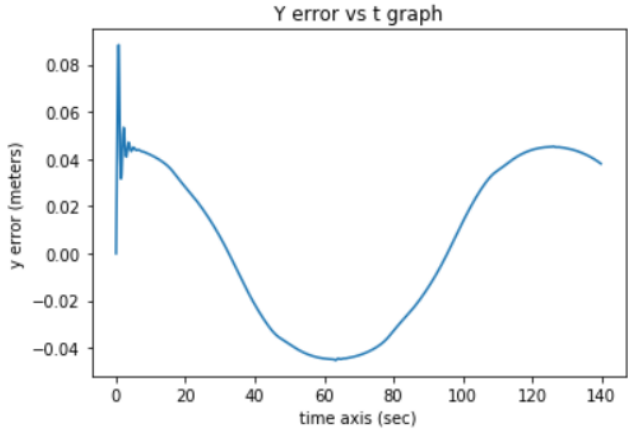




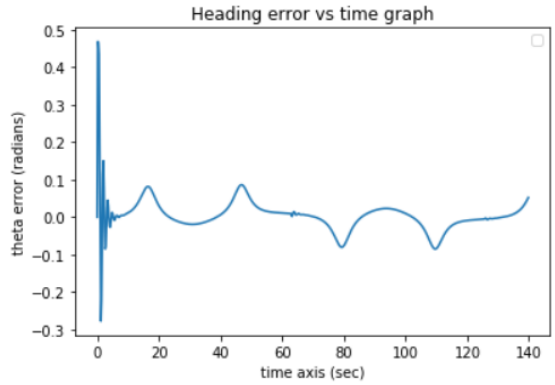




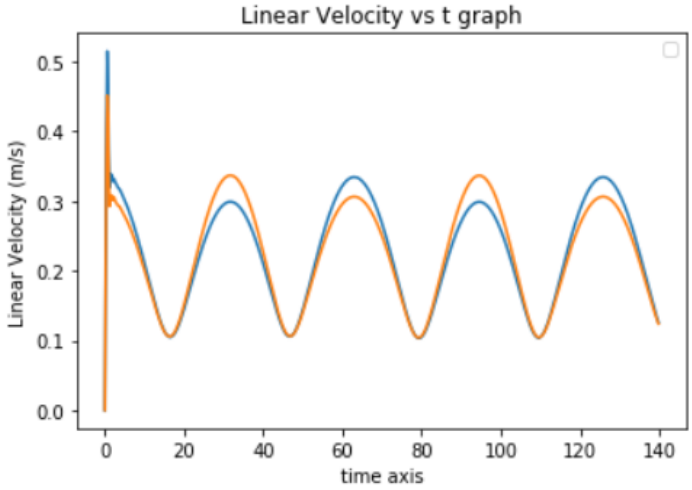
('The RMSE of x error is (in meters) :', '0.0718075444144')

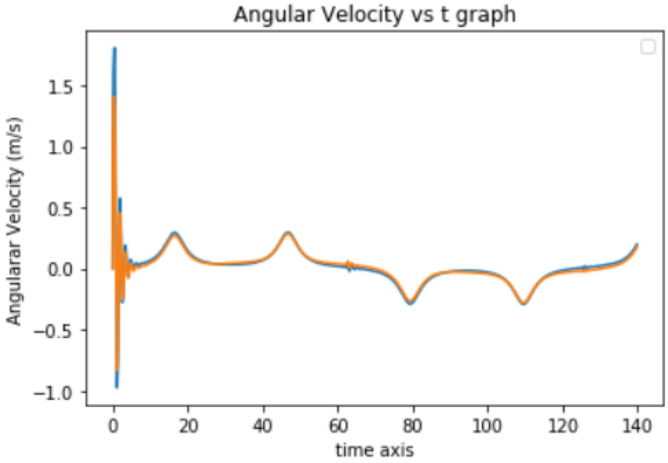


('The RMSE of y error is (in meters) :', '0.0349158097168')

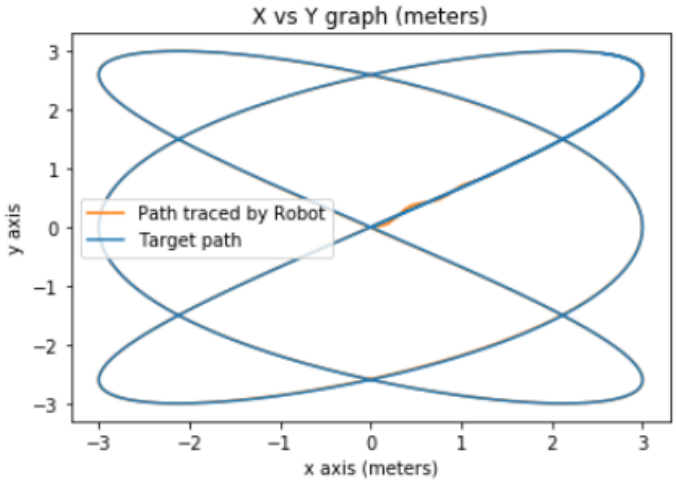


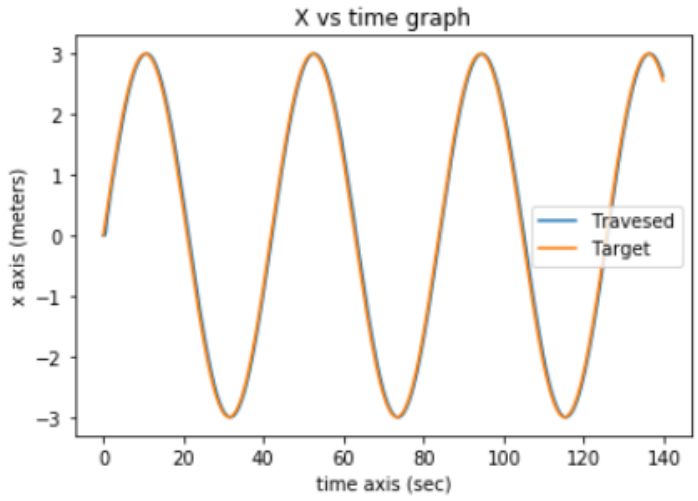
('The RMSE of heading error is (in meters) :', '0.0472764585063')

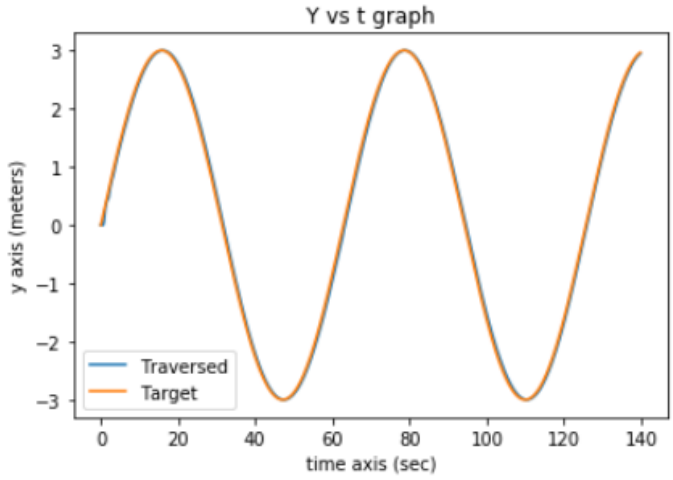


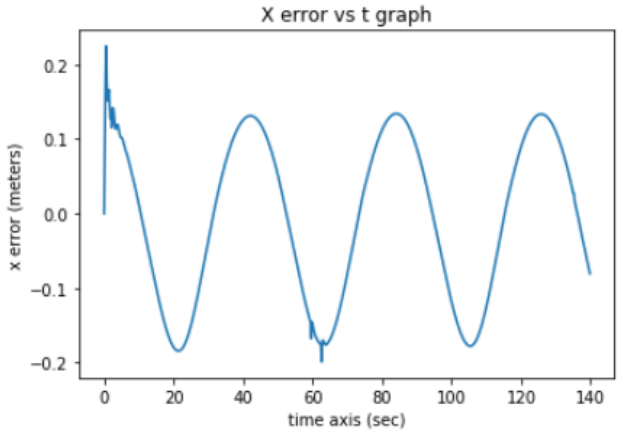


For the Second Lissajous curve.

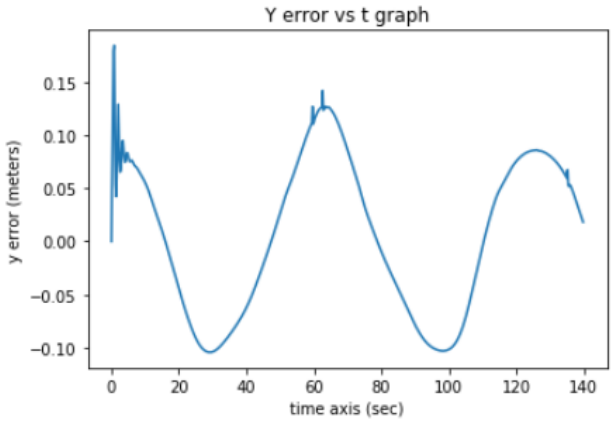




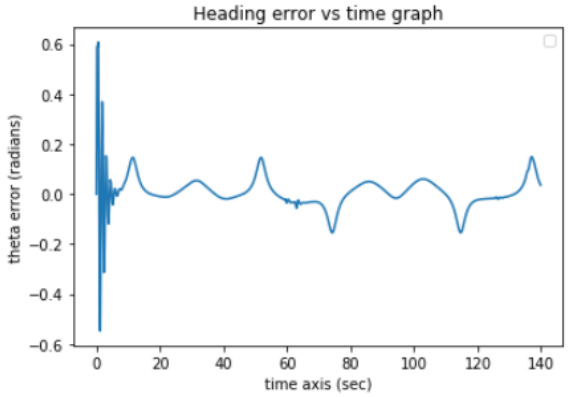




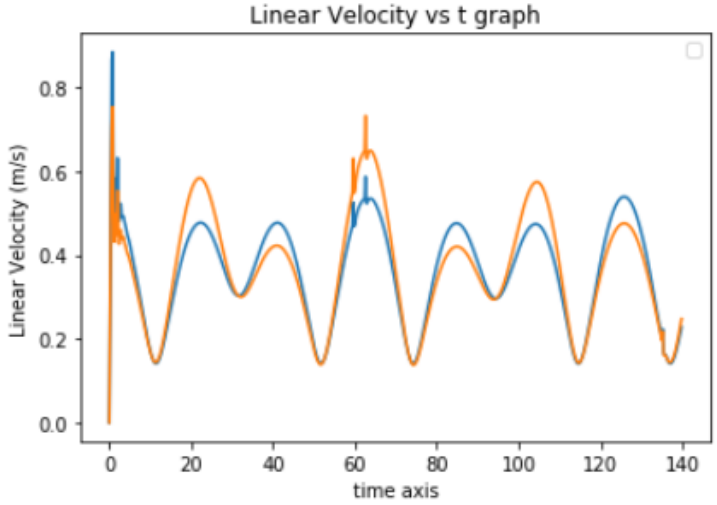
('The RMSE of x error is (in meters) :', '0.108350270138')

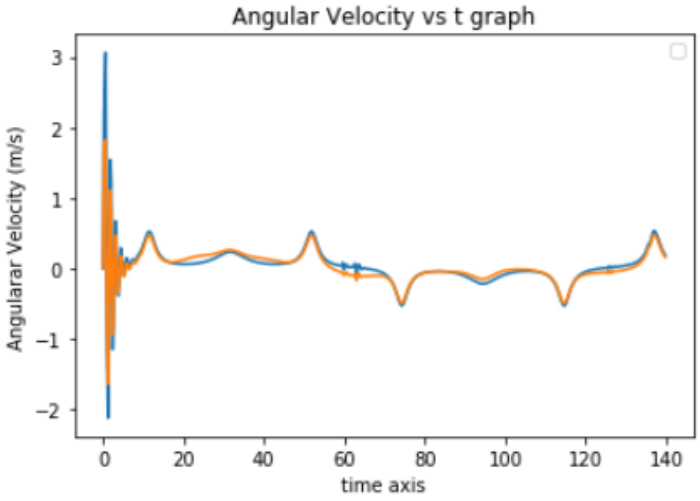


('The RMSE of y error is (in meters) :', '0.0734374059539')



('The RMSE of heading error is (in meters) :', '0.0742491505051')





0.5.5 Conclusion

Controller	Dist. RMSE (meters)	Heading RMSE (radians)	Time to finish (sec)
Non Linear Controller	0.0805	0.0491	124
Follower Robot	0.079550	0.04030	126

These are the performance metrics of a the reference alone and the tracing robot. Judging by the values we can clearly find the Non-Linear controllers perform very good. It evenly traces the trajectory indistinguishably, which is a good sign. And since the follower angular and linear velocity depends on the parameters of the reference also its execution is lagging by a few seconds, we can be assured that it follows the reference diligently. Although for the beginning of the 2nd Lissajous curve, there was a bit of wobble but pretty soon enough it became smooth.

Video link of the non-linear controller :
Infinity Curve: `https://drive.google.com/file/d/1-T6PsyRr00oK62QU2g3t3knDM-Lk9trh/view?usp=sharing`
Lissajous 2nd Curve: `https://drive.google.com/file/d/1TssLis_u48SsM8GVbIp3eMnusmdTaanQ/view?usp=sharing`

0.5.6 References

1. Wikipedia page for Lissajous: `https://en.wikipedia.org/wiki/Lissajous_curve`
2. Surveillance and Monitoring Strategies for Autonomous Mobile Agents by Mr. Aseem Vivek Borkar.
3. "Springer Handbook of Robotics" by Bruno Siciliano and Oussama Khatib.
4. Python documentation.