Robotics II Laboratory

Report on Task 02

Kinematics

SS 2020

Group-06_Task-02		**Matric.No**
Chaithanya Tenneti	-	1537943
Kaushik Goud Chandapet	-	1536199
Nishanth Iruthayaraj	-	1522044
Sai Radha Krishna Vangaveeti	-	1533846
Venkata Sai Tarak Padarthi	-	1536988

Date : 17.05.2020

FIGURE:

1. HOLONOMIC ROBOT:

Robot is said to be Holonomic robot if the controllable degrees of freedom of the robot is equal to the total degrees of freedom. So, the robot can move freely in any direction. Robots made with omni wheel drive which can move in any direction without any constraints are the best example for Holonomic drive[1].

2. INVERSE KINEMATICS OF MOBILE ROBOT:

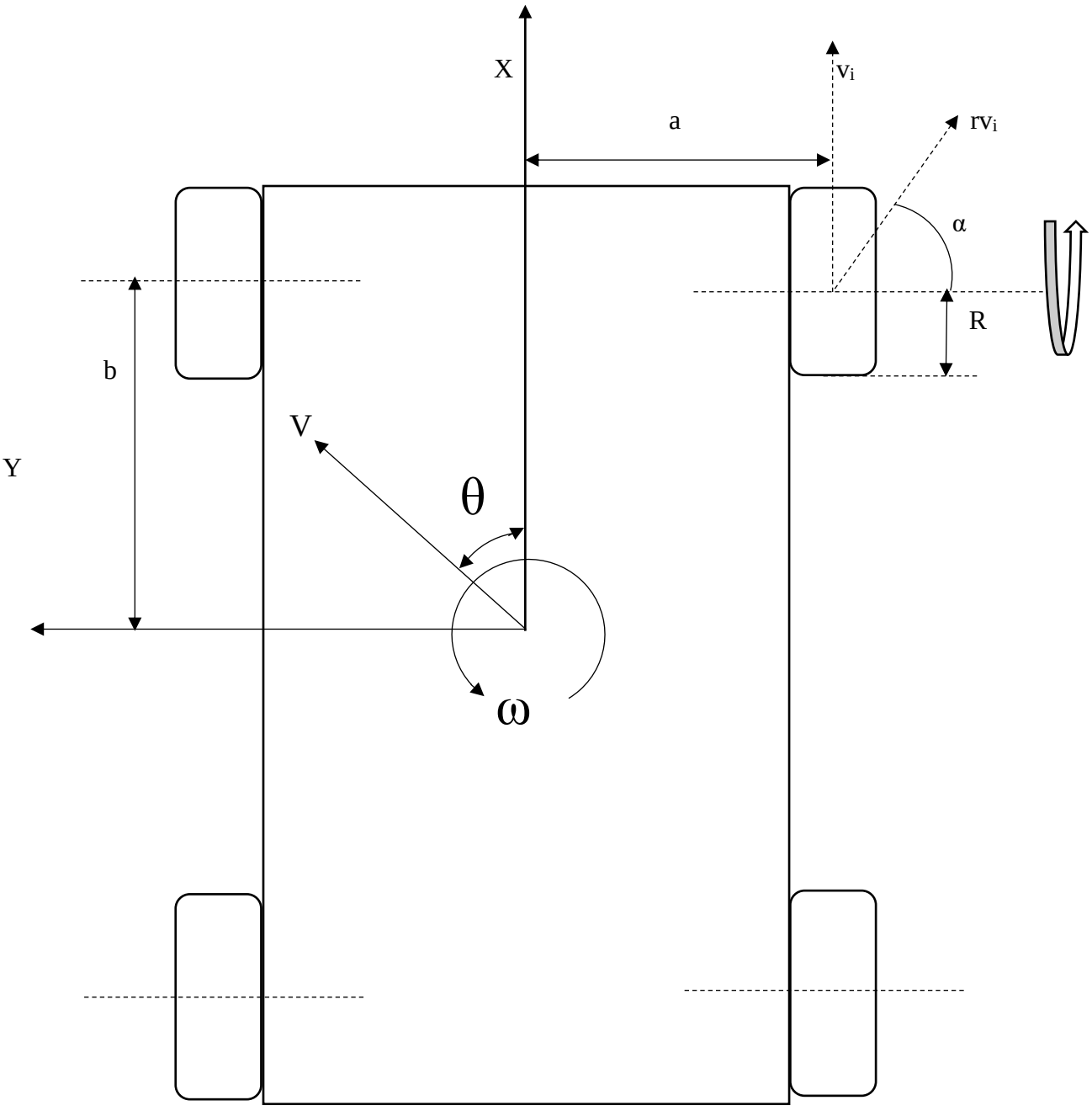

Figure 1: Co-ordinate system of mobile robots

Inverse kinematics of mobile robot can be derived from the movement vector of wheels. Considering v as velocity vector of mobile robot, the velocities v_x and v_y are the velocity of the robot in the direction of heading and perpendicular to the heading of the mobile robot respectively[2].

Considering ω as angular velocity of mobile robot, a and b as distance from center of robot to center of wheel and wheel axis respectively and θ as angle of robot in lateral direction.

Therefore, vector equation for the velocity of robot with respect to its coordinate system can be calculated by following equation,

$$v_i + r * v_i * cos(\alpha_i) = v_x - a_i * \omega \quad \ldots\ldots\ldots (1)$$

$$r * v_i * sin(\alpha_i) = v_y + b_i * \omega \quad \ldots\ldots\ldots (2)$$

v_i - *linear velocity vector of robot wheel*
rv_i - *linear velocity vector of roller*

By solving the above equations, linear velocity of wheels can be found by:

$$v_i\,(1 + r*cos(\alpha_i)) = v_x - (a_i * \omega)$$

$$r = \frac{v_x - (a_i * \omega) - v_i}{v_i * cos\alpha_i} \quad \ldots\ldots\ldots (3)$$

$$r = \frac{v_y + (b_i * \omega)}{v_i * sin\alpha_i} \quad \ldots\ldots\ldots (4)$$

Equating the above equation (3) and (4), we get

$$\frac{v_x - (a_i * \omega) - v_i}{v_i * cos\alpha_i} = \frac{v_y + (b_i * \omega)}{v_i * sin\alpha_i}$$

$$v_x - (a_i * \omega) - v_i = \frac{v_y + (b_i * \omega)}{tan\alpha_i}$$

$$v_i = v_x - (a_i * \omega) - \frac{v_y + (b_i * \omega)}{tan\alpha_i} \quad \ldots\ldots\ldots (5)$$

i as integer to define no of wheels we get the following values for different dimensions of robot wheels.

$$a_i = \{a, -a, a, -a\}$$
$$b_i = \{b, b, -b, -b\}$$
$$tan\ \alpha_i = \{1, -1, -1, 1\}$$

Taking v_1, v_2, v_3, v_4 as linear velocities of Front Left, Front Right, Rear Left, Rear Right of robot wheel respectively.

$$v_1 = v_x - v_y - a\omega - b\omega$$
$$v_2 = v_x + v_y + a\omega + b\omega$$
$$v_3 = v_x + v_y - a\omega - b\omega$$
$$v_4 = v_x - v_y + a\omega + b\omega \quad \ldots\ldots\ldots (6)$$

3. POSITION AND ORIENTATION CONTROL OF MOBILE ROBOT:

3.1 Discrete PID Controller[3]

The PID controller function is implemented in the discrete time form. The controller is programmed with algorithms for the P (proportional), I (integral) and D (derivative) and is tuned to move the robot to desired location.

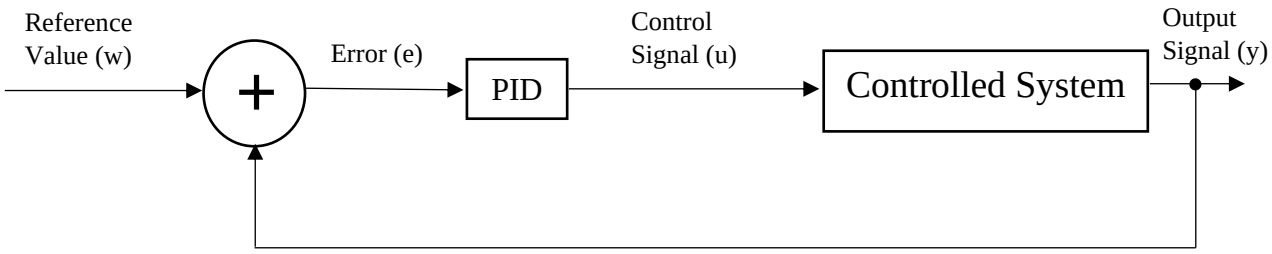

Fig. 2: Block diagram of PID controller

Based on the following three steps our controller has been tuned. They are,

- **Proportional tuning** involves correcting a target proportional to the difference. However there will be an error difference at the end.
- **Integral tuning** tries to remedy this by effectively cumulating the error result from the "P" action to increase the correction factor. So "I" attempts to drive the cumulative error to zero, which may also results in an overshoot.
- **Derivative tuning** is then used to minimize this overshoot by slowing the correction factor applied as the target.

PID control in a microprocessor can be realised by using the following difference equation (7).

$$y_k = K_p \,.\, e_k + K_i \sum_{i=0}^{k-1} e_i \,.\, \Delta t + K_d \,.\, \frac{e_k - e_{k-1}}{\Delta t} \qquad \text{.....(7)}$$

where,

y_k - *Output at k^{th} step*
e_k - *Error during k^{th} step*
e_{k-1} - *Error during (k-1)th step*
Δt - *Sample time*
K_p - *Proportional constant*
K_i - *Integral constant*
K_d - *Differential constant*

3.2 Implementing PID control for position and orientation of mobile robot:

Taking Reference (w), Control signal (u), Output (y), Error (e) with respect to mobile robot.

w : Target position of mobile robot
u : Velocity as input to mobile robot
y : Current position of the mobile robot
e : w-y (Target position - Current position)

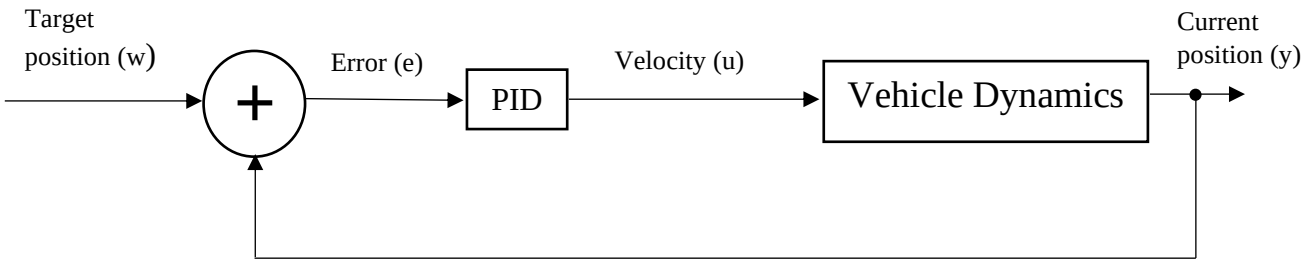

Fig. 3: PID control for position and orientation of mobile robot

Linear velocity along x, y direction and angular velocity are given as control signal to robot. Using the inverse kinematic equations (*), these velocity control signals are converted into angular velocity for each of the wheel. More details regarding these conversions are covered in section 4.

Pseudo code Implementation of discrete PID equations for position and orientation control of mobile robot is given below.

```
y = 0;                          // Output
error= 0;                       // Error during kth step
previous_error = 0;             // Error during (k-1)th step
total_error = 0;                // Cumulative error till kth step
t = 0.05;                       // Sample time


Loop {
error = ref - y;    // variable ref indicates the reference value
     y =  (kp * error) +
          (ki * t * (error + total_error)) +
          (kd * (error - previous_error) / t);

     total_error = total_error + error;
     previous_error = error;

}
```

4. INTEGRATION OF MATHEMATICAL MODEL AND CONTROLLER

A PID controller is implemented as per pseudo code for position in x and y directions and orientation γ (refer Fig. 4). Proportional (K_p), Integral (K_i) and Derivate (K_d) parameters are tuned for our robot and the values K_p , K_i and K_d are found to be 2, 0.01 and 0.2 respectively. An error threshold of 0.01 is considered for both position and direction.

```
errorx = x(i) - current_pos(1);
errory = y(i) - current_pos(2);
errorgamma = theta(i) - current_ori(3);

if (abs(errorx) < error_threshold &&
    abs(errory) < error_threshold &&
    abs(errorgamma) < error_threshold)
    break
end

px = proportional * errorx;
ix = integral * t * (errorx + total_errorx);
dx = derivative * (errorx - previous_errorx) / t;

py = proportional * errory;
iy = integral * t * (errory + total_errory);
dy = derivative * (errory - previous_errory) / t;

pgamma = proportional * errorgamma;
igamma = integral * t * (errorgamma + total_errorgamma);
dgamma = derivative * (errorgamma - previous_errorgamma) / t;

total_errorx = total_errorx + errorx;
total_errory = total_errory + errory;
total_errorgamma = total_errorgamma + errorgamma;

previous_errorx = errorx;
previous_errory = errory;
previous_errorgamma = errorgamma;

vx1 = px + ix + dx;
vy1 = py + iy + dy;
w1 = (pgamma + igamma + dgamma);
```

Fig. 4: PID control implementation for position and orientation of mobile robot

In our current world the robot axis is different from the inertial frame of reference so a transformation is applied for the velocities acquired from the PID controller. A 3×3 rotation matrix is used to transform the axes.

```
A = [cos(theta(i)) sin(theta(i)) 0; -sin(theta(i)) cos(theta(i)) 0 ; 0 0 1]*[vx1; vy1; w1];
vx = A(1,1);
vy = A(2,1);
w = A(3,1);
```

Fig. 5: Rotation matrix for transformation of velocities

By applying the Inverse kinematics, the linear velocities are then converted into the individual wheel velocities of the robot.

```
v1 = (vx - vy - (a+b) * w);
v2 = (vx + vy + (a+b) * w);
v3 = (vx + vy - (a+b) * w);
v4 = (vx - vy + (a+b) * w);
```

Fig. 6: Linear velocities of the individual wheels

SIMULATION GRAPH OF MOBILE ROBOT

The desired position is achieved by the mobile robot without any overshoot and steady state error. The position of mobile robot is recorded with respect to time in the Time-graph and also the position of the robot in XY-graph.

4.a: The following graphs (Fig. 7(a) and Fig. 7(b)) show the motion of robot from $p_0(0,0,0)$ to $p_1(1,0,0)$

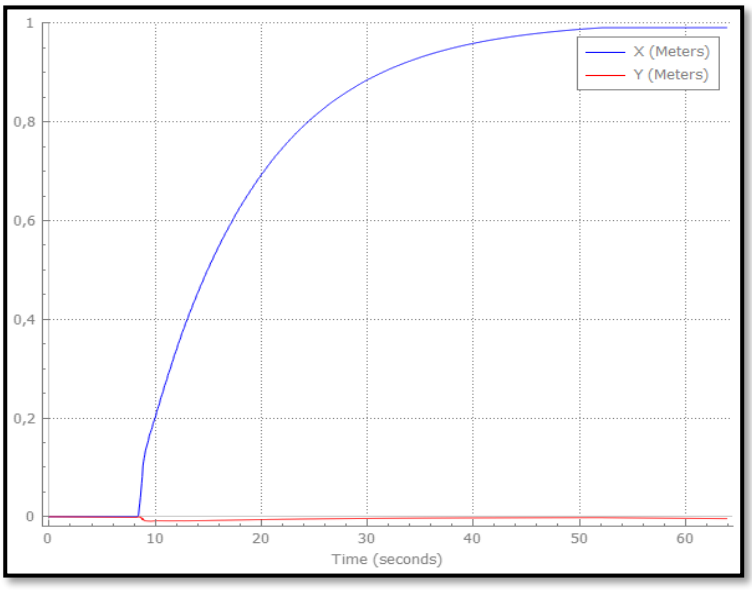

Fig. 7(a): Time –graph $p_0(0,0,0)$ to $p_1(1,0,0)$

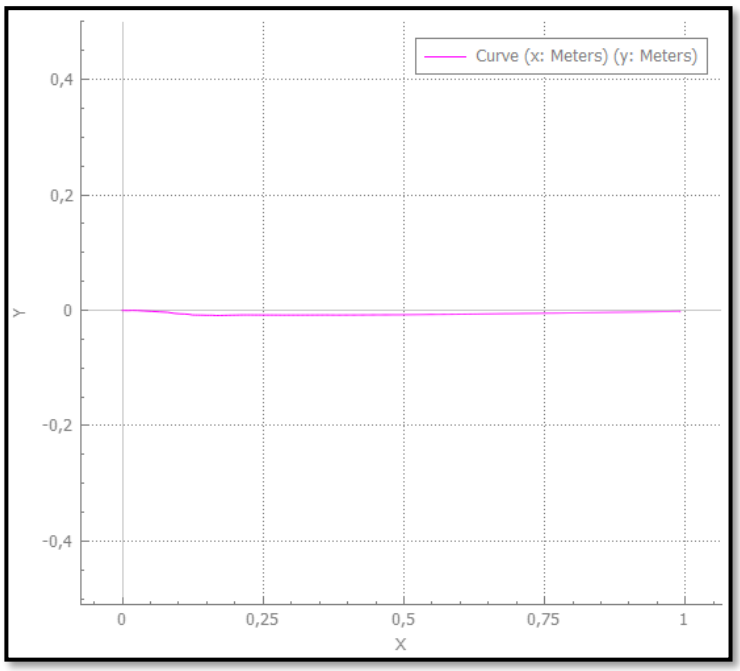

Fig. 7(b): XY – graph from $p_0(0,0,0)$ to $p_1(1,0,0)$

4.b: The following graphs (Fig. 8(a) and Fig. 8(b)) show the motion of robot from $p_0(0,0,0)$ to $p_2(0,1,0)$

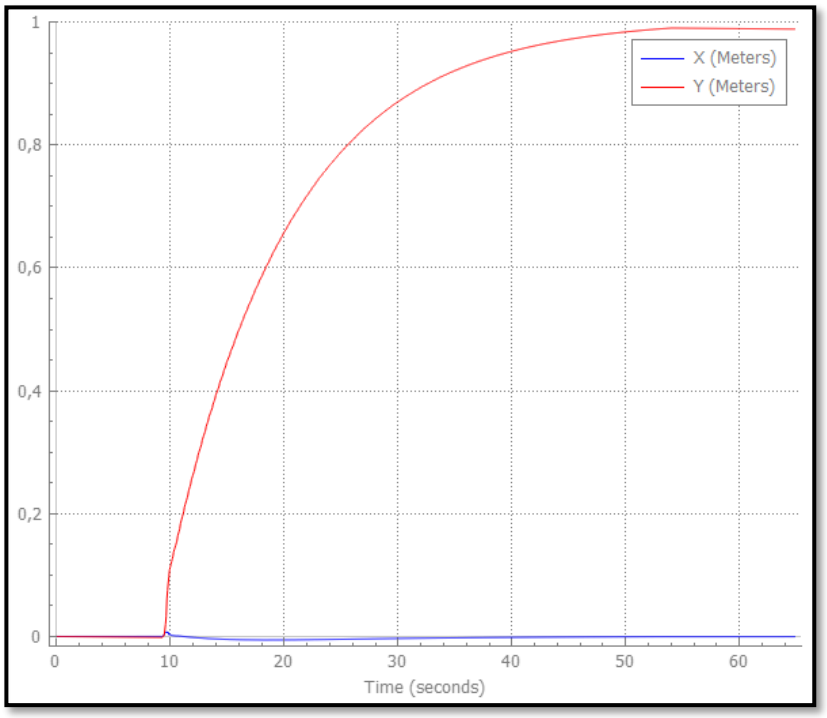

Fig. 8(a): Time – graph from $p_0(0,0,0)$ to $p_2(0,1,0)$

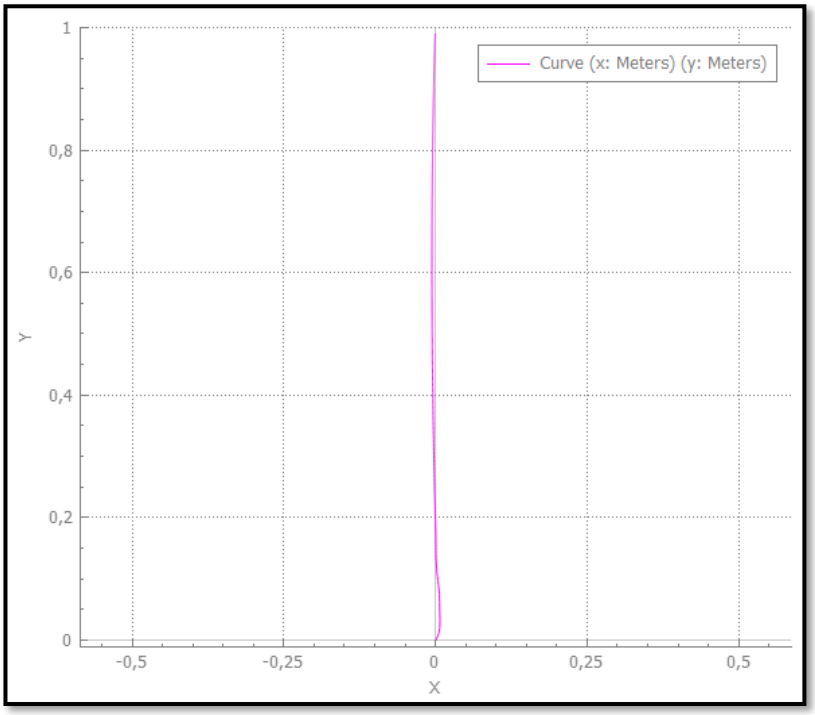

Fig. 8(b): XY – graph from $p_0(0,0,0)$ to $p_2(0,1,0)$

4.c: The following graphs (Fig. 9(a) and Fig. 9(b)) show the motion of robot from $p_0(0,0,0)$ to $p_3(1,1,0)$

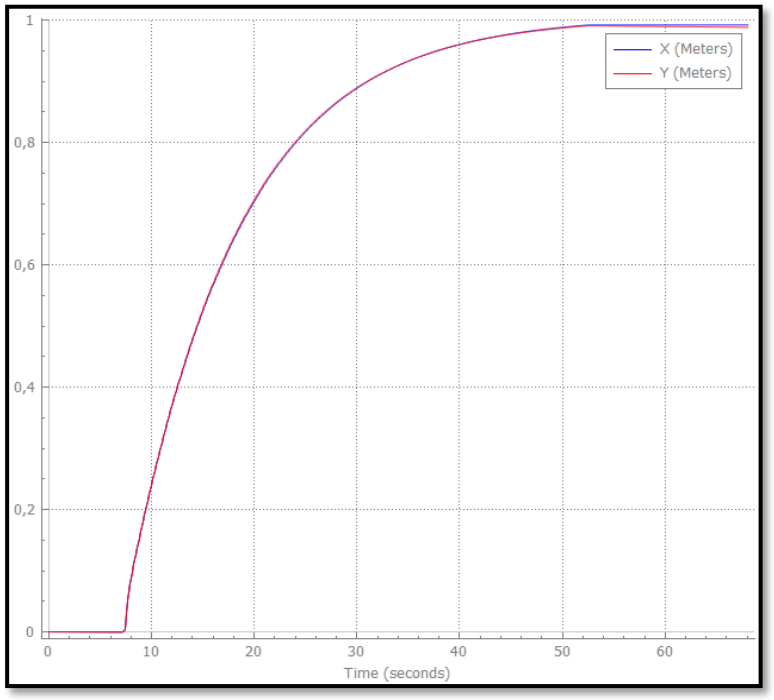

Fig. 9(a): Time – graph from $p_0(0,0,0)$ to $p_3(1,1,0)$

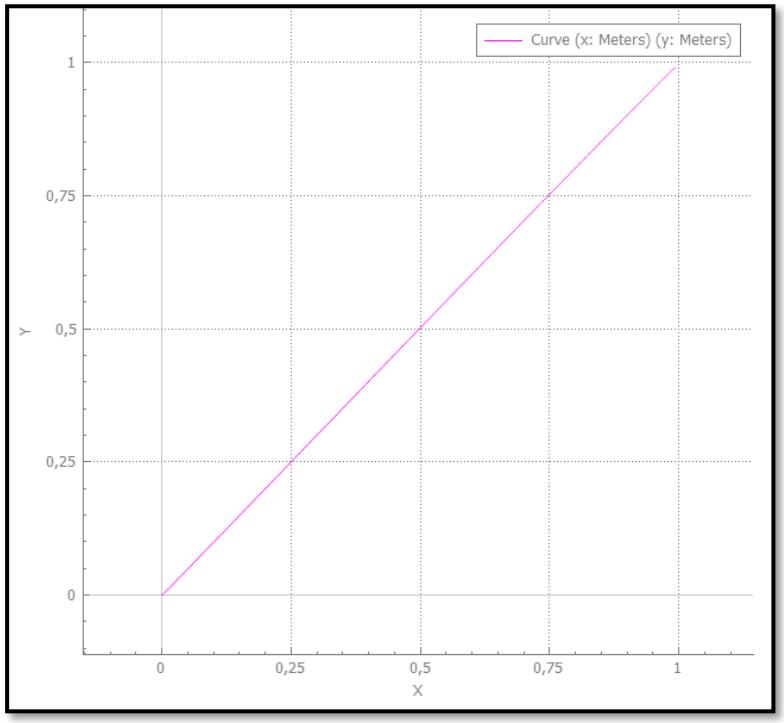

Fig. 9(b): XY – graph from $p_0(0,0,0)$ to $p_3(1,1,0)$

4.d: The following graphs (Fig. 10(a) and Fig. 10(b)) show the motion of robot between $p_0(0,0,0)$ - $p_1(1,0,0)$ - $p_2(0,1,0)$ - $p_3(1,1,0)$ - $p_0(0,0,0)$ without any intermediate points.

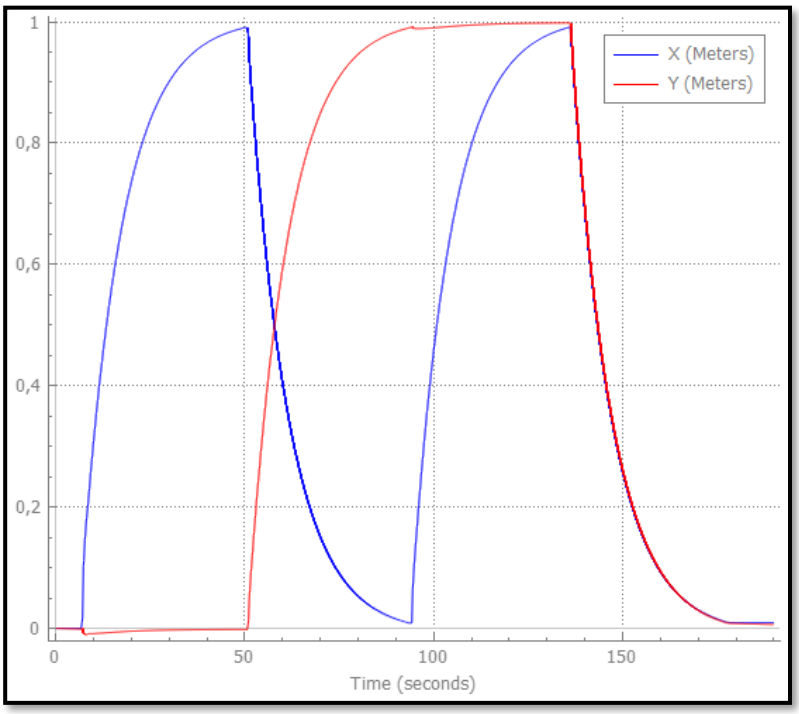

Fig. 10(a): Time – graph from $p_0 - p_1 - p_2 - p_3 - p_0$

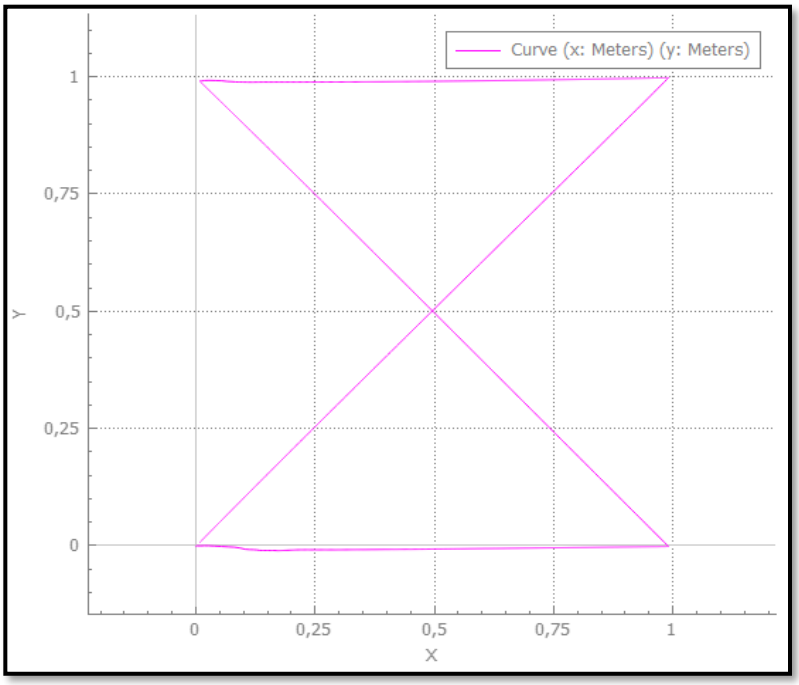

Fig. 10(b): XY – graph from $p_0 - p_1 - p_2 - p_3 - p_0$

4.e: To avoid a standstill position of robot before proceeding to a next point, we are using an intermediate point of 0.2 between the given way points and also an error threshold of 0.01 to determine when to move to the next waypoint. By using the above mentioned values, relatively smooth transition between waypoints is achieved at the end. The following graphs (Fig. 11(a) and Fig. 11(b)) show the motion of robot from $p_0 - p_1 - p_2 - p_3 - p_0$ with intermediate points and threshold.

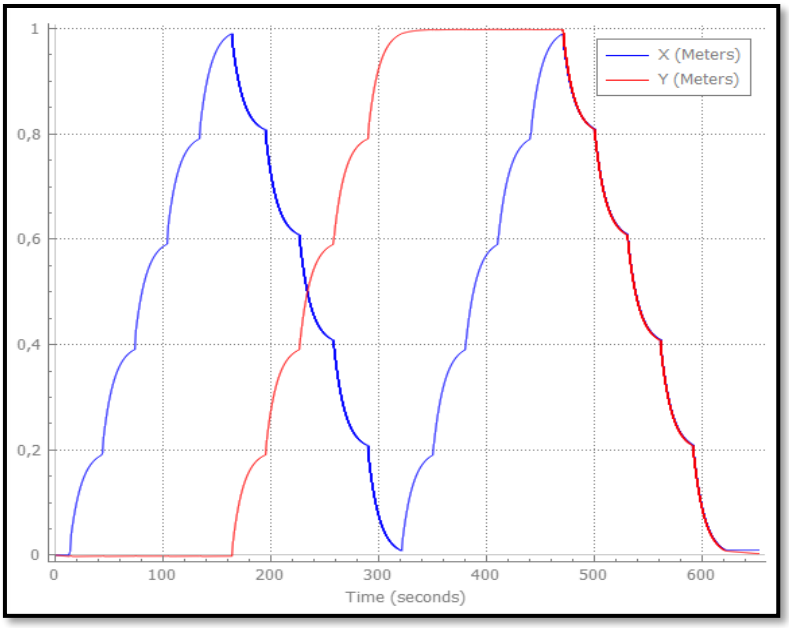

Fig. 11(a): Time – graph from $p_0 - p_1 - p_2 - p_3 - p_0$ with intermediate points and threshold

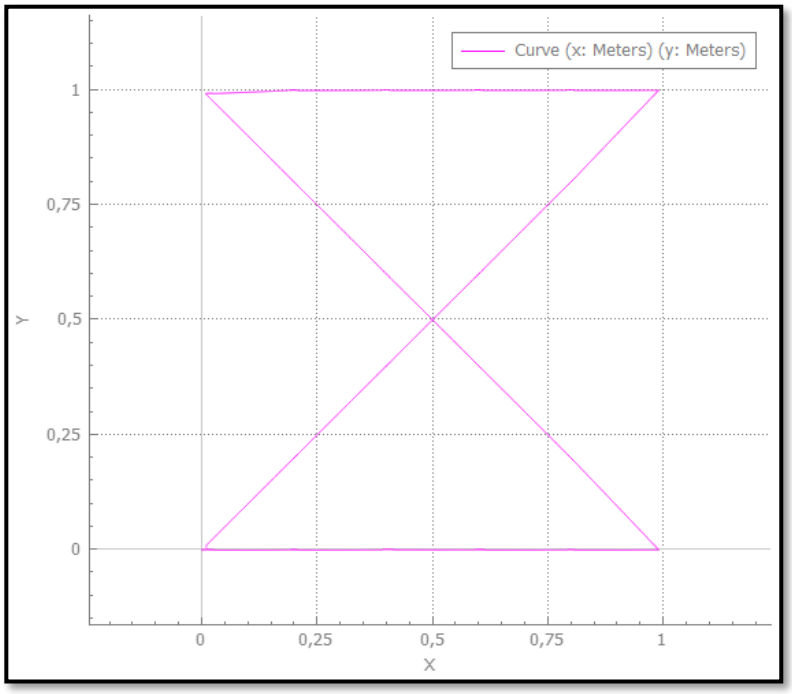

Fig. 11(b): XY – graph from $p_0 - p_1 - p_2 - p_3 - p_0$ with intermediate points and threshold

4.f: To avoid collision, we are generating the path manually for the robot to move on maze and also changing the orientation of robot when required and upon achieving the final desired point, we are making the robot to celebrate by spinning twice in clockwise and counter-clockwise direction. The following graphs (Fig. 12(a) and Fig. 12(b)) show the motion of robot on maze.

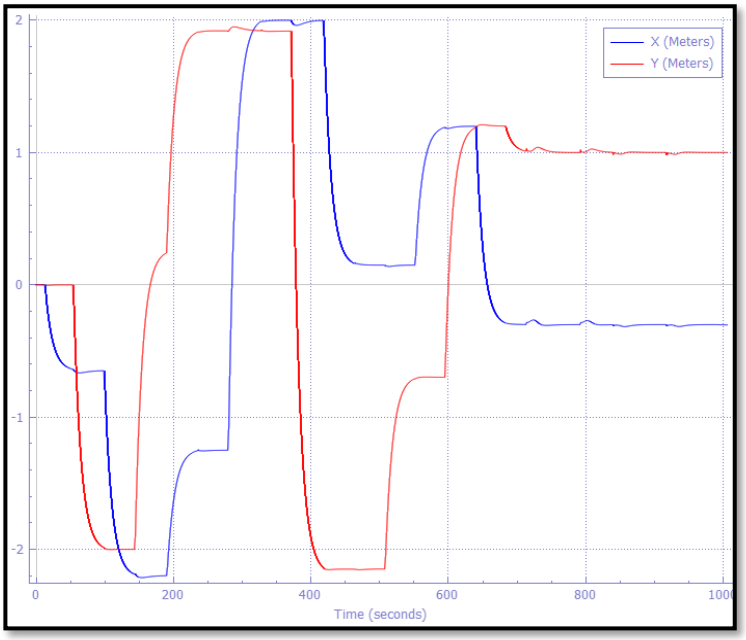

Fig. 12(a): Time – graph of robot on maze

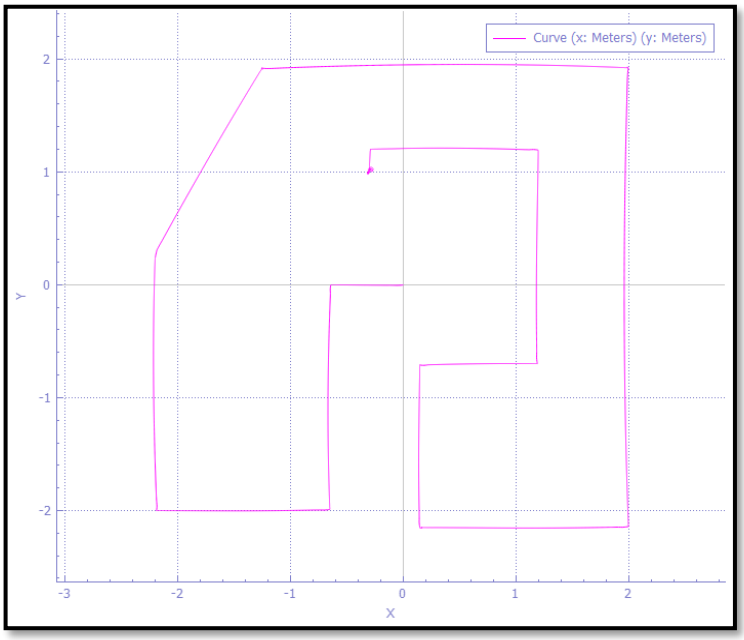

Fig. 12(b): XY – graph of robot on maze

4g: The robot is controlled to drive around the circular obstacle with radius of 1.8m. The robot moves around the obstacle by generating its own path using the following terms,

$$\theta = 0 \text{ to } 360; \quad x = r*\cos\theta; \quad y = r*\sin\theta$$

The distance between obstacle centre and robot centre (r) = 2.1m

The following graphs (Fig. 13(a) and Fig. 13(b)) show the motion robot around the circular obstacle.

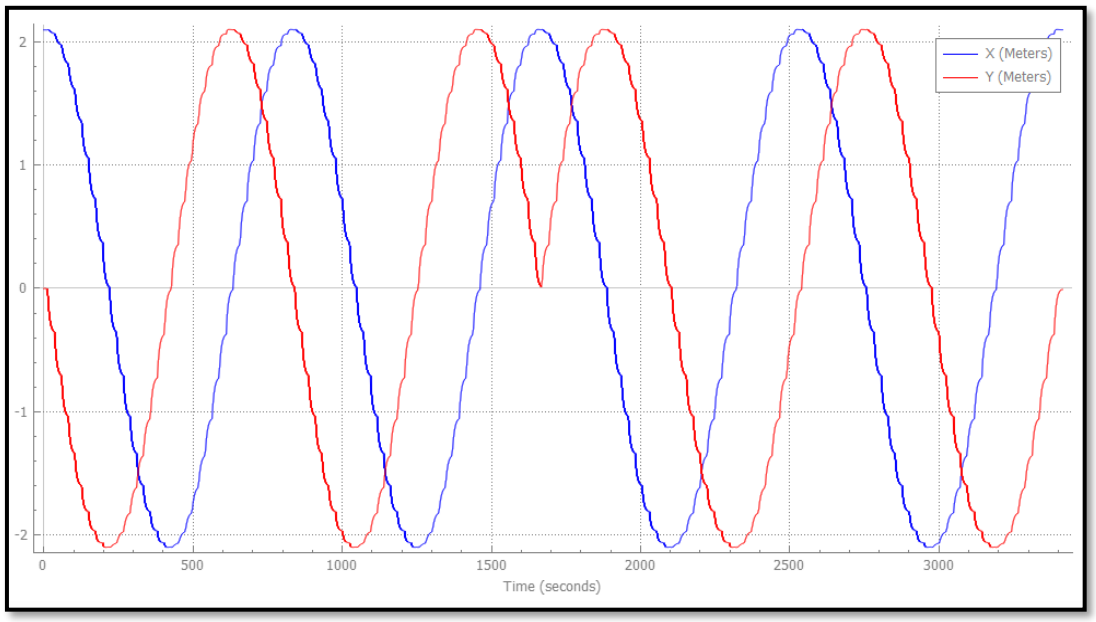

Fig. 13(a): Time – graph of robot around circular obstacle

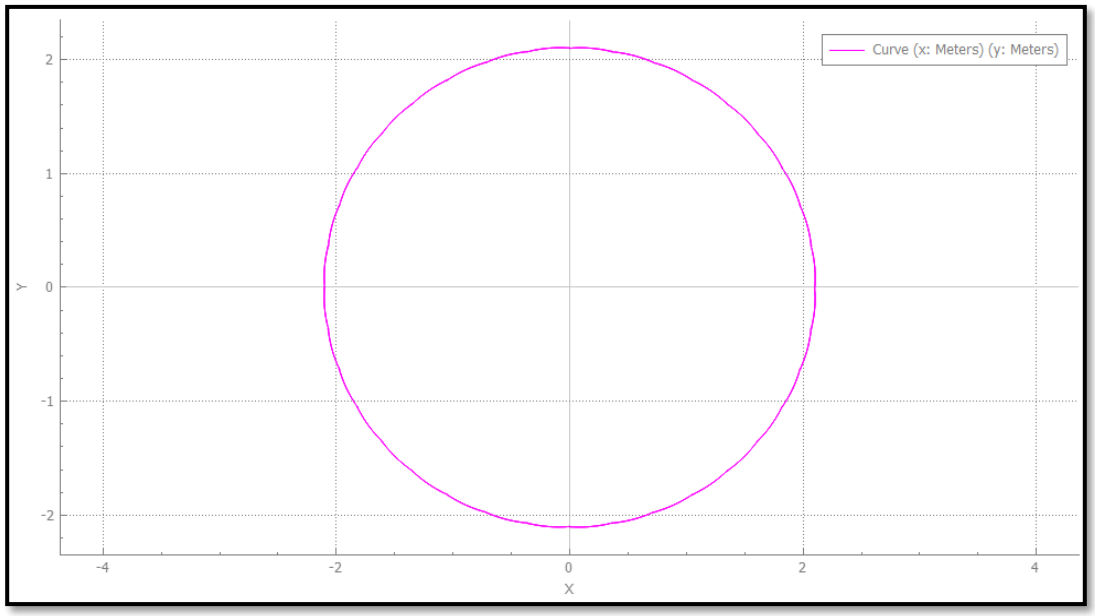

Fig. 13(b): XY – graph of robot around circular obstacle

References

1. http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html.
2. E. Maulana, M. A. Muslim and V. Hendrayawan, "Inverse kinematic implementation of four-wheels mecanum drive mobile robot using stepper motors," 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), Surabaya, 2015, pp. 51-56, doi: 10.1109/ISITIA.2015.7219952.
3. https://www.autonomousrobotslab.com/pid-control.html