TurtleBot Path Tracking using PID Controller

Nagarjun Vinukonda, Rishi Madduri, Rishabh Chadha, Vrushabh Desai

***Abstract*—An optimal control techniques well known in control community is proposed i.e. PID controller to address the problem of path tracking in autonomous vehicles. In this project we propose PID controller for turtle bot Navigation in order to determine steering control at each instant of time. This project aims to demonstrate the the control performance when robot attempts to follow a path.**

***Index Terms*—PID Controller, simulation, gazebo, ROS.**

I. Introduction

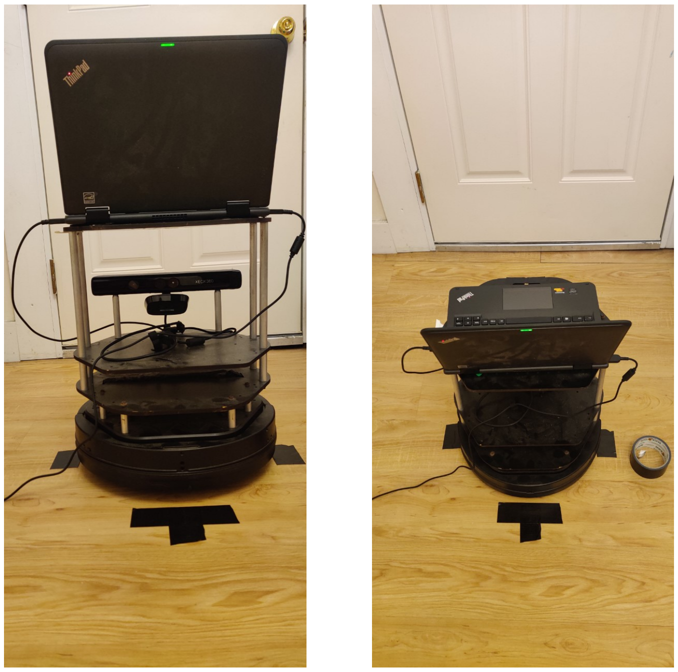
There are multiple control methods available to control robot which depends on application specific and path planning done.

The robot controls field is exceptionally advance and grows with more exiting new research. Many researchers have invented interesting projects, for an instance, Khnissi et al, implemented global asymptotic stability for robot tracking tasks using Neural Network controllers while navigating mobile robot [1]. Parra-Vega et al, have proposed a simple decentralized continuous sliding PID controller [2].

All these experiments and ventures require a software platform to develop and test for which a simulation tool called Gazebo is used from ROS platform. ROS is widely used framework in field of robotics for writing robust robot software using a collection of tools and libraries that facilitate the end goal of robot control.

Turtlebot is a differential drive mobile robot that exists as a personal robot kit with open-source software. The project is intended to allow users to have an inexpensive means of exploring ROS, implementing control systems for learning and for fun. In the absence of real hardware, the software that is generated for the Turtlebot in ROS can be used in the Gazebo, later after testing in software, hardware implementations can be done.

Path Tracking is a considerable problem in autonomous industries taking over past few years for which many control methods for path tracking have been developed considering the nonholonomic constraints of autonomous vehicles. PID has advantages that include robustness and simplicity for which best choice of PID parameters are searched for different process models. In this project we are using closed loop ROS Turtlebot as our Kinetic Model as shown in Fig.(1). It is assumed that our kinetic Model (Turtle bot) moves without slipping on a plane, that means there is a pure rolling contact between the wheels and the ground and also there is no lateral slip between the wheel and the plane. Its Kinematic analysis can be conducted by using Cartesian coordinates.



(a) Front view (b) Top view

Fig. 1: Turtlebot

II. Literature Survey

The field of wheeled mobile robots has seen a dramatic rise in applications across multiple domains, in the last decade. Tasks and areas that can endanger human life can now be accomplished by these wheeled robots. However, a robust control system is required to deploy these robots and ensure a safe human robot interaction. Chang et al. in [3] present a simple way of implementing an adaptive tracking controller based on PID for mobile robot trajectory tracking. They apply a classical PID approach for a path-following controller. They have used a non-linear model of mobile robot kinematics to perform an accurate trajectory prediction. They construct their control law based on Lyapunov stability and consists of a parallel structure PID controller with fixed gain. They also perform a computer simulation for a differentially driven non-holonomic mobile robot.

In [3] Mayyahi et al. propose an optimal control technique that addresses the problem of path tracking. They utilize a Fractional Order Proportional Integral Derivative controller to control an autonomous ground vehicle. They track the behavior of a predefined reference path. In their work, they design two such controllers. And utilize the input torque to manipulate the vehicle to obtain path. In comparison with [4], [3] uses both Kineamtic and Dynamic model instead of just Kineamtic Model. To further optimze the FOPID parameters, the authors of [4] use a particle swarm optimization algorithm. Normey-Rico et al. [5] propose a robust PID controller for path tracking. They use a linearized model for the mobile robot which consists of an integrator and a delay system. They have tested their PID controller on synchro-dirve mobile robot and have shown good performance. The paper by Luo et al. [6] proposes a rule-based expert control PID algorithm. It combines position error and error rate of change output to correct PID parameters. They perform simulation in MATLAB and prove an improved dynamic performance, adaptability and a better tracking function.

III. Dynamic Model and Control

Most indoor mobile robots do not move like a car. Our Turtlebot is differential drive bot. There are two main wheels of equal radius, each of which is attached to its own motor. To construct a simple model of the constraints that arise from the differential drive, let us assume a distance L between the two wheels, and the wheel radius, r as shown in Fig.(2).

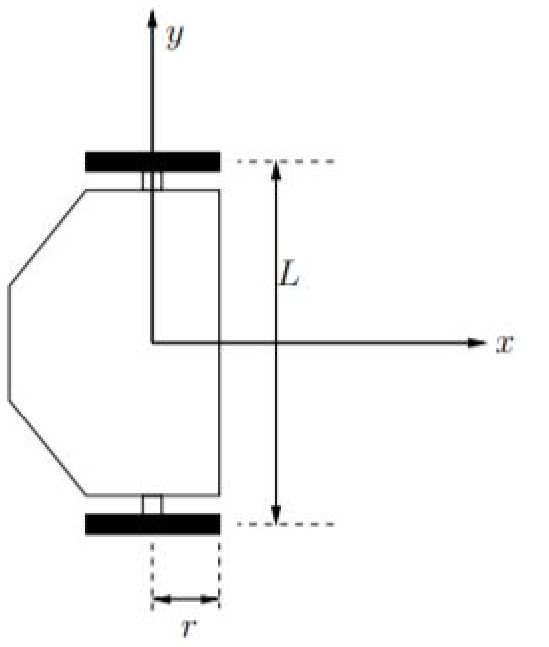


Fig. 2: The parameters of a generic differential-drive robot

While the robot is moving based on the observations, the transition equations are:

$$\dot{x} = r/2 * (u_l + u_r) * cos(\theta) \tag{1}$$

$$\dot{y} = r/2 * (u_l + u_r) * sin(\theta) \tag{2}$$

$$\dot{\theta} = r/L * (u_r - u_l) \tag{3}$$

In the above equations u_r and u_l are right and left wheel velocities, x and y are position coordinates of robot and $\dot{\theta}$ is angular velocity.

In the Fig.3 provided, it represents various subsystems that comprise the trajectory planning of an autonomous driving system and its controller. The desired position and velocity is fed into the different controller (i.e. PID) as its input. Using these inputs and feedback from the system, the controller produces an output signal that represents the force that must be applied to the system (in the form of motor torque). The goal of this project is to demonstrate the effect that a controller has when a robot attempts to follow a path, which will be explained in detail below.

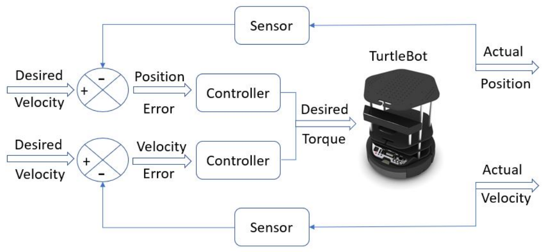


Fig. 3: Turtlebot Control Structure

IV. PID Controller

The desired output for the mobile robot is its position while applying position control which is a PID controller. An error is calculated on the position when control is applied, and this is used to compute a voltage that is sent to the motors, which ultimately drives the motors toward a desired position. The control law for a PID controller is listed below in equation (4).

$$\tau = K_p\tilde{q} + K_v\dot{\tilde{q}} + K_i \int_{v}^{t} \tilde{q}(\sigma)d\sigma \tag{4}$$

τ represents the torque, $\tilde{q}$ is the position error, $\dot{\tilde{q}}$ is the velocity error, and $\tilde{q}(\sigma)d\sigma$ is the integral term obtained by integrating, with respect to time and the position error term. Kp, Kv, and Ki represent the proportional, derivative and integral gains, respectively.

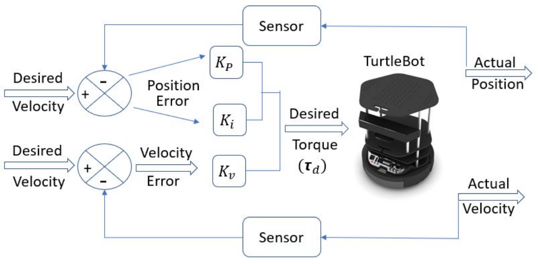


Fig. 4: Turtlebot PID control diagram

As shown in Fig.4, the PID Control block receives the desired position and desired velocity of the wheel. The goal of the PID block is to give the mobile robot the motor torque required to achieve the desired position and velocity.

Note that the role of the integral component in a PID controller is to make the steady state error in the PD control system go to zero and the role of the derivative component is to dampen the response, acting similar to friction. As mentioned, the desired terms are received and are subtracted from the actual positions and velocities to create the position and velocity errors. The calculated torques are fed into the TurtleBot block, which produces the actual position and velocity for the system.

V. Methodology

A. Coding Libraries

We are using ROS(Robot Operating System) for TurtleBot3 Navigation, Rviz for visualisation and Gazebo simulation tool. We are using C++ as our programming platform.

B. Pseudo code:

Algorithm 1: General PID Controller implemenation

```
Generate a random start node
Set Goal node
Set orientation range w.r.t start node
Set initial Kp,KI,Kd values
while not reached goal: do
    Calculate distance Goal position
    Calculate change in angle
    Multiply error to Kp,Kd,Ki and keep updating
end while
When Goal has reached:
Calculate error in orientation and set the angle
```

C. Working method:

We implemented a closed Loop PID controller for which both position and orientation tracking is done while navigating the turtlebot3.

1) Proportional Error Kp:
It is the Euclidean distance between goal position and start position.

$$d = \sqrt{(goal_x - start_x)^2 + (goal_y - start_y)^2} \tag{5}$$

2) Integral Error KI:
It is calculated by updating distance at each interval. Where D_t is Total distance travelled.

$$D_t = D_t + d \tag{6}$$

3) Derivative Error Kd:
It is the difference between the current node and previous node. Where D_p is previous dtance to node.

$$D_D = d - D_p \tag{7}$$

Similarly, error is calculated for orientation of turtlebot while navigating.

4) Turn Angle:
It is the angle turned by the turtlebot to travel from point 1 to point 2

$$\phi = \arctan \frac{y_2 - y_1}{x_2 - x_1} \tag{8}$$

VI. Experiments

The below figure shows our results while navigating our turtle bot in Rviz. We used trail and end method for tuning and we can tune further our PID parameters in order to reduce the error. In the figure we are navigating our turtlebot through pentagon trajectory and the path followed by bot is visualized in red color. Following the success of our simulation, we deployed our PID Controller on a Turtlebot.

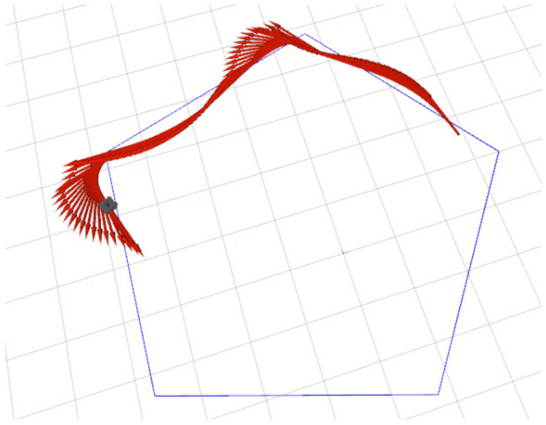
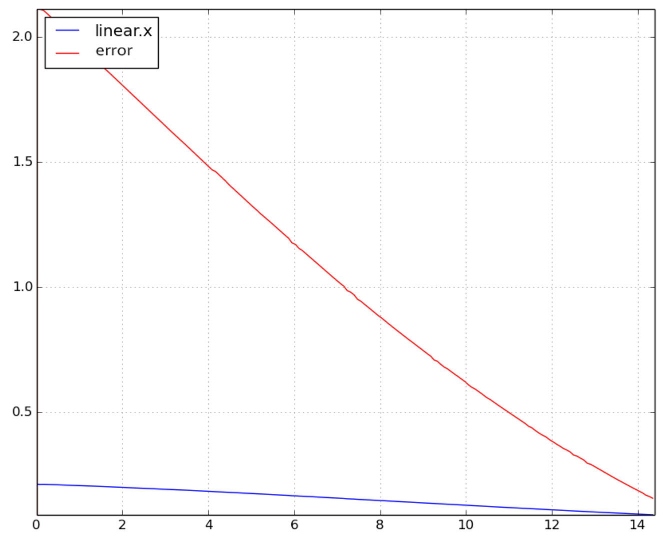


Fig. 5: Turtlebot Navigation on pentagon path

VII. Results and Analysis

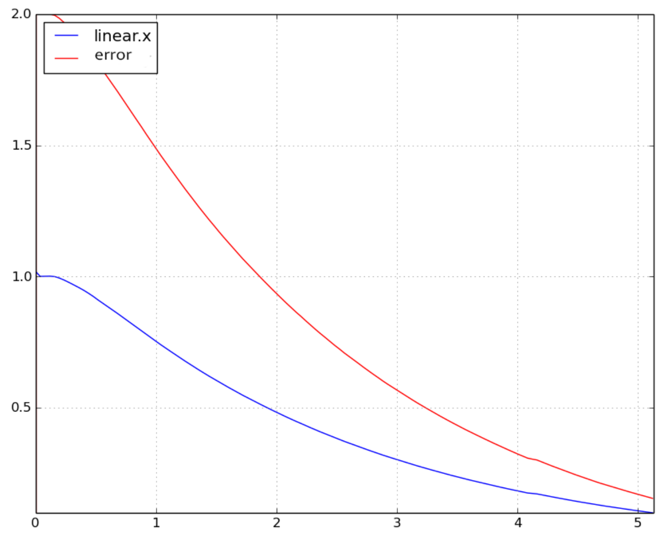
We have navigated our turtlebot in a closed environment. The following are the results of our experiment. We have conducted our experiment into two parts: Single Goal and Multi-Goal experiment.

We have tuned our K_p, K_d, K_i values using trail and end method. While Navigating our turtlebot in single goal experiment, we set our initial K_p, K_d, K_i values as 0.1, 0.001, 0.001 respectively and it resulted as Fig.6, where the error is decreasing with time, as well as linear velocity when reached goal. We have tuned our parameters to K_p, K_d, K_i values as 0.3, 0.001, 0.0001 respectively and resulted Fig.7. Similarly we have increased K_p, increased K_d and decreased K_i as 0.5, 0.01, 0.0001 respectively and resulted Fig.8. As observed



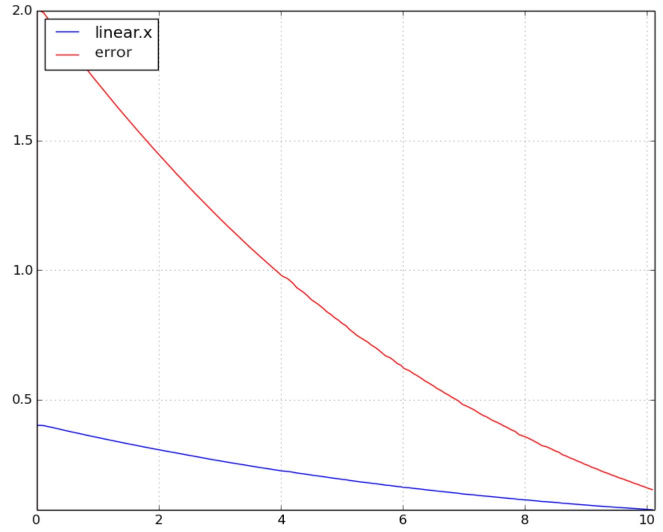
Linear.x is the linear velocity of turtlebot.

Fig. 6: PID error vs time



Linear.x is the linear velocity of turtlebot.

Fig. 8: PID error vs time



Linear.x is the linear velocity of turtlebot.

Fig. 7: PID error vs time

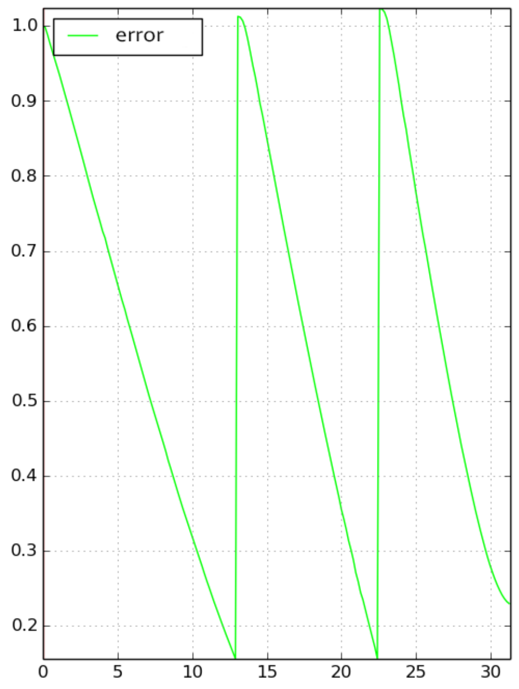


Fig. 9: Position error vs time

from graphs as the error decreases the time taken to reach the goal decreases, therefore increase in velocity.

In the second experiment, we kept multiple points as our goal as shown in the video recorded. The graph Fig.9 shows position error vs time. As there are multiple goals available, the error decreases with time when reached the goal and the sudden spike in graph is due to started of new goal point, which is different from previous goal point.

The Fig.10 and Fig.11 shows the change in theta error w.r.t time. As the robot follows straight path from start node to first goal near start node, there is no change in error in angle (i.e. the curve is flat), but as soon as the robot sets its goal to a new point, the change in angular error decreases in both directions w.r.t time with our tuning parameters.

VIII. Future Scope and Conclusion

The PID controller was the most effective in terms of maintaining a small steady-state error and providing a very fast response time. The addition of the integral controller helped to reduce the steady-state error, as it sums small amounts of error over time and accommodates for them with corrections when they've become significant enough to require correction.

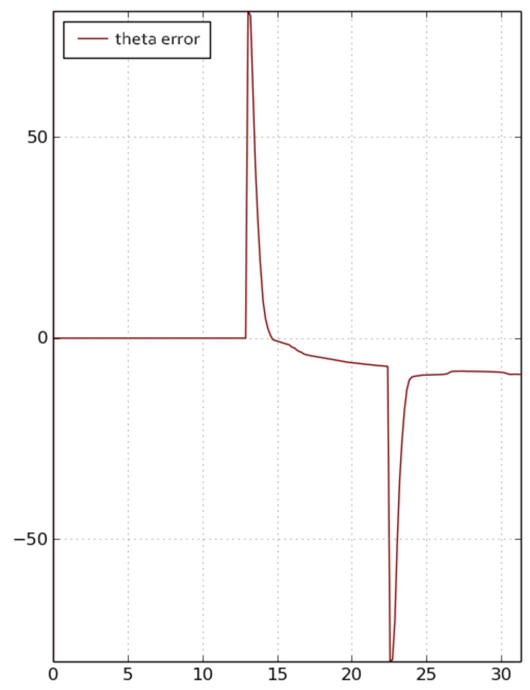


Fig. 10: Angular error vs time

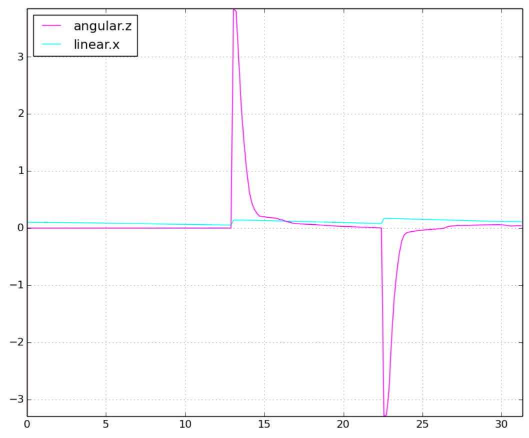


Fig. 11: Angular error vs time

In this project we have successfully implemented PID control on turtle bot both on Hardware and Software. From this experiment we have gained knowledge on ROS and its tools and experience on hardware implementation. Project work was distributed evenly among team members to ensure that each individual was actively participating in group work. Through tuning in experiment we found, increase in K_p, increase in K_d and decrease in K_i will reduce error effectively as discussed in class.

In Future we can work on to tune the gains and perform more testing to ascertain the effect of the tuning on the steady-state error, overshoot, and response time of the controller.

References

[1] K. Khnissi, C. Seddik, and H. Seddik, "Smart navigation of mobile robot using neural network controller," in *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. IEEE, 2018, pp. 205–210.

[2] V. Parra-Vega, S. Arimoto, Y.-H. Liu, G. Hirzinger, and P. Akella, "Dynamic sliding pid control for tracking of robot manipulators: Theory and experiments," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 6, pp. 967–976, 2003.

[3] H. Chang and T. Jin, "Adaptive tracking controller based on the pid for mobile robot path tracking," in *Intelligent Robotics and Applications*, J. Lee, M. C. Lee, H. Liu, and J.-H. Ryu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 540–549.

[4] A. Al-Mayyahi, W. Wang, and P. Birch, "Path tracking of autonomous ground vehicle based on fractional order pid controller optimized by pso," in *2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2015, pp. 109–114.

[5] J. E. Normey-Rico, I. Alcalá, J. Gómez-Ortega, and E. F. Camacho, "Mobile robot path tracking using a robust pid controller," *Control Engineering Practice*, vol. 9, no. 11, pp. 1209 – 1214, 2001, pID Control. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967066101000661

[6] Z. Luo and W. Li, "Tracking of mobile robot expert pid controller design and simulation," in *2014 International Symposium on Computer, Consumer and Control*, 2014, pp. 566–568.