ME 5751 Module 1: Proportional Control for Point Tracking

Erik Duque
Cal Poly Pomona
Pomona, CA
eduque@cpp.edu

Jonathan Jenkins
Cal Poly Pomona
Pomona, CA
jjenkins@cpp.edu

Abstract*—This project looks at creating a proportional controller for a 2 wheeled "Roomba" style robot to track a point. The methods used to test the controller involve a gui (graphic user interface) created by Dr. Chang which simulates the robot. The mathematics involved were sourced from "Introduction to autonomous mobile robots", 2nd Ed. by Siegwart, Nourbakhsh, and Scaramuzza. The findings in this project were each k value had a slightly different effect on the robot trajectory as a whole. Kp effects the speed of the robot, Ka effects the angle of attack from the current position to the next, and lastly Kb effects how aggressively the robot oriantes itself to the desired position.

Keywords*—Proportional Control, Trajectory, Motion Planing, Error, Proportional gain,

I. INTRODUCTION

In modern robotics control theory is a critical area of focus, without accurate articulation, navigation, and control many robots would not be able to function in real world environments. Modern robotics implements a wide variety of different control methods, the goal of this lab is to explore proportional control. Proportional control is a type of linear feedback control system in which a correction is applied to the controlled variable, and the size of the correction is proportional to the difference between the desired value and the measured value

In this experiment proportion control is used to control a robot navigation route in a virtual environment. As shown in figure 1, a robot will independently navigate to 4 different waypoints. The robot will use a custom proportional controller to influence the linear and angular velocity until it reaches the desired waypoint.

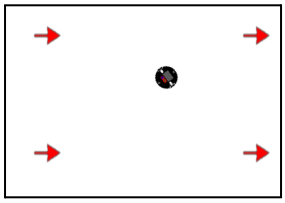


Figure 1. Waypoint Map

This paper will go into depth on different elements of the experiment including control theory, robot kinematics, programing script, controller improvements and evaluation of the final robot motion.

II. BACKGROUND

Virtual simulation is used to assist in many robotic development processes. We will be using a custom virtual environment to build and test a proportional control algorithm.

Proportional control (P control) is a type of linear feedback control system in which a correction is applied to the controlled variable. The size of the correction is proportional to the current error of the system. Error (e) is the difference between the desired value and the measured value. Proportional gain (k) is the size of the correction that will be implemented. Figure 2 shows a standard proportional control logic diagram.

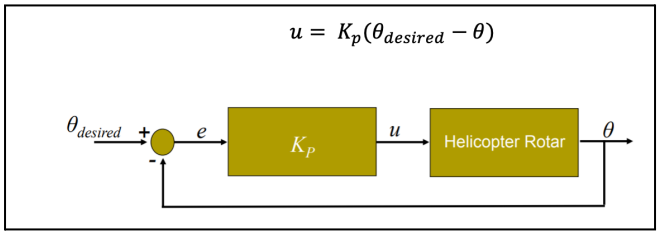


Figure 2. Proportional Control Diagram

There are many ways to apply P control to a system. In this experiment P Control will be used to adjust the robot's linear and angular velocity in order to control it to reach a desired point (goal point). As shown in figure 3.

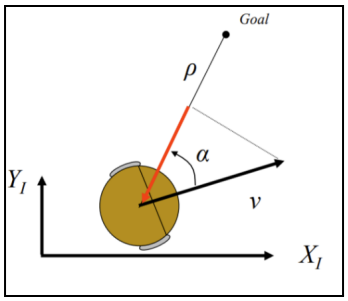


Figure 3. Robot and Goal Point

In order to adjust the angular and linear velocity an error needs to be measured over a period of time. Our error will be calculated from the current position and orientation of the robot and measured and compared to a goal position and orientation every 0.1 seconds. The kinematics between these two points are shown in figure 5.

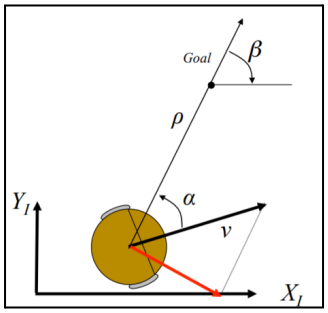


Figure 5. Robot and Goal Point Kinematics

The value rho (ρ) is the positional vector between the current and desired point in space. Alpha (α) is the angle between the robot direction of travel (Xr) and the rho position vector. Lastly beta (β) is the angle between the x-axis of the goal point and the rho position vector. These variables can be re-written as the following equations.

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \quad (2.1)$$

$$\alpha = -\theta + atan2(-\Delta y, -\Delta x) \quad (2.2)$$

$$\beta = -\theta - \alpha \quad (2.3)$$

Alpha, beta, and rho make up the error components of the proportional controller. A fixed proportional gain variable (k) is used for each error variable. Lastly the proportional controller equations to control the robot linear velocity (v) and angular velocity (w) can be written as as shown in equation 2.4 & 2.5.

$$v = k_\rho \rho \quad (2.4)$$

$$w = k_\alpha \alpha + k_\beta \beta \quad (2.5)$$

III. KINEMATIC MODEL

The robot used in our simulations is based on a simple 2 wheeled robot with differential driving, and so the standard differential driving kinematics was used. Differential driving meaning that each of the wheels are independently driven. This robot consists of two fixed standard wheels. Each wheel provides 2 DOF, allowing for linear translation and rotation around the point of contact.

The equations (3.1-3.3) below show how the linear and angular velocity of the robot are dependent on the individual wheel speeds.

$$\dot{x_R} = \frac{\dot{\varphi}_1 r}{2} + \frac{\dot{\varphi}_2 r}{2} \quad (3.1)$$

$$\dot{y_R} = 0 \quad (3.2)$$

$$\dot{\theta}(t) = \frac{\dot{\varphi}_1 r}{2L} - \frac{\dot{\varphi}_2 r}{2L} \quad (3.3)$$

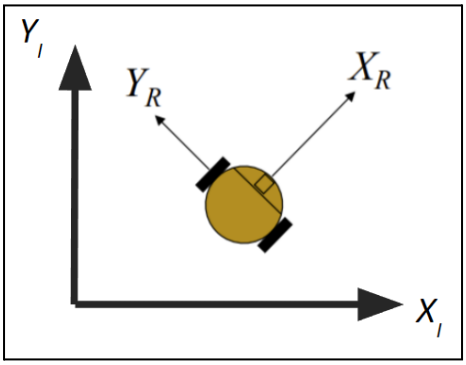


Figure 6. Coordinate System for Robot

In Figure 6, it can be seen how the linear velocity $\dot{X}_R$ is relative to itself in the robot frame XYR. In this case equation (3.2) shows that the velocity $\dot{Y}_R$ is zero since the robot does not move in that direction when driven by the motors, relative to the robot frame. The global frame is shown as $X_I Y_I$.

In order to calculate the kinematics required to design the proportional controller, it will require us to write our mathematics in the robot frame. To do this we can do a homogeneous transformation from the global frame to the robot frame as shown below.

$$\begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta}_I \end{bmatrix} \quad (3.4)$$

With the linear kinematics known we then derive the kinematic equations in relation to the polar coordinate system as shown in figure 5. As mentioned before, the kinematics involving α (alpha), ρ (rho), and β (beta) are used to evaluate the difference between the current position and the goal position. For example the linear speed rho dot can be derived from Figure 5 as shown in equation (3.5).

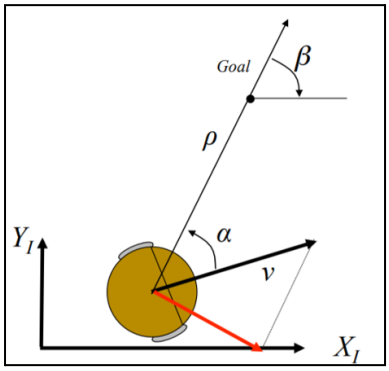


Figure 5. Polar Coordinate System

$$\dot{\rho} = -v\cos\alpha = -k_\rho \rho \cos\alpha \quad (3.5)$$

IV. CONTROL DESIGN

As mentioned the type of control implemented in this project is a Proportional control system. To implement our P controller onto our virtual robot we developed custom python code to take the current robot kinematics and derive the error between the current robot position and desired robot position and apply the proportional gain value to correct the robot motion every 0.1 seconds till the robot reached the desired waypoints.

Our code is separated into 6 steps, each done in a specific order to derive the system error and apply the proportional gain. Step 1 is to store the current kinematic data from both the desired goal position and the current robot position. As shown in figure 7. The python script runs a series of functions and stores the new kinematic data in a series of variables.

```
# REV 4.0 (w/ homogenious transform)
#------------Main PController Code----------------------------------------
#STEP 1:  begin by pulling current  position, velocity data
    # All d_ means destination
(d_posX, d_posY, d_theta) = self.robot.state_des.get_des_state()  # get next
    # All c_ means current_
(c_posX, c_posY, c_theta) = self.robot.state.get_pos_state()  # get current p
(c_vix, c_viy, c_wi) = self.robot.state.get_global_vel_state() #get current v
(c_v, c_w) = self.robot.state.get_local_vel_state() #get current local veloci
```

Figure 7. Python Current kinematic data

Step 2 takes the positional data from the robot in the global coordinate system and transforms them to positional data relative to the goal frame. Position transformation is used to allow for easier calculation of rho, alpha, and beta. Using homogeneous transformation equation 4.1- 4.3 we are able to obtain the robot's position and angle relative to the "goal" frame and vice versa. The python script used to perform the transformation is shown in figure 8.

$$H = \begin{bmatrix} c\theta & -s\theta & 0 & p_x \\ s\theta & c\theta & 0 & p_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

$$p'^{(I)} = Hp'^{(R)} \quad (4.2)$$

$$p'^{(R)} = H^{-1}p'^{(I)} \quad (4.3)$$

```
#STEP 2:  Calculate robot pos in terms of Goal reference position
#GLOBAL TO ROBOT TRANSFORM
RobotH =np.array([[math.cos(c_theta),-math.sin(c_theta),0,c_posX],
        [math.sin(c_theta),math.cos(c_theta),0,c_posY],
        [0,0,1,0],
        [0,0,0,1]])  #  Translation with x and y Rotate z-axis (delta) TxyRz
#GLOBAL TO GO TRANSFORM
GoH =np.array([[math.cos(d_theta),-math.sin(d_theta),0,d_posX],
        [math.sin(d_theta),math.cos(d_theta),0,d_posY],
        [0,0,1,0],
        [0,0,0,1]]  )  #  Translation with x and y Rotate z-axis (delta) TxyRz
#Globe frame TO GO frame  TRANSFORM
invGoH = np.linalg.inv(GoH)
P_globe_robot= np.array([[c_posX],[c_posY],[0],[1]])        # Robot location wi
P_go_robot= invGoH.dot(P_globe_robot)                       # Robot location wi
dx_go_robot= P_go_robot[0][0]                          #  First element of first
dY_go_robot= P_go_robot[1][0]                          #  First element of second
#Globe frame TO Robot frame  TRANSFORM
invRobotH = np.linalg.inv(RobotH)
P_globe_go= np.array([[d_posX],[d_posY],[0],[1]])       # Robot location with r
P_robot_go= invRobotH.dot(P_globe_go)                # Robot location with respe
```

Figure 8. Python Position Transformation

Step 3 of the python program derives the alpha, beta, and rho values from the known positions points. The equation for rho (ρ), Alpha (α), and beta (β) was highlighted in the equations 2.1, 2.2, and 2.3 respectfully. Figure 9 shows the custom python script for step 3.

```
#STEP 3:  Return the square root of x.                              ent instance
rho= math.sqrt((dx_robot_go)**2+(dY_robot_go)**2)       # distance between robot's current location and the desired location----- in robot frame
alpha= math.atan2((dY_robot_go),(dx_robot_go))          # angle between robot's XR axis and the rho vector----- in robot frame
beta= -(d_theta-c_theta)- alpha         # angle between rho vector and the Xg axis----- in robot frame
```

Figure 9. Python Error Calculation

Step 4 is the regulation of alpha and beta. Both of these units are in the polar coordinate system. And to be used by the proportional gain equation we need to have the values within the + pi and - pi range. A simple function is used to regulate the variables with our desired range, as shown in figure 10.

```
#STEP 4:  Regulate the rho, alpha, and beta
if alpha > math.pi:
    alpha =  math.pi
if alpha < - math.pi:
    alpha =  - math.pi
if beta > math.pi:
    beta =  math.pi
if beta < - math.pi:
    beta =  - math.pi
```

Figure 10. Python Error Regulation

Step 5 applying the calculated error values , (rho, alph, and beta) to the two proportional controllers. The linear velocity proportional controller as shown in equation 2.4 and the angular velocity proportional controller as shown in equation 2.5. The python script for step 5 is shown in figure 11.

```
#STEP 5: # Propotional Controller for local linear and angular velocity
c_v = self.kp*rho
c_w = self.ka*alpha-self.kb*beta
```

Figure 11. Python Proportional Controller

The final step is used to identify when the robot has successfully reached the desired goal point. As mentioned before each desired goal point specifies a position (x,y) and an orientation (theta) relative to the global frame. During each loop the difference between position and orientation of the robot is calculated. The robot is considered at the desired point when the difference between the goal point's and the robot's position and orientation is zero (or close to zero).

Once the robot has reached the desired location the next waypoint location data is loaded and the robot proceeds to the new goal point, running the same proportional control algorithm. Only when all goal points have been reached does the robot stop. The python script shown in figure 12 identifies when the robot reaches the desired goal point.

```
#STEP 6: # reach way point criteria
#you need to modify the reach way point criteria
if  abs(d_theta-c_theta) < 2 and abs(rho) < 1 : #you need to modify the reach way point criteria
    self.robot_destination_reached = True
else:
    self.robot_destination_reached = False
```

Figure 12. Python Point Reach Criteria

V. P CONTROL IMPROVEMENTS

In this experiment multiple waypoints were selected and the robot pathway was evaluated. Upon review it was clear that the robot trajectory had a few issues that would make the pathway less than ideal.

The primary issue we wanted to address was the robots inefficient turning when moving toways a waypoint who's orientation axis is facing the opposite direct from the robot's direction of travel. When the error between goal orientation and robot orientation "alpha" is less than +/-pi/2; alpha will converge to 0 as it approaches the goal point as shown in figure 13. As alpha converges to 0 the robot performs predictable turing adjustment.

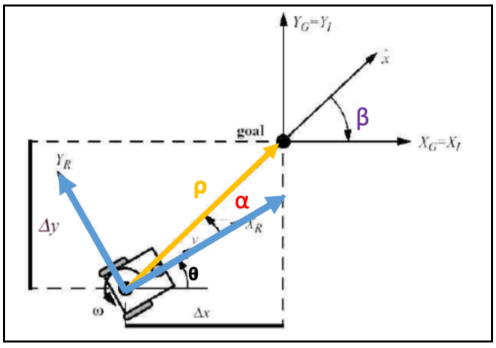


Figure 13: Angles In Goal Frame

However, when a goal point orientation is opposite to the required path orientation of the robot irregular turing occurs. In this scenario the equation used to calculate alpha approaches pi as the robot moves closer to the goal point. It's not uncommon for pi to jump between -pi and +pi in this period of time. When pi jumps from positive and negative values it causes the robot to perform twists, drastically affecting the robot trajectory. This scenario is shown in figure 14.

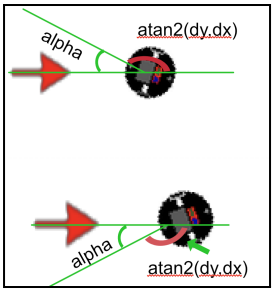


Figure 14: Alpha and Beta Irregularity

In order to fix the twisting issue we simply change the frame of reference from the goal position to the robot frame. In this different point of view the alpha angle derived by atan2(dy,dx) will always approach zero as long as the robot is moving towards the goal point. This change of reference frame is shown in figure 15.

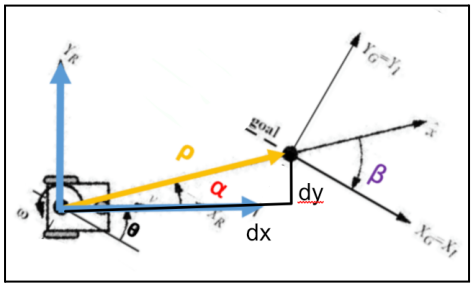


Figure 15: Angles in the Robot frame

The change of reference frame also simplified the equation of alpha. With the slightly modified equation we updated the python p controller code as shown in figure16.

```
rho= math.sqrt((dx_robot_go)**2+(dY_robot_go)**2)
alpha= math.atan2((dY_robot_go),(dx_robot_go))
beta= -(d_theta-c_theta)- alpha          # angle be
```

Figure 16: Updated P Controller for Robot frame

Upon completion of the new p controller referencing the robot frame we experience better robot performance and smoother pathways trajectories, especially when the goal points orientation was opposite to the necessary robot direction of travel.

VI. METHOD

After the completed implementation of the P control algorithm in python we then conducted a series of testing to evaluate the effect changing the proportional gain values Kp, Ka, and Kb had on the system. For each test we had the robot move to the exact same goal point positions. As shown in figure 17 the robot begins to travel to the right of the map and then proceeds clockwise around the entire map.

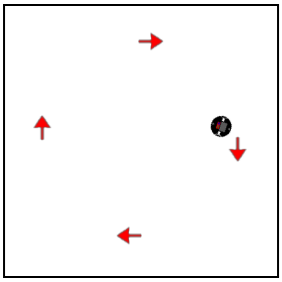


Figure 17. Waypoint Map

In order to determine the effect of the proportional k values we began with a control group and during each trial we modified one k value out of the set of 3. We then describe the effects of the robot motion trajectory when each value was changed.

kp = 3 # k_rho
ka = 8 # k_alpha
kb = -1.5 # k_beta

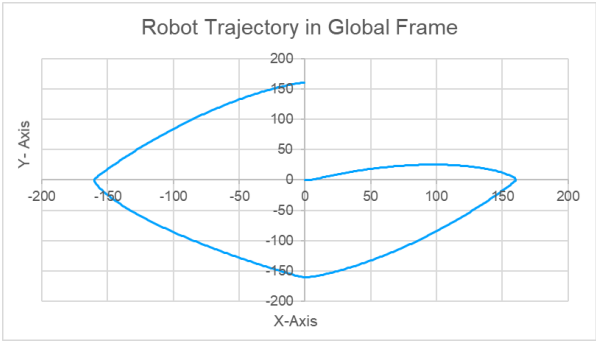
VI. EXPERIMENT RESULTS AND DISCUSSION

As mentioned in order to determine the effect of the proportional k values we began with a control group. When choosing the k values we ensured they meet the general k value criteria shown in equation 5.1. Our control k values were:

kp = 3 # k_rho
ka = 8 # k_alpha
kb = -1.5 # k_beta

$$k_\rho > 0, k_\beta < 0, k_\alpha + \frac{5}{3}k_\beta - \frac{2}{\pi}k_\rho > 0 \qquad (5.1)$$

The robot completed the control mission with no visual anomalies or impossible frames. As seen in figure p18 the robot trajectory also shows a clean and strain path between all the waypoints.



left wheel speed 3.62 impossible frame 0 right wheel speed -2.12

Figure 18. Control Trajectory

Now with the control completed we conducted another test with a slightly different Kp value. The values used in this test are listed below. The robot trajectory in this test is shown in figure 19. Reducing Kp would reduce the overspeed of the robot; it also seemed to cause the over compensation of alpha and beta after point 2. This trajectory also had 3 impossible frames.

kp = 1 # k_rho
ka = 8 # k_alpha
kb = -1.5 # k_beta

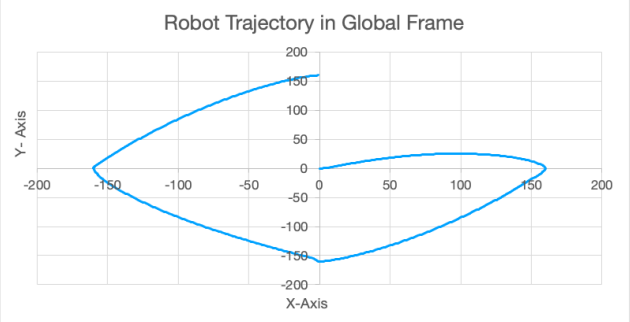


Figure 19. Kp Modified Trajectory

Next we conducted a test with a slightly different Ka value. The values used in this test are listed below. The robot trajectory in this test is shown in figure 20. Reducing alpha widens the arch of the trajectory of the robot between waypoints. This path also had no impossible frames.

kp = 3 # k_rho
ka = 6 # k_alpha
kb = -1.5 # k_beta

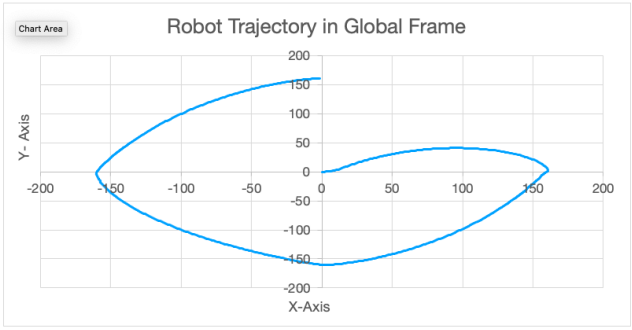


Figure 20. Ka Modified Trajectory

Our last test we slightly changed the Kb value. The values used in this test are listed below. The robot trajectory in this test is shown in figure 21. Reducing beta caused a significant alteration to the robot trajectory. Expecily between the 3rd and 4th waypoint. This new k value also caused 7 impossible frames to be formed.

kp = 3 # k_rho
ka = 8 # k_alpha
kb = -3 # k_beta

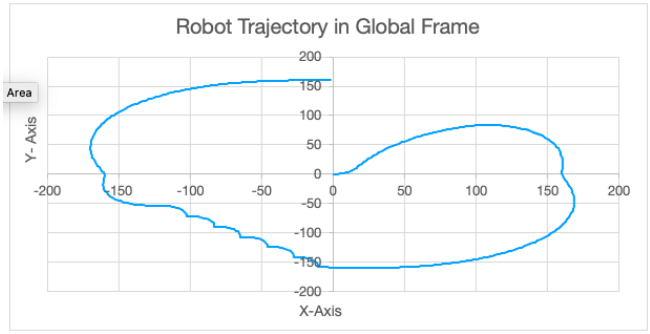


Figure 21. Kb Modified Trajectory

VII. CONCLUSION

In modern robotics control theory is a critical area of focus, it is only with the use of these control algorithms that robots can function in real world environments. The goal of this lab is to explore proportional control to control a robot's route in a virtual environment. Our robot independently navigates to 4 different waypoints using a custom proportional controller algorithm we developed in python. This proportional controller uses the linear and angular velocity of the robot until it reaches the desired waypoint.

After completing a series of tests using different K values we were able to achieve a trajectory that worked on a wide variety of motion pathways without any impossible frame errors.

In our experimentation we determine the effects different k values for alpha, beta, and rho had on the system. Reducing Kp would reduce the overspeed of the robot but it could cause imbalance between linear and rotational velocity causing oscillations. Reducing Ka affects how direct the robot will take to the next goal frame. A lower value allows the robot to take a larger arch. Lastly Kb would also cause the path to widen its arch but it could cause oscillations as the angular velocity becomes over compensated.

Overall we are satisfied with the effectiveness and the level of versatility shown by our custom Pcontrol algorithm.

VIII. ACKNOWLEDGEMENTS

None

IX. REFERENCES

Jenkins, Jonathan, and Erik Duque. “ME 5751 Module 1: Proportional Control for Point Tracking.” *ME5751_Module1*, GitHub, 2022, https://github.com/jonjenkins31/ME5751_Module1 .

Siegwart, Roland, et al. *Introduction to Autonomous Mobile Robots*. Second Edition ed., Massachusetts Institute of Technology, 2011.