ME5751 Lab 1 – Proportional Control for Point Tracking

Ryan Chao
Cal Poly Pomona
Pomona, CA
rchao@cpp.edu

Cameron R. Weigel
Cal Poly Pomona
Pomona, CA
crweigel@cpp.edu

***Abstract*— In this lab, point tracking was implemented on a simulated robot through using kinematic models and a proportional controller to drive the robot to a desired state. Several issues the robot experienced during trials were indefinite oscillations and swinging which were dependent on the gains set. With our tolerances, $|\rho| < 0.05$ and $|\alpha| < 0.025$, the best performance when occurred when $k_P = 3.5$, $k_\alpha = 10.0$, $k_\beta = -1.0$. Compared to our default case, this set of gains performed roughly 10% better in terms of reaching the final point.**

Keywords*— *odometry; point tracking; kinematic modeling; proportional control; Vieta's formula

I. INTRODUCTION

Motion control of a differential drive robot can be described as, given a global frame for reference, and a goal frame, navigate the robot from its current position to the goal point in the correct orientation. There are many methods that can be utilized to move the robot efficiently to its goal. The method we have developed in this paper can be described as a proportional controller based on the kinematic model of a differential drive robot. This paper discusses the theory of the kinematic model and p-control followed by our method for implementation. Utilizing a simulated environment with odometric data, we achieved a robust P-controller algorithm that can accurately locate the robot in both position and orientation in a 2D plane. By varying the gain constants used within the P-controller, the robot arrives at its goal frame with varying degrees of overshoot and travel time. Additionally, by implementing a simulated limit to the robot's velocities, an algorithm was formulated for adaptively maintaining the robot's velocities within the set bounds.

II. BACKGROUND

Since the robot was developed in a simulated environment, many physical factors such as slip or friction can be ignored which allowed for simplification of the kinematic models for motion planning. The robot is a 2D representation of a differential drive robot with the characteristics of having two drive wheels, and a body of fixed length. It has 2 DOF, forward/backward velocity and angular velocity. Maximum allowable velocities are determined by parameters that limit the left and right wheel velocity in radians per second. Odometry data is calculated based on the current linear and angular velocity in the global frame. Due to the precision of the odometric data utilized, error in position is significantly reduced in complexity and is considered only as a means of preventing the robot from asymptotically approaching its goal. All code was developed in Python 3.x and tested on both Windows 10 and Ubuntu 20.04. *Fig. 1* details the testing environment for the robot.

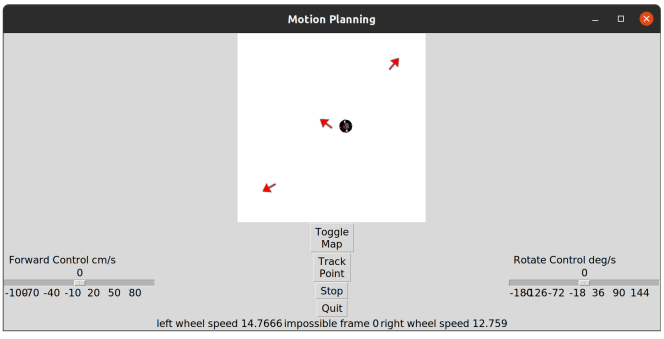

Fig 1. An example of the test environment with three goal positions.

III. KINEMATICS

In order to establish the kinematic models required to control a robust, stable robot, we reference Fig. 2 found in Introduction to Autonomous Mobile Robots 2nd Edition by Roland Siegwart. Using this setup, the velocities can be derived from the position and orientation of both the robot and goal relative to the global coordinate frame.

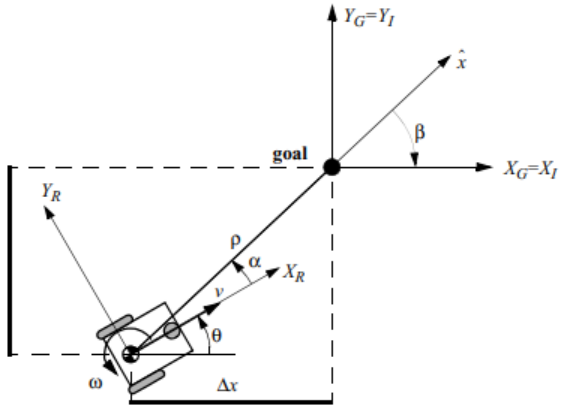

Fig. 2 Robot kinematics and frames of interest.

Our kinematic equations for controlling a differential drive robot with wheel radius *r*, and body radius *L*, can be described as:

$$\dot{x} = \frac{\dot{\varphi_L} r}{2} + \frac{\dot{\varphi_R} r}{2}$$

$$\dot{\theta}(t) = \frac{\dot{\varphi_L} r}{2L} - \frac{\dot{\varphi_R} r}{2L}$$

Letting θ_c and θ_d be defined as the current angle of the robot frame relative to the global frame and the angle of the goal frame relative to the global frame respectively, The pose error given in polar coordinates can can be described with:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = -\theta_c + atan2(\Delta y, \Delta x)$$

$$\beta = -\theta - \alpha$$

Where, $\theta = \theta_c - \theta_d$ *and*

$$\{\alpha, \beta\} \in [-\pi, \pi]$$

Given odometric readings of the robot's current *x*, *y* and angle θ_c in the global frame we can utilize the equations prior to create a proportional controller for the robot's linear and angular velocities. Using α, β, and ϱ with scalars k_α, k_β, and k_ρ, we can determine our current linear and angular velocities (c_v, c_ω) by the following two equations:

$$c_v = k_\rho \rho$$

$$c_\omega = k_\alpha \alpha + k_\beta \beta$$

With constants k_α, k_β *and* k_ρ satisfying the following stability conditions:

$$k_\rho > 0,\ k_\beta < 0,\ k_\alpha - k_\rho > 0$$

These k-values are normally constrained by the physical characteristics of the system such as the motors' max RPM; in our case, we limit our robot's wheel motion to less than or equal to 16 rad/sec.

IV. METHODS

In order to satisfy the imposed limitation on wheel velocity, we must preemptively calculate wheel velocity and adjust it before sending it to our velocity control function. To accomplish this effectively, we introduce a function that iteratively reduces k-values by 10% if the robot's initial velocity is outside its limit. The function then checks the new wheel velocities φ_L and φ_R on each iteration and once the wheel velocities are within the specified limit, the function outputs linear and

angular velocities (c_v, c_ω), and wheel velocities (φ_L, φ_R).

```
def check_wheel_speed(self, c_v, c_w):

    phi_l_tmp = (c_v + self.bodyL*c_w)/self.wheelr
    phi_r_tmp = (c_v - self.bodyL*c_w)/self.wheelr

    ka = self.ka
    kb = self.kb
    kp = self.kp

    while ((phi_l_tmp < -16.0 or phi_r_tmp < -16.0) or (phi_l_tmp > 16.0 or phi_r_tmp > 16.0)):
        ka = ka/1.1
        kb = kb/1.1
        kp = kp/1.1
        #print(ka, kb, kp)

        c_v = kp*self.rho
        c_w = ka*self.alpha + kb*self.beta

        phi_l_tmp = (c_v + self.bodyL*c_w)/self.wheelr
        phi_r_tmp = (c_v - self.bodyL*c_w)/self.wheelr

    return ka, kb, kp, phi_l_tmp, phi_r_tmp
```

Fig 3. Function to calculate and adjust wheel velocities to be within the allowable range.

This function allows for a rough maximization of wheel velocity based on initial k-values and the max wheel velocity limitation. In a future design, this function should be rewritten to individually determine the necessary changes to each k-value instead of applying a 10% reduction across all three k-values.

Additionally, we implemented a variation of the P-controller for situations when the robot's goal is behind the robot to avoid unnecessary traveling between points.

$$If \ -\frac{\pi}{2} > \alpha > \frac{\pi}{2},$$

$$\alpha = -\theta_c + atan2(-\Delta y, -\Delta x)$$

$$\rho = -\rho$$

In this situation, it is more efficient for the robot to drive backwards to reach its goal and at the end, turn around to face the goal orientation if it faces the opposite of the robot's current direction.

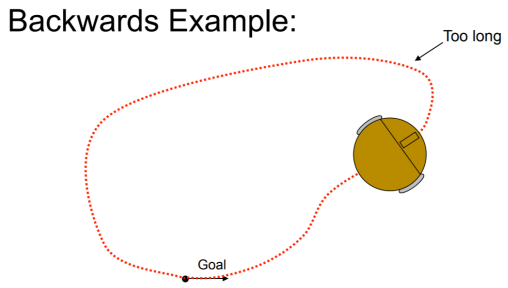

Fig 4. In this path example, it is faster to reach the goal by driving in the reverse direction

V. RESULTS AND DISCUSSION

The results of the three experiments using different k-values are presented in this section. Goal points are held constant and the paths are logged to compare changes in the path computed and time taken to reach each goal.

A. Path Comparison (Virtual)

Fig. 5 shows the path taken by the robot to reach each goal point based on three different gain settings and does not take into consideration elapsed time per trial. The red path will be referred to as setting 1, blue path as setting 2, and green path as setting 3 where the values are in the following order: k_P, k_α, and k_β. The fastest and most optimal motion taken by the robot occurs with setting 1, in which it reaches each goal point without having to stop and orientate itself. Setting 2 represents a set of gain values randomly selected from an algorithm that solves for all possible gains that satisfy Vieta's formula for robust stability control via the following conditions.

$$k_P > 0,\ k_\beta < 0, \text{ and } k_\alpha + \frac{5}{3}k_\beta - \frac{2}{\pi}k_P > 0$$

However, when testing settings 2, it did not travel efficiently and overshot its turn when reaching each goal point. Based on data logs, it can be observed that the robot would overshoot by no more than $\pm\frac{\pi}{12}$ to each goal point before turning to correct itself. Setting 3 yielded the most, pragmatic

path by reversing backwards before driving forward to the next goal point without needing to orientate itself. This was an unexpected result caused by the robot reversing at a large radius which placed the goal point in front of the robot thereby satisfying the condition to move forward.

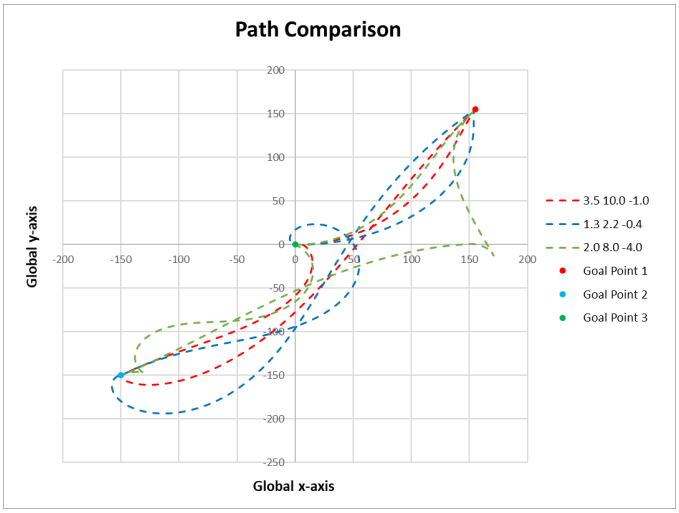

Fig. 5. Graph of robot path based on a variety of gain values.

B. Time Comparison (Virtual)

Fig. 6 shows the time taken for the robot to reach each goal point based on three varying gain settings. Please note that the times in the figure are in multiples of 0.1 since every iteration occurred after 0.1 seconds. When comparing each setting to the average, setting 1 reaches its destination at least 17.37% sooner at every goal point whereas setting 2 is late by at least 21.18% demonstrating the importance of optimizing the stability control in a motion robot system. This was to be expected as the speed gain is inversely proportional to elapsed time – the slower the robot is, the longer it takes to reach the end of its path. However, other factors such as path curvature and orientation contribute to the time needed for completion. These are directly affected by the angular stability gain constants, k_α and k_β, which are utilized, linearly for the robot's angular velocity.

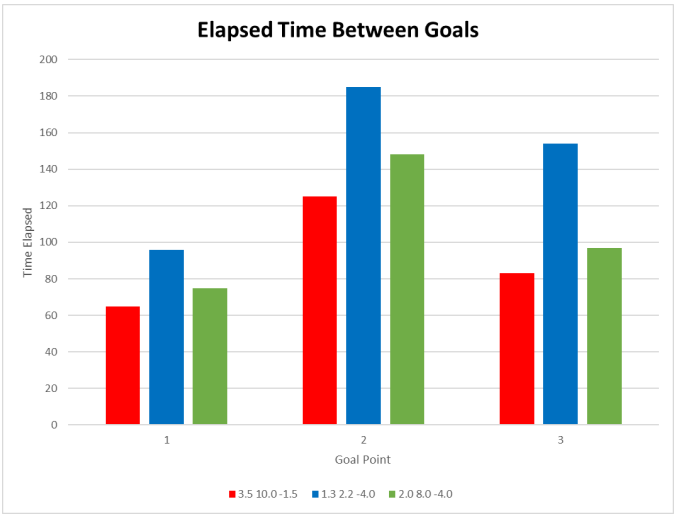

Fig. 6. Graph of elapsed time between each goal point.

C. Observations

From Fig. 5 and Fig. 6, we can characterize each gain's effects on the robot's control. Since k_P is directly proportional to the linear speed, increasing or decreasing its value will do the same for robot's velocity as observed in the path and time visual representations. Comparing the path of setting 1 to settings 2 and 3, it can be determined that k_α is directly proportional to the angular velocity of the robot's motion. This is shown as setting 1's k_α of 10.0 results in a significantly straighter path or larger curvature than that of settings 2 and 3. Lastly, k_β is observed to have an opposite and weaker relation to path curvatures compared to k_α as shown in the path of settings 2 and 3 when traveling from goal point 1 to goal point 2. The path of setting 3 starts off by reversing in the CCW direction as opposed to the CW direction the other two settings travel in. Furthermore, the magnitude of the turn requires k_β to be significantly, 10 times that of k_α, to affect the robot's orientation which demonstrates its weak relation to angular velocity.

The summary of the relationships between gain and velocities are shown in Table 1. These values are only through observation of the different tests run in this lab.

Table 1. GAIN AND VELOCITY RELATIONSHIP

	Lin. Velocity	**Ang. Velocity**
k_P (↗ / ↘)	↗ / ↘	—
k_α (↗ / ↘)	—	↗ / ↘
k_β (↗ / ↘)	—	↙ / ↖

VI. CONCLUSION

Varying the gain values k_α, k_β *and* k_ρ directly influenced the trajectory of the simulated differential drive robot's point tracking algorithm. This is evident in the various tests performed and of the values tested, the stability gains which yielded the most optimal motion for the robot to reach each of the goal points was discovered to be: $k_\rho = 3.5$, $k_\alpha = 10$, $k_\beta = -1.5$ with the wheels having a velocity constraint of 16 rad/s.

Since minimal error was incurred due to a testing in a simulated environment, linear and angular error in the final goal pose was determined based on the trade-off between error and time elapsed. Overshoot in angular position was present due to the gain values and sampling frequency but from our findings, overshoot was bounded by $\pm \frac{\pi}{12}$ radians before correction. Under optimal conditions, the overshoot was undetectable by the human eye and had an estimated bound of $\pm \frac{\pi}{240}$ as this is the angular tolerance set in the P-controller.

Overall, the data discovered in this lab allowed for a visual understanding of the complexities behind controlling an independent, mobile robot. In future iterations of a proportional control algorithm, additional gain values should be tested and a deeper understanding of how to proportionally reduce or increase those gain values due to wheel velocity constraints would lead to an overall better performance. Our implementation reduced gain values by a constant proportion in order to achieve velocities within the constraints, but there may be a more sophisticated method that would aid in achieving faster and/or more efficient point tracking.

VII. REFERENCES

[1] R. Siegwart, I. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., The MIT Press : Cambridge, 2011.

[2] Y Chang, California Polytechnic University, ME5751 Lecture Slides, Fall 2022