Dean Fletcher

Report – Lab 3

Video: https://youtu.be/PG-UzVzCtXw

Task 1:

Given – Distance (3 – 7 inches), V
Given – Kp (0.1, 0.5, 1.0, 2.0, 2.5, 5.0)
Choose – Wall distance, Kp

For this task we use the left/ right distant sensors to remain at a desired wall following distance. Ideally with the use of both sensor we can keep the robot moving straight while staying in the middle of a 10" tile. The algorithm chooses the closest wall. By doing so it remains between walls and close to the middle. For my build the values below that I found worked best and most accurate:

Wall follow distance = 3	Gain (Kp) = .1

My reasoning for these values is that the tile is 10" across. A wall follow distance of 3" means that there shouldn't be less then 3" per side. This makes up 6" of the 10" with 4" remaining. This is reserved for the diameter of the robot and some room for error. The Kp value is used to scale how much the wheel velocities are affected by the error being accumulated. I found that a higher Kp value resulted in a less accurate outcome. This makes sense when looking to the equations to derive the velocities.

PID:

$$error = desired - measured$$
$$control = Kp * error$$
$$saturation = f_{sat}(control)$$

$$f_{sat}(control)\begin{cases} if\ control > max & control = max \\ if\ min \leq control \leq max & control = control \\ if\ contorl < min & control = min \end{cases}$$

Implementation:

$$\text{max velocity: } -6.28 \leq max \leq 6.28$$
$$Error = sensor_{value} - Goal_{distance}$$
$$Wheel\ PHI = Error * Kp$$

This is applied to the wheel you want to effect based on a positive (+) or negative (-) error. Resulting in turning toward or away from the wall. In the case of stopping the calculated velocity is applied to both wheels to slow the robot. In the circumstance were the robot overshoots the goal, the error will become negative causing the robot to move in reverse.

Task 2:

For this task we are to follow a wall and find our way through the maze. The assumption is that the starting position can currently see the wall it is trying to follow. The algorithm proceeds as such:

1. Wall Infront (obstacle):
    a. At a corner rotate $90°$ to open direction # following wall
    b. At a dead-end rotate $180°$
    c. Follow_Wall()
2. Past Wall:
    a. Turn radius to wrap around wall
    b. Follow_Wall()
3. Follow Wall:
    a. Drive straight until:
        i. Wall_Infront() OR Past_Wall()
        ii. Reach goal

The logic used for this is the same as used in task 1. The added complexity was to deal with error building up after every turn. Each turn adds another level of uncertainty for the robot's position. The wall follow algorithm presented in task 1 is slow to correct the over/under turning that occurs. This is due to the use of a lower Kp, which scales velocity less aggressively to the error. I was able to overcome this by making precise turns and rotations. The kinematics for these are below:

Rotation:

Wheel Velocities: $\max = (\pm)6.28$
Wheel Velocities: $(\max, -\max)$ or $(-max, \max)$
$\text{Time} = \frac{\text{Distance}}{\text{Velocity}}$
If IMU degrees equal target degrees or Time reached:
- Return

Curve:

max velocity: $-6.28 \leq max \leq 6.28$
$Theta\ Change = 90°$
$omega = \frac{v}{R}$
$Wheel\ Velocity = omega * \left(R \pm \frac{axle}{2}\right)$ # Keeping in mind curve direction
If Wheel Velocity > MAX:
    PHI_2 = MAX * (PHI_2/|PHI_1|) # scale back proportionally
    PHI_1 = MAX * (PHI_1/PHI_1) # set to max preserving sign
If current degrees == goal degrees:
- Return

Velocity:

max velocity: $-6.28 \leq max \leq 6.28$

Phi = velocity / wheel radius

Max Phi = 6.28

Conclusion:

I am continuing to learn just how much error accumulates over time in robotics. The more maneuvers the robot performs the more uncertainty there is. While testing different gains and following distances I found a trend where lower values tend to drive smoother. As the the values increase the robot's movements become jerkier and erratic. This presents issues when trying to traverse the maze. As the robot begins a correction the sensors can lose the wall. This in turn makes the robot think it has pasted the wall being follow and needs to perform a corrective action, such as a drive some curve. At which point it is very difficult for the robot to recover and often results in hitting a wall. Implementing PID into my controller has given the robot the ability to continuously make small corrections. In conclusion, this project required a lot of little adjustments to design an algorithm where the robot drove as expected.