Dean Fletcher

Report – Lab 4

Video: https://youtu.be/dk1rRNzb04A

Task 1: Motion to Goal

For this task there is 2 possibilities, either it can see the goal and performs motion to goal (MG), or it can't and performs rotation until goal is located again.

The controller does the following:

1. Rotate until yellow goal found
2. Stop once goal is center of camera
3. Drive to goal performing a MG() process
4. While performing MG() stop if:
    a. Within stopping distance of goal
    b. Object no longer in field of view

PID:

Kp (0.1, 0.5, 1.0, 2.0, 2.5, 5.0)

$error = desired - measured$

$control = Kp * error$

$saturation = f_{sat}(control)$

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

Implementation:

$Error = sensor_{value} - Goal_{distance}$

$Wheel\ PHI = Error * Kp$

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

If max velocity is reached, I have a speed correction function that will drop the excess speed to the max allowed. I added an extra step that in the event where the excess velocity is reached during a curve it not only sets that wheel to the max but drops the other wheels velocity the same proportion. This is to ensure that the desired radius is not affected by the change in velocity.

Task 2: Bug Zero Algorithm

This task expands on task one by using motion to goal. Once an object prevents the robot from performing MG() it must avoid it by using the Bug Zero algorithm.

The algorithm proceeds as such:

1. Locate Goal
2. MG() unless:
    a. Hit wall
3. Turn Left or Right (unchanging)
4. Follow wall until:
    a. Can see the goal again
5. Perform MG()
    a. Repeat until goal reached

The motion to goal algorithm is the same as it was in task one. The wall follow algorithm is like it was performed in Lab 3. Note that the wall follow process uses the same PID idea as in motion to goal. The added difficulty was first to add in a leave point and second to deal with hitting walls at non-perpendicular angles. My algorithm has a leave point whenever it can see the goal and can drive in that direction again. This was done during a curve since at that point it should have passed a wall and as it turns should scan for the goal. If it sees the goal, then break from the curve and start heading to the goal.

Curve:

Omega = Velocity / Radius
Velocity = omega * (Radius + Axle/2)
Wheel Velocity = velocity / wheel radius

Conclusion:

This lab surprised me in several ways. My controller was first designed with right turns only. Once that was working, I had to really tweak my controller to effectively drive using left turns as well. After that I had to add a leave point once motion to goal was possible. This presented a problem since when the robot leaves the angle at which it is driving makes, it difficult to sense an approaching wall. Once the wall is sensed it needs to break and perform wall follow again. This was a touch and go process since adding this logic presented numerous bugs in existing code. In result, I found a lot of little issues that by fixing them really improved the robustness of the controller.