**Name: Chen Hao Cheng**

**Lab 2.2**

The key equations for the controller are as follows:

\*\*gX, gY, gT are the x, y, theta coords of the goal point.

\*\*rX, rY, rT are the x, y, theta coords of the robot's position

dX = (gX-rX);

dY = (gY-rY);

rho = sqrt(pow(dX, 2) + pow(dY, 2));

Alpha = pi/2 - atan2(dY,dX);

dT = alpha - rT

nu = gT - rT

x\_rp = **A**\*rho

theta\_rp = **B**\*dT + **C**\*nu

A, b, and c are constants that we tweaked to make our robot perform better. A is proportional to the distance away from the point our robot is. B is proportional to the difference between our robot’s heading and the heading of the point we are trying to get to. C is proportional to the desired orientation of the robot.

X\_rp and theta\_rp are used directly to calculate the desired wheel speeds of the robot. Changing A, B, and C change the way the robot performs. For instance, if you increase B, the robot will try to turn faster. If you decrease B, it will not care as much about turning. A is similar, but has to do with the speed the robot travels, instead of the speed at which it turns.

We set C to almost 0, which is 0.01 because it wasn’t as important of an aspect to get to the desired point as A and B were. As long as we got to the point, then the ‘nu’ term takes over, and turn the Sparki to the desired orientation.

If the constants get too big, the robot will calculate the desired wheel speeds to be too big. For instance (180%, 100%) left and right motor powers should hypothetically make the robot turn while still moving forward, but in the physical world, the motors can’t go past (100%, 100%). This results in the robot just driving forward instead of homing to the point. This problem can be fixed by scaling the motor speeds down to both be between -100 and 100%.

Assuming the robot is able to turn away from obstacles, if a u-shaped obstacle is placed between the robot and the goal point, the robot might get caught inside of it. It tries to drive towards the point, sees the obstacle, and turns away. The code would then make it turn back and try to drive towards the point again. This might result in a robot turning back and forth without moving forward at all. This problem could be fixed by implementing a state machine, switching between “seek point” and “avoid obstacle”.