**Team Members: Keenan Griffin, Justen Stall**
**Course: 499/592**
**Assignment: Lab05**

## Task 1: Getting To The Wall

Objective:

On button press, drive straight at the wall and stop when the wall is found with the light bump sensor. On button press, perform a 90 degree turn to the right placing the left side of the robot next to the wall

Design:

Our wall finding code is reimplemented using the code from Lab 04 task to stop when a wall is detected with any light bump sensors. The robot is placed intentionally facing the wall to ensure the wall is found quickly. The 90 degree turn function is called “rotate_until” and takes a degree input and performs the turn by distance. The distance is the circumference calculated with the wheel diameter of the iCreate Robot times pi (wheel diameter * pi). When aligning the robot to find the wall, the robot is always placed perpendicular to the wall so the turn will always place the robot parallel to the wall after the rotate function is complete.

Description:

Task 1 design and implementation was pretty straightforward given wall finding was done in the Lab 4. There were no challenges finding the wall, and after noting the wheel diameter in the iCreate Robot manual calculating the circumference to rotate did not have any challenges. We even had distance driving figured out from Lab 4, so we could use the same logic to ensure the robot would turn the correct number of degrees.However,  there is one snag with the robots turning. Giving an exact degree value of “90” would consistently result in the robot stopping just shy of an actual 90 degree turn. To compensate for this the input for degrees to turn was a little higher than 90; sometimes rotate was given 100 degrees to turn just to overcorrect and give the robot more room when wall following.

Another design we had in mind for handling the turn was to use the light bump sensors, and set a threshold to stop the robot when the light bump sensor value reached that threshold. This approach to determining when to stop the robot’s turn would also allow us to align the robot in different angles to the wall other than relying on a perpendicular alignment. Given more time we would implement this version of turning.

## Task 2: Wall Following

Objective:

On button press begin wall following. Maintain a close following to the wall, adjusting accordingly if the robot is too close or too far from the wall. When facing a concave turn, perform a turn to the right. When facing a convex turn, perform a turn to the left.

Design:

For this task we went with a PID controller over a state machine to handle sensor outputs and adjusting wheel velocity to navigate along the wall. The sensor values from each left side light bump sensor are added up to one sensor value and stored in a circular array. These sensor readings are then used in the PID controller and the output is used to determine if the robot needs to deviate its wheel velocity if the output falls in a certain range. We used the magically appearing pseudo code on the white board in the robotics lab to help with designing the logic statements for ranges and deviation. When the output falls within a range considered too close to the wall, the robot will deviate to the right. Similarly when the output falls within a range too far from the wall the robot will deviate to the left.

Description:

Right off the bat we had challenges with this task. Finding the proper ranges to determine when to deviate was a massive trial and error effort. We were able to get the robot to follow the wall pretty consistently but had troubles when making turns. The common issue is the robot would start to make the deviations but it was already too close to the wall and in the corner too much. To compensate we attempted to adjust the robots deviation speed in hopes of quickly moving away from the wall. This created issues with steady wall following as the robot would quickly oscillate left and right when making corrections.

To help the robot navigate out of turns it got stuck in, we implemented a reverse function to back the robot up to the left so as to give it more space to complete the turn. Seeing as the robot wall follows on the left it only needs to correct right hand turns where it could potentially get stuck as the robot will have plenty of room to make left hand turns. Still the robot would not turn quickly enough even given space from reversing and often get stuck driving and reversing.

One potential solution for handling turns better is creating range checks with greater deviation values assigned to the ranges. For example if the PID output got large all of a sudden then the robot would be really close to a corner and need to make a quick turn out.

Reflection:

Our process of controlling the robot used a PID controller. Using the PID controller we were able to more accurately take into consideration the current readings of the light bump sensor and adjust accordingly. Our alternative option was to essentially map out all possible conditions the robot could find itself in, then program the robot to change direction accordingly. With the PID controller we were able to implement the robot's control to react to the feedback it received from the sensors allowing the robot to handle more of the navigation issues the robot would face. By using a PID controller we allowed the robot to handle navigation on its own instead of considering every situation it could find itself in, and hard code the robots response to those situations.