# CSCE 274

# Team# : 05

# Project# : 01

Josh Benton

Nick Wrobel

Chris McKinney

**Description:**

The main actions of our program take place inside an infinite event loop located in the main function of `main.c`. Inside this loop, we execute a periodic function, which does long-running tasks. These tasks include blinking the LEDs on the command module on and off and checking if the bump sensors are depressed. If the left bump sensor is down, the play LED on the Create is turned on, and if the right bump sensor is down, the advance LED on the Create is turned on. Additionally, we check if the front caster on the Create is up or not. As a safety precaution, if the wheels drop or the robot is picked up, the robot is turned off.

Also inside the infinite loop, we check the status of the play and advance buttons on the Create. If the play button is pressed, we drive the Create along a pentagon in a clockwise direction. If the advance button is pressed, then we drive the Create along a pentagon counterclockwise.

To deal with differences in robots, we have a tolerance added to the rotation and movement commands to help counter issues related to movement and rotation. We also found moving a bit slower (50 mm/s) improves accuracy.

A challenge we faced in this project was being able to check the status of the bump sensors and toggle the LEDs while still driving the robot. The robot could not execute both at the same time due to limitations of the existing drive timer function.

To deal with this, we wrote a modified timer function which executes commands based on the timer (the timer counts down, and when it hits a specified interval it executes a function that was passed with a function pointer). This fixed our issues related to the command module and robot, where commands sent from the robot, such as checking the bump sensors were only executed after movement code ran. As a result, we were able to make the robot check the bump sensor values and blink the LEDs while still driving along a pentagon.

Additionally, we upgraded the timer functions to use 32 bit integers. This allows for easier use when doing math, and now we can move a very long distance very slowly.

We also created libraries to encapsulate various routines. We wrote a library (`irobled.c`) that keeps track of led states and allows modification on an individual led level. This has helped with blinking and is expected to help with IO in the future. We have a library (`sensing.c`), consisting of a single function that will return the bytes from the robot, given the desired packed ID. We also have a library (`driving.c`) to make driving the robot more intuitive, allowing us to do things such as drive the robot a given distance, a given angle, and to stop the robot, without having to know about messy opcodes.

**Evaluation:**

Our program works fairly well. However, we do have some slight error in the path of our polygon. This is difficult because of the differences in robots, but could potentially be improved by tweaking the speed of the robot, or driving on a hard surface with less friction (tile, for example) rather than carpet.

**Allocation of effort:**

| | Paper | Primary Programmer | Debugging | Scripting / Ease of Use | Solution Generation | Planning | Data Collection |
|---|---|---|---|---|---|---|---|
| Chris | | X | X | X | X | | |
| Josh | X | | X | | X | X | X |
| Nick | X | | X | | X | X | X |