Group 2: Alex Medellin and Nick Rummel

CSC463

Dr. Girard

December 13, 2017

**Final Project**

1. For the robot’s design, the two reflective-opto sensors were mounted on the front of the robot to the left and right of the motor that controlled the robots claw. They needed to be spaced fairly close together so that either sensor would be able to detect if the robot veered off the line. The left and right photo sensors were placed on the left and right sided of the front of the robot and were used to detect whether a block was present at an intersection. In order to detect the light from the blocks, long side structures had to be constructed so that the photo sensors extended far enough out into each intersection of the arena. The front photo sensor was mounted at the front of the claw apparatus between the arms of the claw. The claw arms were designed to close around the block when the photo sensor registered a low enough value, indicating that the block was in between the arms of the claw.

For the code, the right and left reflective-opto sensors on the front of the robot were used to follow the line around the arena. The code uses fuzzy logic to follow the lines. The two reflective-opto sensors also work together to determine if it has reached an intersection. This is crucial to the project because it allows the robot to know where it is at in the arena. For example, the robot counts the number of intersections it has encountered to determine its location. The robot also used three photo sensors. One photo sensor determines if the robot is approaching the light mounted on top of the block. The values are set in code so that once the block is inside the claw, the claw will shut around the block. The other two light sensors mounted on the left and right side of the robot are used to determine at an intersection if a block is present on either side of the robot. This location is then stored.

1. For the robot, a claw was used to close around the block once located and open once it reached the drop off point. The claw was powered by a motor that was mounted to the front of the center of the robot. The motor axel drove a worm gear that drove two 16-tooth gears to the left and right of the worm gear. Attached to these gears were the arms of the claw and other mounting pieces to connect and better support the gears and arms of the claw. The arms of the claw were made out of two different types of angled pieces. The first that was attached to the gears was longer than the second which enclosed the front of the block. At the point where the two angled pieces meet a few additional pieces were mounted to enclose the back of the black. This design helped keep the block from moving around while being transported.

For the code, picking up and delivering the block was implemented using two states. The pick-up state assumed that it was already at the collection area near the first intersection. Using the block locations from the light levels detected earlier at each intersections, the robot determines which intersection it needs to return to and whether a block is located on the left and/or right. The light sensor will allow the robot to approach the block. Once the light level reaches a certain threshold defined in the code, the block will be in the claw and the claw will close by using a motor function. The robot will drag the block as it turns around, locates the intersection, and turns the opposite way it took to find the block. Utilizing fuzzy logic, the robot will continue to follow the line to the collection area at the end. It will move forward a short distance past the intersection. The motor function in invoked again but with the opposite level of power in order to release the block.The robot will back away from the block and turn around to do one of two things. First, it could increment the array index variable and locate the next block to collect. If all three blocks were collected, then the robot returns to the start point.

1. For the robot, the gearing used was mostly the same as the original Handy-bug design used in all previous projects. Two spacing pieces were placed on the inside of each of the front axles to help prevent the wheel axles from sliding out of place, which would misalign the gears of our robot and cause them to mesh. The speeds and turns the robot uses were chosen so that the robot operated as fastly and efficiently as possible, without subjecting the gears or other mechanics of the robot to unnecessary stress, and while still being able to detect and follow the line and detect and locate the three blocks.

For the code, we had to be careful with synchronizing the speed of the robot’s movements to how often a sensor (particularly the reflective-opto sensor) needed to retrieve a new value. This was crucial due to utilizing the fuzzy logic for line following. If the robot was moving too quickly, it would not turn quick enough to stay on the line. The code kept the speed at a default level of 60 for going downhill, a default level of 70 for going uphill, and a default level of 85 for turning. More speed was needed for going uphill, which is why there was an increase from going downhill. The turning levels are much higher than both the uphill and downhill levels because we had times for the 90 degree and 180 degree turns from previous labs.

1. For the robot, one issue that we ran into was the structural integrity of our wheel gearing. While operating on a constant incline or decline, additional stress was placed on the front wheels of the robot which caused the wheel axles to slightly slide out of place. This would cause the gear that drives our wheels to become misaligned with the gear that is driven by the motor and would cause the gears to mesh. As stated above, to help mitigate this two spacing pieces were placed on the inside of each wheel axle. This gave the axle less room to move around and helped prevent the axle from sliding out of place.

For the code, we initially wanted to use follow the line using one reflective-opto sensor. However, this proved to be difficult because the code was long and inefficient before even programming the other robot actions. We switched the code to use fuzzy logic and adjusted thresholds for the various sensors. Not only would the robot only turn if it moved onto the line but the two reflective-opto sensors would determine if the robot had found an intersection. The robot also ran into an issue where it would turn at an intersection too soon, missing the line that lead to a block. We used the code to move the robot forward before turning, allowing the robot to catch the edge of the line and find the block. To save time with the robot, we decided to allow the robot a run-through of all intersections to detect if blocks are present. Then, the robot would collect blocks starting from the top (the collection area). In order to resolve any issues with the block locations, the robot reverse-numbered the locations and turns for the block locations. For instance, if a block was found to the left of the first intersection at the bottom of the arena, the robot would store the location value in the array as the block to the right of the fifth intersection. Finally, we ran into issues where the robot would skip blocks when determining the locations or collecting the blocks. The issue was caused by incrementing and resetting the array index variable in multiple parts of the code. The location detection state was only allowed to use the array index as a count for the for loop. Then, the variable was reset when transitioning from the detecting state to the collection state. Only the collection state is able to increment the index for the location array. The delivery state could only use the index variable for value comparison. The issue was eliminated by limiting the places that the index variable could be changed.