Team Members: Keenan Griffin, Justen Stall
Course: 499/592
Assignment: Lab04

Task 1: Reimplement Lab02 functionality

For Task 1 we reimplemented the functionality of Lab 2. The reimplementation was actually quite straightforward and didn't provide any challenges. The new features from the createlib library were well documented and provided a lot of the functionality we had to implement ourselves for Lab 2. All of the sensor checks are now performed by the "get_sensors" function, which checks all sensors and conveniently loads them into a named tuple. This increased code readability and cut down on the time spent cross-referencing the iRobot interface specification. We also took advantage of the names for charging states to get rid of a convoluted "match" block. The led control task was also simpler than our Lab 2 implementation, but not as simplified as the sensor checking. We used the RepeatTimer built into createlib, rather than using the Periodic thread example from the threading lecture. The led controls are now handled by a wrapper function, increasing readability since the arguments are named. Most of our work on Task 1 was to get familiar with how the new createlib library works and what features are available.

Task 2: Drive with Bump, Wheeldrop, and Light Bump

Objective for Task 2 is to drive the robot forward until either the bump/wheeldrop or light bumper sensors go off. Our initial design for this functionality included some python threading to handle the driving state and sensor checking. This threading design allowed the robot to perform other tasks, like flashing LEDs or play songs while driving and sensor checking. Unfortunately the threading functionality was commented out in order to more quickly troubleshoot some errors with driving and sensing. In future labs we will reimplement the threading as it will provide an efficient way to maintain a driving state while sensor checking and performing other tasks.

Our team wrote two different functions; one to handle Bump and Wheeldrop sensors, then another to handle light bump sensors. Drive with Bump and Wheeldrop sensors is activated with "W " and will drive at a hard coded wheel velocity of 200mm/s until any of the sensor values returns "True". The Light Bump sensor drive works the same way; the robot drives at the same wheel velocity until the Light Bump sensor returns "True". Some initial challenges with implementing these two functions included how to represent the returned values from the sensors. The returned values are stored in a tuple as boolean values for Bump and Wheeldrop which made for a pretty straightforward implementation. However Light Bump does return integer values that increase the closer the sensor gets to an object. Initially our team considered setting a threshold that when the returned sensor values exceeded the threshold the robot would stop. A more simple implementation would be to use the tuple of boolean values, and that was the implementation our team went with. So the Light Bump functionality works like the Bump and Wheeldrop function in the way it drives and checks sensor data.

Task 3: Distance Drive

For Task 3, we were to implement a function to have the robot drive a set distance before stopping. We started with the same layout used for the Task 2 driving functionality. To keep track of the distance traveled, we decided the simplest way would be to check the elapsed time since the robot started driving. When our “goTheDistance” function is called, it initializes a start time representing the time when the robot started driving, and sets the distance traveled to 0 before starting the robot’s drive. Once the robot has started driving, the sensors are constantly checked as in the previous functionality, along with the elapsed time since the robot started driving. The distance traveled is calculated as the velocity times the elapsed time since the robot started driving. Once either the requested distance has been traveled or a bump or wheeldrop is registered, the drive loop will be exited, the robot is stopped, and the current values for the distance traveled and elapsed time will be printed. To easily incorporate the bump and wheeldrop sensor polling, a helper function was implemented which takes the sensor packet as an argument and returns a boolean that is True when any of the sensors have been triggered. To check the time, we used the time module, specifically the perf_counter function which returns the current time in seconds. Getting the current time in seconds simplified the distance calculation, since the robot’s speed is measured in millimeters per second. There were various ways we could have measured the distance traveled, and deciding on the approach to take was the most challenging part of the task, but we think our implementation is just about the simplest way we could implement it.

Our original design for Task 3 functionality utilized the driveBumpWheeldrop function to handle drive and sensor checking seeing as that was already implemented in the previous task. However some issues came up in the process of testing and troubleshooting driveBumpWheeldrop, so we moved away from relying on the same function and decided to implement Task 3 functionality on its own. The main issue with driveBumpWheeldrop may have been the threading we attempted to implement. Another issue could have been when we called the robot’s stop function.

Challenges for implementing Task 3 functionality were getting the distance calculation right and resuming drive when an obstacle is removed from the path. Our solution to the distance driven calculation is detailed above. Some other groups noted a distance sensor that is a part of the packet file. It’s possible the robot has on board sensors and functionality to determine distance traveled, and it’s possible using those sensors is more efficient. Given more time our team would look into using these features. The resuming drive challenge was solved by setting the self.driving boolean to false and not changing its status until all bump and wheeldrop sensors are False. It’s important that once the robot resumes drive that startTime is also reset because of how we measure distance with velocity over time. Essentially when the obstacle is removed the robot returns to a state similar to its initial drive state. The important distinction is currentDistance keeps the distance traveled so far.