Scott Dudley

**Sprint 4 Research/Report:**

***Servo Replacement / Configuration / Testing: (7 Hours)***

While building the robot for Sprint 3, I realized that there was a broken servo and ordered replacement servos. For this Sprint, I had to take apart some of the robot, replace the servo, then rebuild it. Now that all the hardware was completely ready, I configured the servos to be at 90 degrees as well as configuring the motors and camera on the raspberry pi. I also double-checked that everything was wired correctly. Before we started the movement testing with our network protocol, I decided to install the rest of the dependencies and finish the configuration for SunFounder’s code. SunFounder was the company that created the robot kit for the raspberry pi and provided their own code on running the on their github page. I followed the rest of the manual and tested their code out. Luckily the testing of their code proved the robot moved correctly as well as capturing the video feed correctly, because the other robot had its controls reversed. It turns out that the provided servos and motors were wired randomly. This mean that if the controls were reversed, we would have to manually change SunFounder’s configuration, meaning that we had to establish separate configuration files for each robot to ensure that they both moved the same direction with our own code. SunFounder’s manual and github pages for the PiCar-V are provided below:

PiCar-V Manual:

<https://www.sunfounder.com/learn/download/U21hcnRfVmlkZW9fQ2FyX1YyLjBfZm9yX1Jhc3BiZXJyeV9QaV9QaUNhci1WXy5wZGY=/dispi>

PiCar-V Git:

<https://github.com/sunfounder/SunFounder_PiCar-V>

<https://github.com/sunfounder/SunFounder_PiCar>

***Battery Life Sensor Research: (5 hours)***

Another problem that came up during our work with the hardware was the inability to see the battery life of the system. When the batteries were low, the robot would not listen to the correct commands and would almost do its own thing. Originally we did not understand what was causing the error, but recharging the batteries caused the problem to fix itself. I spent a heavy amount of time into research for battery life sensors. I found a few different options in the process. The original idea was to hook up a voltmeter or multimeter to the robot and manually measure the voltage of the batteries to figure out the battery consumption. While the idea was simple enough, it would not work because a voltmeter would be too big/heavy for the robot, which would affect the speed of the robot and increase the power consumption to move the extra weight, defeating the purpose of attaching the voltmeter. I did some research on battery life sensing components for a raspberry pi and I ended up finding a small circuit chip that could sense the voltage of the batteries and send it to the pi. This chip is called an INA219 chip. It would use simple wiring from the batteries to connect to the chip and an I2C port on the Robot HATS on the raspberry pi to connect to the pi. Unfortunately, the wiring requires a breadboard which causes the same problems as if we a voltmeter to the system. Since I was unable to find battery life sensing hardware that fit the requirements of our system, I researched if the raspberry pi could internally sense its own battery life. The raspberry pi has to be disconnected from a power source, and the batteries provide a power source for the Robot HATS circuit which controls the whole system, including the raspberry pi. Therefore, as the system’s battery supply reduces, so does the raspberry pi’s. I searched around for linux commands that would sense the current battery life of the raspberry pi, but the only command that measures voltage of the raspberry pi only measure’s its core voltage, only measuring the voltage of the CPU, which does not give enough information about the battery life of the overall system. The command that I found was ‘vcgencmd measure\_volts core’. Ultimately, we decided to scrap the idea of a battery life sensor since we could not find a viable way to implement the idea. The references of all of my research on battery life detection are provided below:

INA219 Chip Installation Tutorial for Raspberry Pi:

<https://www.rototron.info/raspberry-pi-ina219-tutorial/>

<https://www.rototron.info/wp-content/uploads/INA219_datasheet.pdf>

Other Implementation with Breadboard:

<https://raspberrypi.stackexchange.com/questions/42832/how-to-check-how-much-battery-is-left-on-my-raspberry-pi-in-the-terminal>

Research for Internally Measuring Raspberry Pi Voltage:

<https://www.raspberrypi.org/forums/viewtopic.php?t=75552>

<https://raspberrypi.stackexchange.com/questions/13886/how-to-know-how-much-battery-power-is-remaining>

<https://www.reddit.com/r/raspberry_pi/comments/2gm8vj/is_there_a_way_to_monitor_the_input_voltage/>

***Sound/Light Research: (2 hours)***

While I was on the topic of battery life sensor research, I decided to also do a little research on some other topics we were on the verge of completely closing. There was an idea to generate sound from the robots like a car horn. This was mostly a cosmetic idea but could be used for sending the user different notifications, such as if the robots are too far apart or if the robots are too close. We ended up scrapping this idea because it would not be possible without adding speakers to the robots as well as breadboards for the wiring. This causes the same problems as adding a regular voltmeter to the system. There would not be enough room to add the breadboards, and both the speakers and breadboards would add too much weight to the system. I also came up with an idea of our leader-follower system working in the dark by attaching lights to the front of the bots, like headlights. Unfortunately, the lights would also require a breadboard, which we cannot fit onto the current system. Both of the implementations for sound and lights can use the I2C ports on the robot’s Robot HATS to connect to the raspberry pi. Though the research on the battery life detection, sound, and lights did not end up being fruitful for the current system, we may be able to find a way to implement some of these ideas if we end up ahead of our sprint schedule. Since there are only two I2C ports on the Robot HATS, we would only be able to implement 2 of the 3 ideas. We would also have to find a way to connect a breadboard to the overall system without adding too much weight or strain to the robots. The best implementations that I could find for adding sound and light to the robots are provided below:

Research on an Implementations of Sound:

<http://www.audioinjector.net/#!/rpi-hat>

Research on an Implementation of Lights:

<https://learn.adafruit.com/matrix-7-segment-led-backpack-with-the-raspberry-pi/overview>

<https://cdn-shop.adafruit.com/datasheets/ht16K33v110.pdf>

***Start on Final Documentation: (2 hours)***

Lastly, I laid out the final documentation, as presented in the syllabus, and moved all of our current documentation to the corresponding areas. I will continue to work on the final documentation for Sprint 5. The ‘Final Documentation’ word document is provided in the Deliverables directory of Sprint 4.

***Total Work for Sprint 4: 16 hours***