Project 1 Report

CSC 460

February 10, 2016

GROUP 18

Joshua Portelance - V00824005

Jakob Roberts - V00484900

Table of Contents

1. Introduction

Project one consisted of two main phases with the combined goal of introducing us to the hardware we will be using for the later projects, and building the physical system and coding the tasks for project 3. The physical system that we created during this project was a remote control Roomba and pan-tilt servo that accepted and sent commands via Bluetooth from a base station with two joysticks.

The first phase of this project was to connect all of the components excluding the Roomba and the Bluetooth devices into a single base station. The finished phase 1 system would allow you to control the pan-tilt servos and the attached laser with a single joystick.

For the second phase of project 1, we split the system built in phase 1 into a base station and a remote station and introduced a time-trigger scheduler. The base station would interact with two joysticks, a bluetooth module, and an LCD screen; the remote station would interact with the pan-tilt servos, laser, Roomba, Bluetooth module, and photoresistor. Using the Bluetooth modules these two stations would communicate with each other using a protocol we created; the base station would send position, velocity, and on/off information to the remote station to control the servos, laser, and Roomba, and the remote station would send back the status of the light sensor.

In the following sections, this report will go into detail for each of the previously mentioned phases describing the system overview, the implementation, and any issues we ran into.

2. Phase 1

The goal of phase 1 of this project was to familiarize ourselves with the majority of the hardware we will being using throughout this project and project 3. The components that were used for this phase are:

- One Arduino Mega 2560
- One Arduino LCD KeyPad Shield
- One AC adapter
- One Laser pointer
- One photoresistor
- One breadboard
- Two analog joysticks
- Two servo motors within a pan-and-tilt
- Miscellaneous wires
- Miscellaneous resistors

Using all of the components listed above the goal was to construct a single functioning base station comprised of all of these components. The following sections will describe the process of

designing and assembling the overall system, as well as any issues or notable events that came up during the process.

2.1. System Overview

The overall system for phase 1 was rather messy because of the large amount of components connected to a single Arduino. A high level diagram of the layout of the components previously listed can be seen in Figure 1 below.

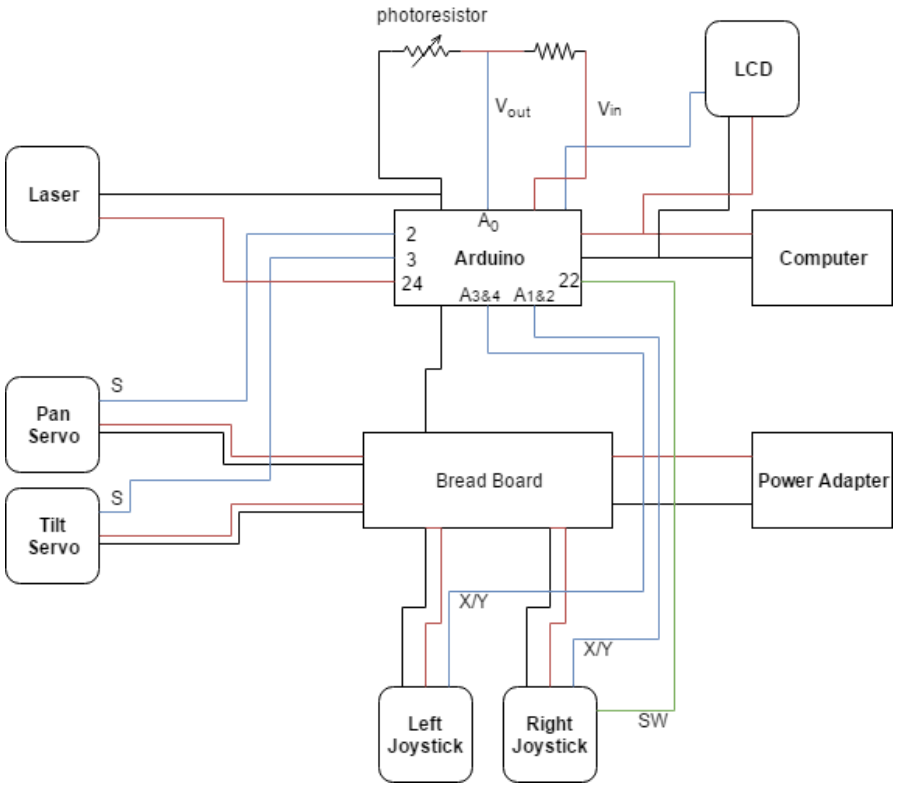

Figure 1. High level overall of the system for phase 1, showing all components.

As you can see in Figure 1, for phase 1 of this project, we were using two separate power sources, one from the computer to the Arduino via the USB cable, and another from an AC adapter into power supply integrated onto the breadboard.

When implementing the system shown in Figure 1, a few changes were made. The photoresistor circuit was moved into the breadboard, and all cables going from the laser, servos, and joystick directly to the Arduino where all first connected to the breadboard, and then any necessary connections were made between the breadboard and the Arduino. Figure 2 below shows the final system assembly with all of the connected components.

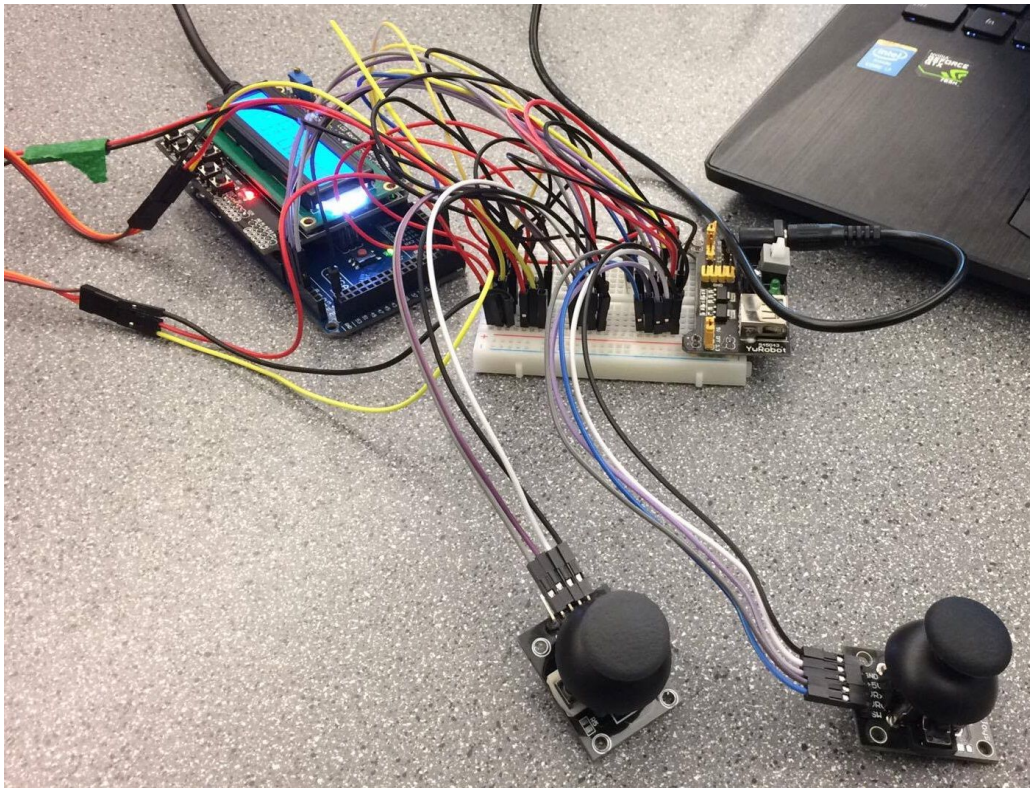

Figure 2. Final system assembly for phase 1, servos and laser out of shot to the left.

As you can see in Figure 2, everything is first connected through the breadboard and then any necessary connections were made to the Arduino. The servos inside of the pan-tilt-kit and the laser are outside of the shot to the left, and the photoresistor circuit which is hard to see will all be covered in more detail in the following section.

2.2. Joystick, Servo and Laser

The process of setting up the joysticks, servo, and laser went pretty smoothly except for two issues which will be explained later in this section. The goal was to have the joysticks control the movement of the two servos inside of a pan-tilt kit, and also control the power to a laser that will be mounted to the tilt platform on the servos kit.

The first step to this process was to set up the servos inside of the pan-tilt kit. In our case we used the purchased black kits with the larger servos instead of the 3D printed kit. Because of this, in order to get the servos to fit, we had to modify the pan-tilt kit with a knife and cutters. After assembling the servos the laser was mounted on top of the kit with tape. This can be seen in Figure 3 below.

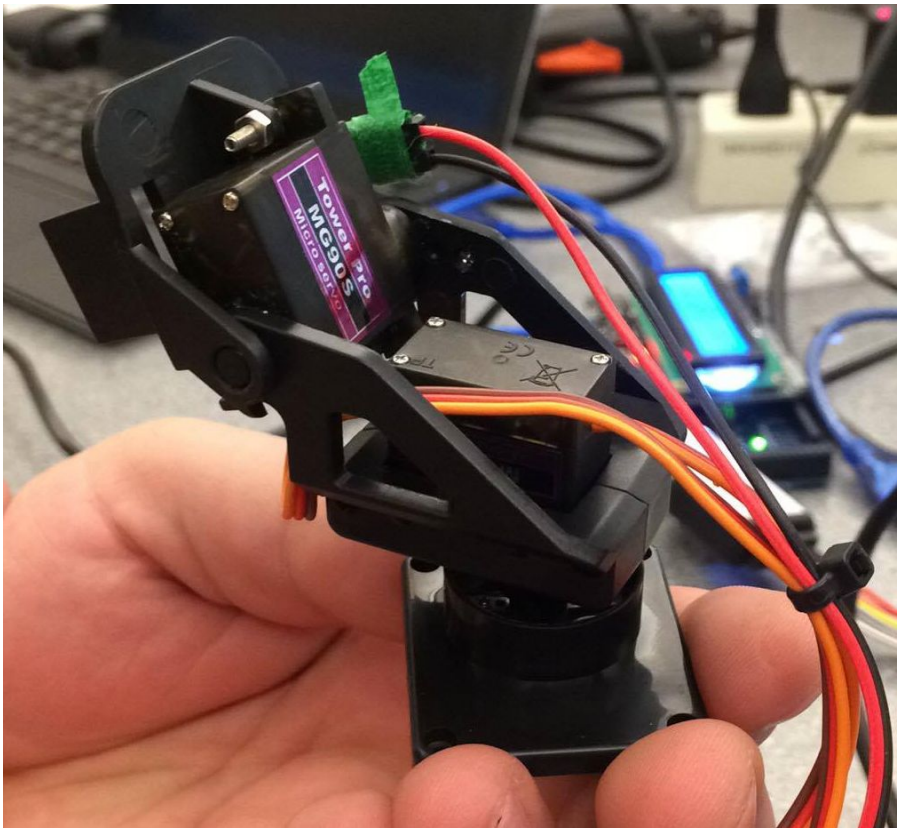

Figure 3. Pan-tilt kit with servos installed and laser attached to the tilt platform with green tape.

After completing the assembly shown in Figure 3, the next step was to wire the servos, laser, and the joysticks to power sources and to the Arduino.

The wiring of the servos, laser, and joysticks to a power source and the Arduino went pretty smoothly despite initially mis-wiring the power and control wires for the servos. The decision was made for this phase to run all wires through the breadboard first and then run any necessary wires to the Arduino from there. The wiring of these components can be seen in Figure 2 in the System Overview section.

The final step was to design and implement the code for controlling the servos and laser with the input up the joysticks. Links to the source code are available in the Links section of Appendix A.

The laser was controlled by the button built into the joystick which acts as a switch. When the switch is pressed the laser is toggled on and stays on until the switch is pressed again.

The servos were controlled by making small changes to the position of the servos at a high frequency. Input from the joysticks was turned into a value of -5 to 5 for both axis and then this value was then added to the current position of the servo. This allowed us to have variable

speed input for the servos while also keeping the speed of the movement at a controllable level based on the combination of the value and the sampling frequency.

The final issue we ran into was while testing the movement of the servos was that the entire servo system would work flawlessly up until a point where in the servos would simply veer off to their max position and jitter without control. This flaw was rather hard to find and took several hours of our time which we spent debugging and rewriting code and double checking and redoing the wiring to no avail. In the end we found that the problem was with the physical assembly of the pan-tilt kit itself, there was a point in the tilt platform that would make contact with the pan platform when the tilt servo was at a certain angle. This contact caused massive power surges throughout the system which caused garbage input values to reach the servos causing non-deterministic behaviour. This was solved by cutting off the piece of plastic that was making contact. After this was solved everything worked perfectly, as planned.

2.3. Light-sensing and LCD Feedback

The task of setting up the light-sensing circuit and having it send any feedback to the LCD was pretty straight forward. The goal was to use a photoresistor to measure the amount of light that was hitting the resistor and if it was under direct light from a laser the Arduino would update the LCD display, saying that it was detecting light.

The light-sensing circuit itself is just a basic voltage divider, which uses a single static resistor and a photoresistor. The implementation of this circuit with annotations can be seen below in Figure 4.

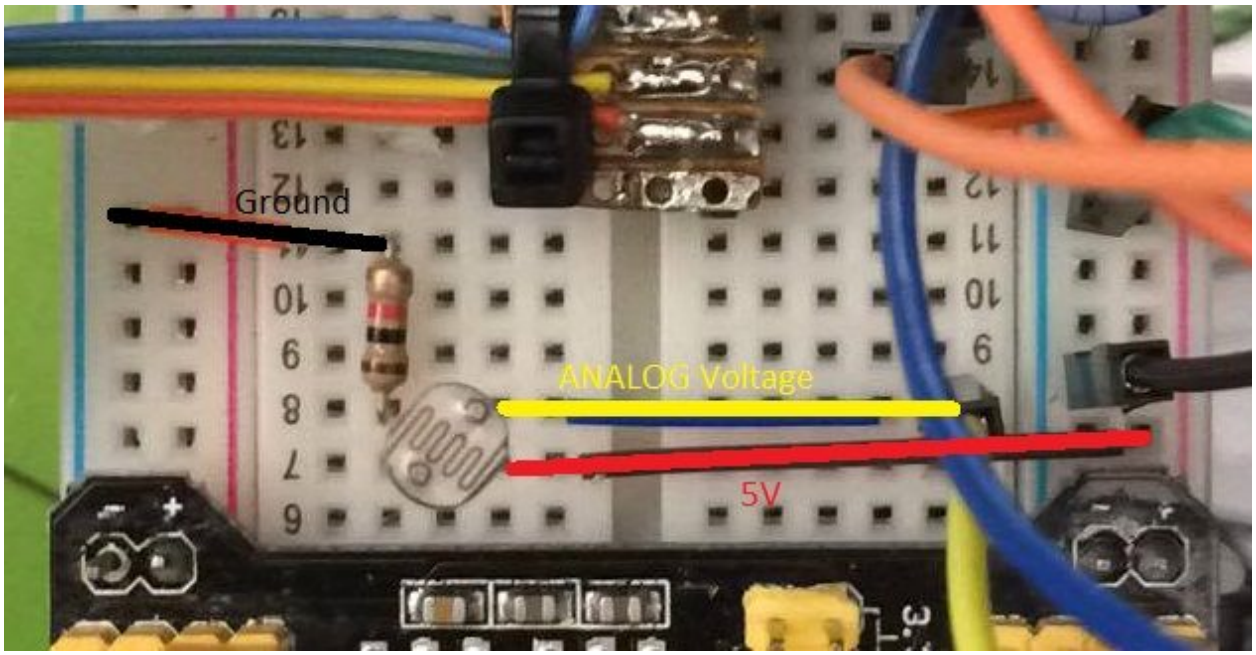

Figure 4. Light-sensing circuit using a photoresistor and a static resistor in a voltage divider setup.

As you can see in Figure 4, the analog signal that we are using to measure the amount of light the photoresistor is detecting comes from right after the photoresistor, the remaining part of the circuit is there to create a voltage divider.
By inspecting the values coming from the analog signal, we were able to determine that if the signal we were receiving had a value of over 500 then light was being detected. If this occurred the code running on the Arduino would update the LCD from displaying “LIGHTOFF” as shown in Figure 5, to “LIGHTON!”.

Figure 5. Showing the light-sensing feedback on the LCD.

Overall, this section of phase 1 went very smoothly. There were virtually no issues other than accidentally wiring the light-sensing circuit incorrectly, the small mistake with the servo wires, and the non-deterministic servo behaviour, but once everything was set up correctly it worked as expected.

3. Phase 2

The goal of phase 2 of this project was to take our gained knowledge from Phase 1 and add onto it through the use of more hardware. The components that were used for this phase are:

- Two Arduino Mega 2560
- One Arduino LCD KeyPad Shield
- One AC adapter
- One Laser pointer
- One photoresistor
- Two breadboard
- Two analog joysticks
- Two bluetooth modules (one master and one slave)
- Two servo motors within a pan-and-tilt
- One Roomba connection Kit
- One Roomba Vacuum
- Miscellaneous wires
- Miscellaneous resistors

Using all of the components listed above the goal was to create a base station and remote station that could do the same functionality as in Phase 1 + more, but be able to do it between Arduinos via Bluetooth. The following sections will describe the process of designing and assembling the overall system, as well as any issues or notable events that came up during the process.

3.1. System Overview with Time-Triggered Scheduler

The overall design of Phase 2 required much less dense wiring and was a much cleaner layout. We also made the switch from a delay architecture to a TTA (Time Triggered Architecture) in order to have smoother movements of our systems. The benefit of a TTA over a delay architecture is that with a delay architecture, you are committed to the single delay for all components. Unfortunately a single delay is not ideal as some things require to be updated much more frequently than others.

3.1.1. Base Station

The base station was the heart of the system as it ran the TTA and kept track of all the tasks. As it was mainly the communication hub and did not interact with any of the servos and other hardware, it only had the two input joysticks, the master bluetooth module, and the LCD screen to display statuses as shown in Figure 6 below.

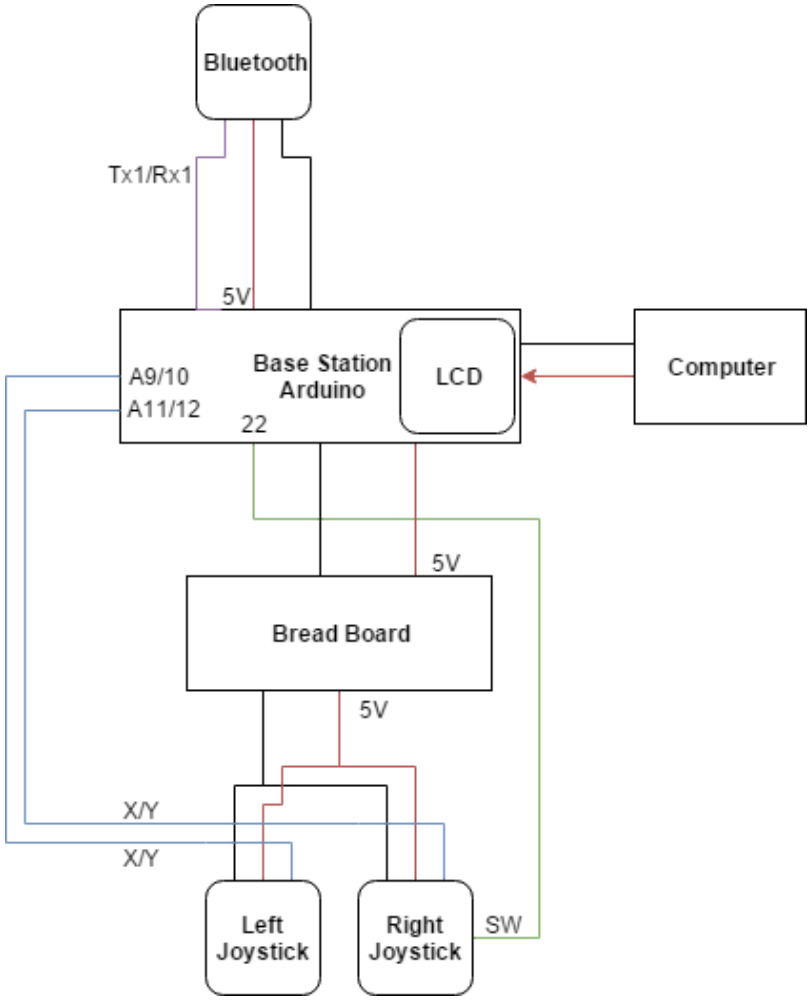

Figure 6. High level diagram of the base station system for phase 2.

Initially when wired, we did not consider the position most comfortable for the user to hold the joysticks and had to modify the code in order to accommodate the positioning of the user's hands on the joysticks. The only power provided to the base station was from the connected computer. As you can see from the above wiring diagram and the picture shown in Figure 7, the setup is quite simple and relied mostly on the code that was sent to the Arduino.

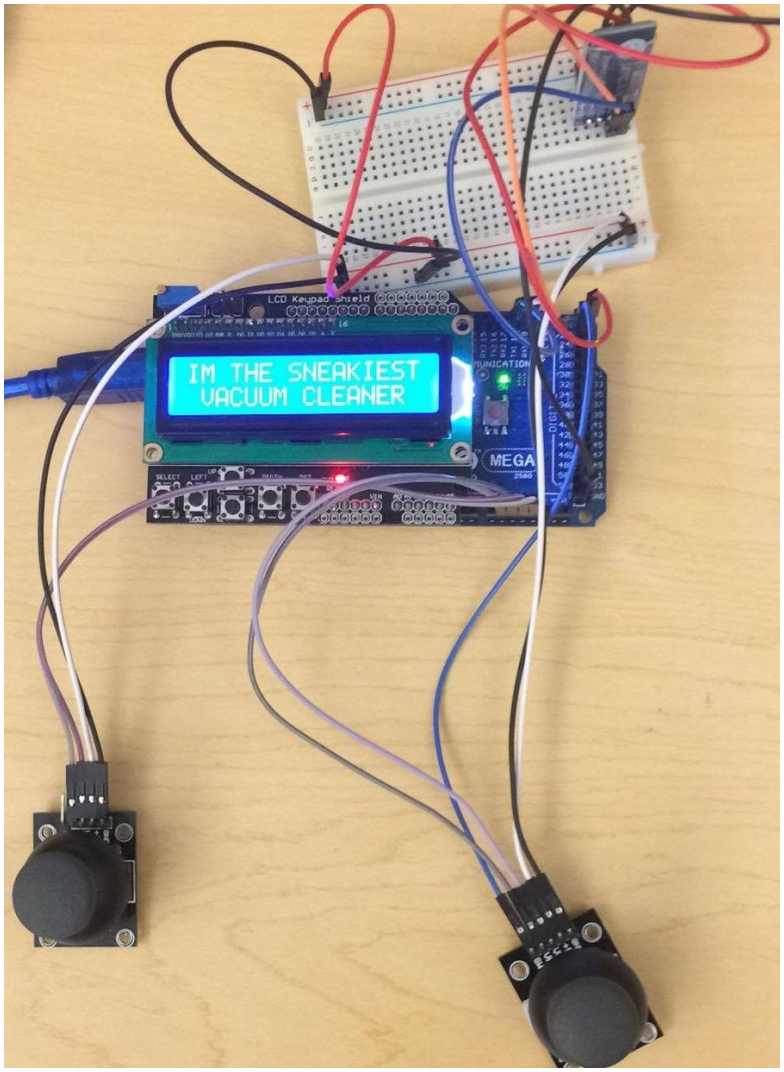

Figure 7. Overview of the final assembly of the base station.

Unlike described in the wiring diagram in Figure 6, the actual wiring done in Figure 7 is slightly different as we wanted the bluetooth module to be directly connected to the breadboard to prevent any movement or disconnection. By wiring it this way, we also had further length available to the joystick wires which made controlling things much easier.

Initially the display had output values that we used for debugging, but then we made more comical reactions to be displayed depending on the status of the program. Unfortunately we discovered that the LCD does not like to display the exclamation mark and ran into upload errors to the Arduino.

3.1.2. Remote Station

The remote station was the main control for all the moving parts for the project. It took in data via Bluetooth from the base station, parsed it, then also using TTA it made changes to the physical hardware. It controlled the movement of the roomba, the pan-tilt servos, the laser, the photoresistors light detection, and return data to the base station to kill input if the photoresistor was triggered.

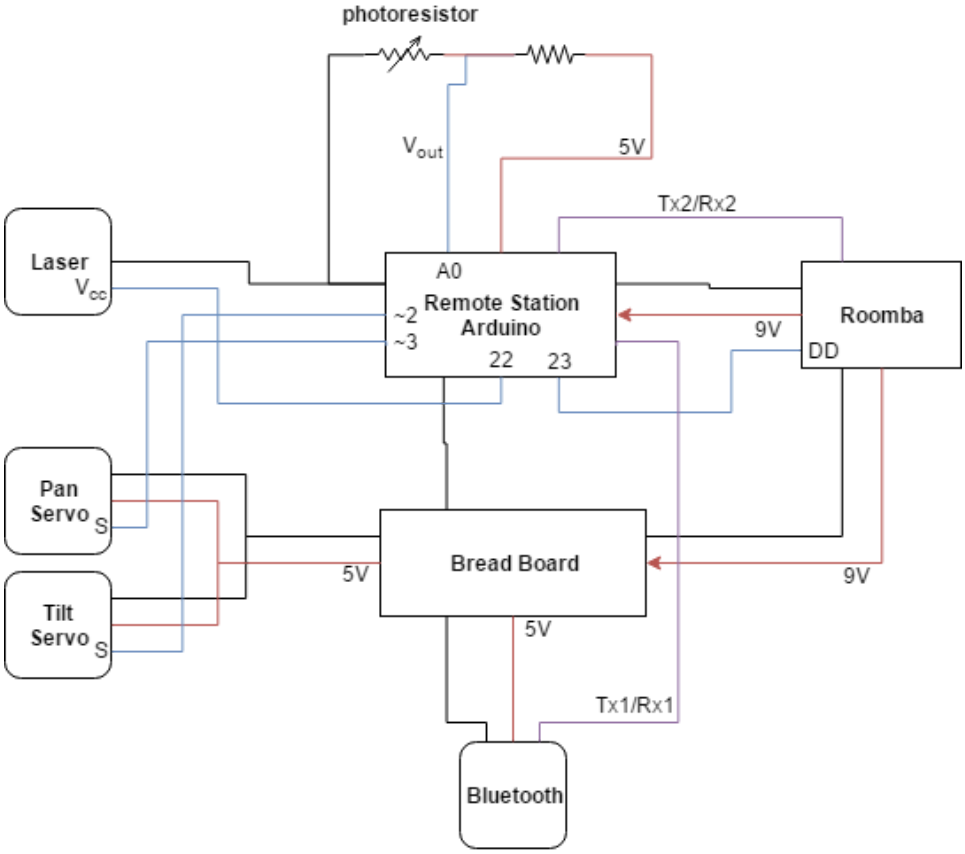

Figure 8. High level diagram of the remote station for phase 2.

Above in Figure 8, you can see that we did not use a LCD for this station as we were limited with PWM connections available to us and it was simpler to connect things to the arduino without the LCD. The laser was attached to the pan and tilt servos, and the photoresistor and bluetooth were both connected to the breadboard. The 3 parts were then connected to the roomba using sticky velcro as shown in Figure 9.

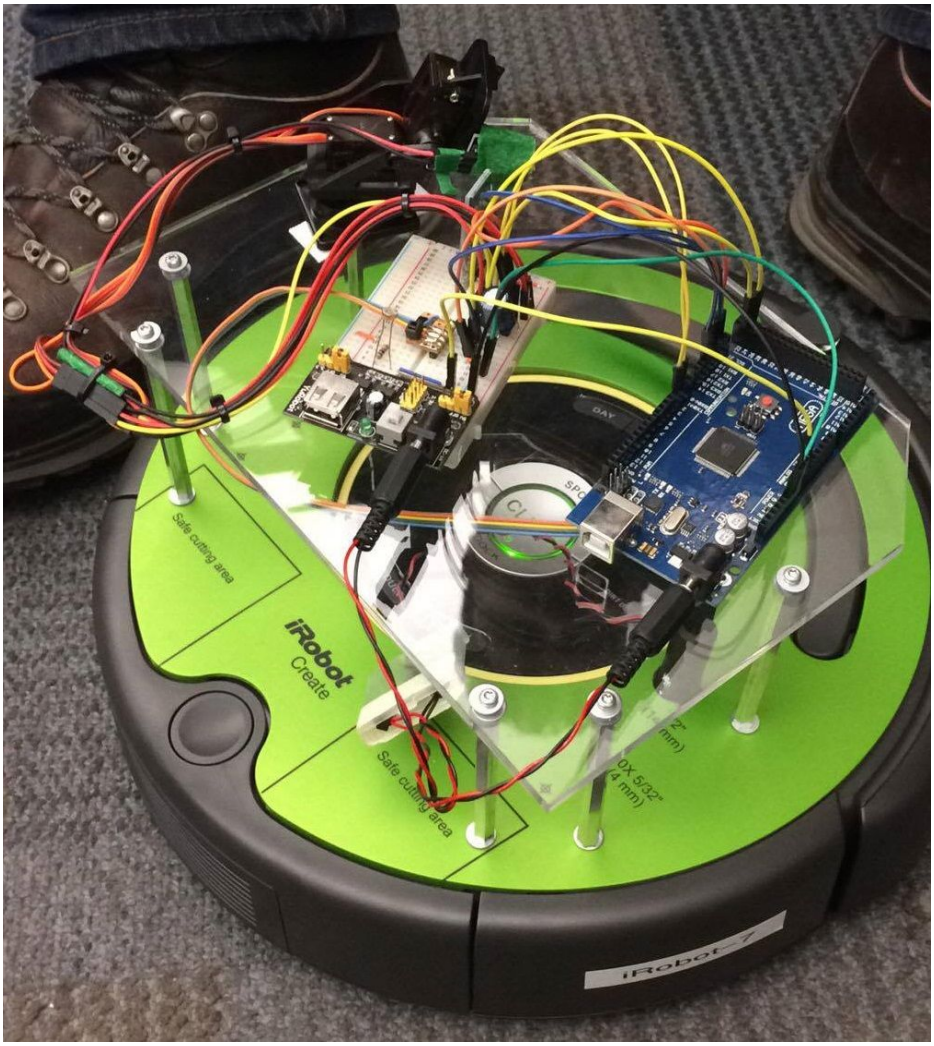

Figure 9. Overview of the final assembly of the remote station.

In the above, Figure 9, you can see the 3 modules attached to the roomba, wired up, and powered on. In the future it would have been nice to have a better mounting fashion for the pan-tilt servo so it was more functional. There were no problems with the remote station except for some bluetooth issues that will be discussed in the following section.

3.2. Bluetooth Radio Interfacing and Integration

The wireless communication between the base station and the remote station was done via bluetooth. Initially this seemed like a daunting task, but upon successfully linking them and reading serial values out from their connection, it was relatively simple. Unfortunately we ran into the problem where the baud rates did not match between the master and the slave bluetooth modules and until it was fixed, we were convinced we were cursed with broken modules.

3.2.1. Setup

After running the bluetooth initialization scripts and successfully discovering the baud rate connection problem, we finally got our bluetooth modules to connect and talk to each other.

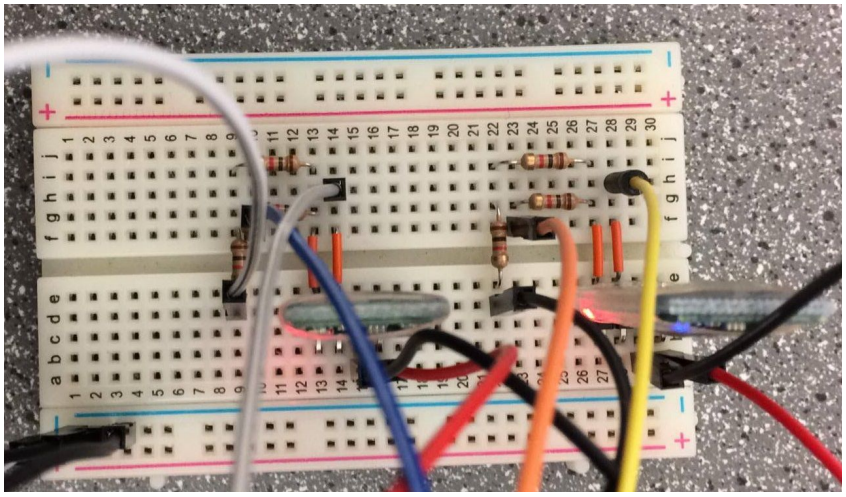

Figure 10. Configuring the bluetooth modules.

As shown in Figure 10, we had both bluetooth modules connected to the same board for the pairing and configuring process. We then started to monitor the data via serial and realized that we were getting some garbage data. This was due to the fact that we initially wired the bluetooth modules in assuming we had an input of 3.3V and not 5V.

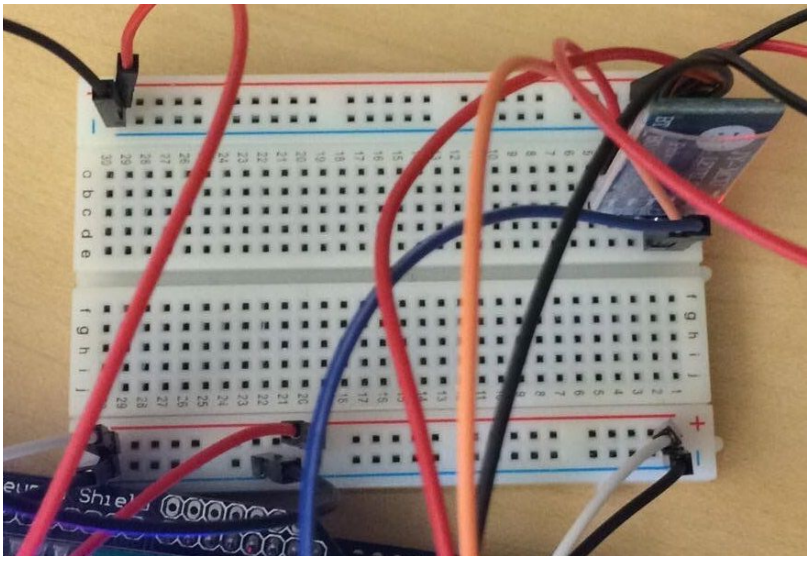

Figure 11. Base station bread board with bluetooth module setup.

We then removed the resistors and connected the bluetooth directly (as you were supposed to for a 5V connection) then separated the bluetooth modules to their appropriate base stations

began to send bits of data back and forth to test, but unfortunately we were getting strange data output on the remote station side.

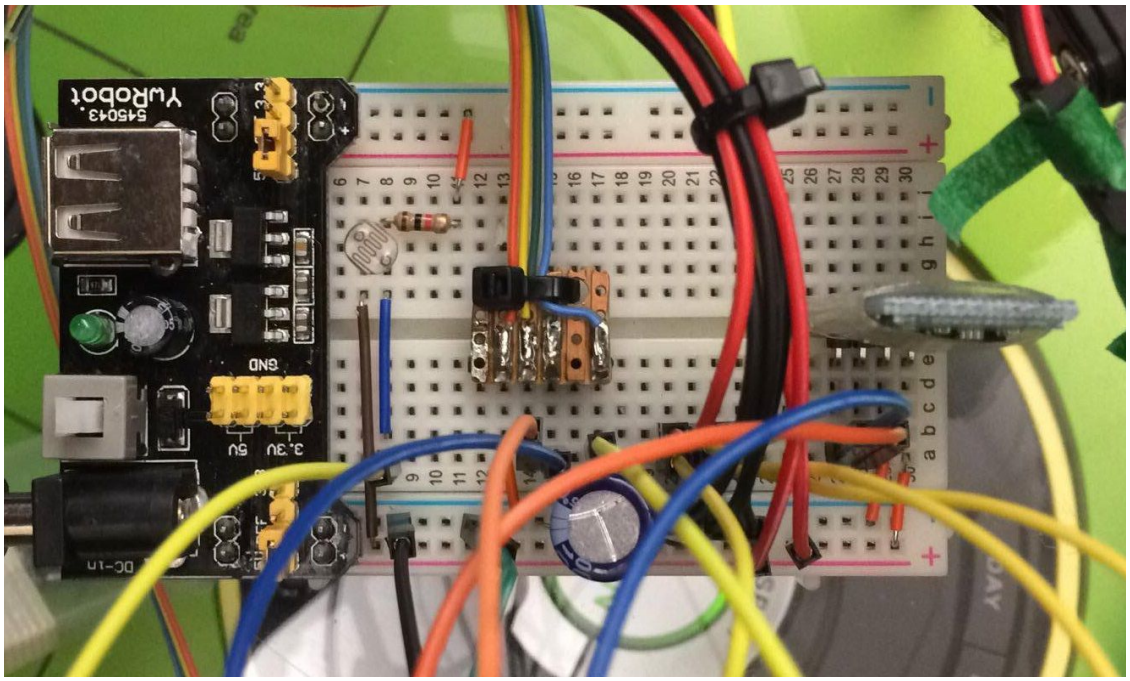

Figure 12. Remote station bread board with bluetooth module setup.

We discovered that the source of the garbage data being sent/received by the remote station's bluetooth module (Figure 12) was due to a grounding problem as the Arduino and the breadboard were not sharing similar signal grounds. A single wire fixed the issue and we were sending and receiving perfect data.

3.2.2. Packet Specification

In order to transfer data between the base station and the remote station we decided to design packets that would allow us to send all of our data at once. The bytes of the packets were framed using a # character as a header and a % character as a footer. In the case that more than one piece of data was being transmitted, the two data segments would be split up using the | character.

In the case of transmissions from the base station to the remote station, there were five values that needed to be passed over. The structure of the packets that were being sent from the base station can be seen in Figure 13.

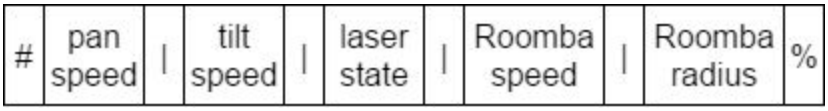

Figure 13. Structure of the packet that was used to send information from the base station to the remote station.

As you can see in Figure 13, each value is divided using a | character and the start and end of the packet are signaled by using the # and % characters for the header and trailer.

In the case of transmission from the remote station to the base station, there was one one value that needed to be passed over. Because of this the packet design was much simpler in this case. The structure of the packets that were being send from the remote station are shown in Figure 14.

Figure 14. Structure of the packet that was used to send information from the remote station to the base station.

As you can see in Figure 14, because there is only one value being sent over there is no need for the | character; therefore, only the header and trailer characters need to be used to frame the data correctly.

To decrease the possibility of the either the base station or the remote station overflowing the buffers we ensured that the frequencies for the transmission tasks were longer than those for the receiving tasks. This allowed us to reliable transfer data between the two stations with minimal to no corruption.

3.3. Timing Specification and Measurements

To measure the actual runtime of each task that was running on either the base or remote station, we used digital logic analyzers. At the beginning of each tasks that was being ran, the tasks would set a digital output pin to high, and then once the task was complete it would set this same pin to low. Setting the digital pins to high or low takes virtually no time, so this was able to provide us with a very accurate measurement of the runtime of tasks.

The following two sections will go into more detail about the measurements taken from the base station and the remote station, as well as calculations of the CPU utilization of each program.

3.3.1. Base Station

The base station had a total of 8 tasks to run to provide all necessary functionality. They controlled the sending and receiving of data via bluetooth, the joystick axis controls, the pan-tilt servo movement speeds, the roomba movement speeds, the LCD printouts, and the switch on the right joystick. Unfortunately due to having 8 tasks, the logic analyzer could not display the

idle pin time as we had no further connections on the logic analyzer. By using the measurement method described earlier in section 3.3, we were able to obtain the following graph of the tasks running, shown in Figure 14, with more detail provided in Table 1.

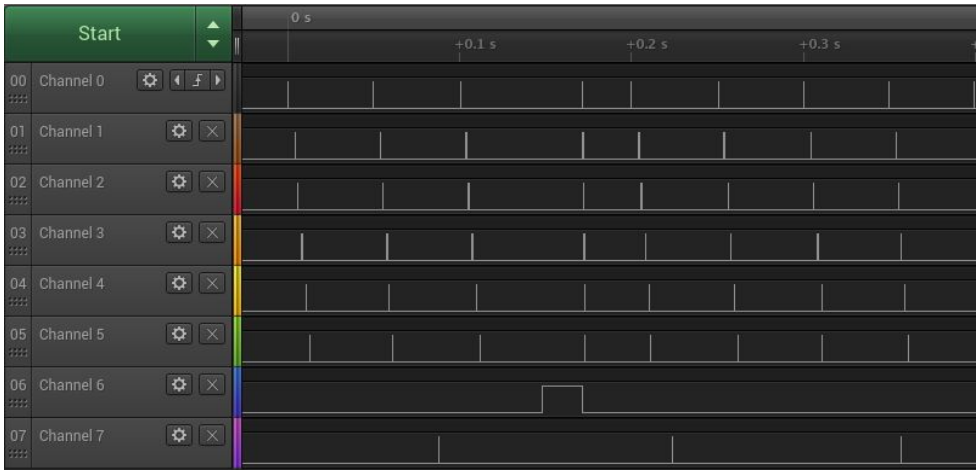

Figure 14. Show the output of the digital analyzer on the base station for a sample of 0.4 seconds.

Channel #	Task	Offset (ms)	Frequency (Hz)	Measured Runtime (ms)
0	bt_getData	2	50	0.05
1	bt_sendData	6	50	0.54
2	checkLeftJoyAxis	8	50	0.27
3	checkRightJoyAxis	10	50	0.27
4	getLaserSpeed	12	50	0.08
5	getRoombaSpeed	14	50	0.08
6	LCDprint	0	500	23.68
7	rightJoySwitch	0	135	0.03825

Table 1. Shows the offset, frequency, and measured runtime of each task running on the base station.

As you can see in Table 1, the runtime of each task is extremely fast which is why it is possible to run each task together with such a small offset. We could have likely combined a couple of the tasks, but as we wanted to maintain expandability, we kept the tasks separated.

Furthermore, using the frequency and runtime data in Table 1, it is possible to get a close estimated CPU utilization by using a standard time frame of 13,500 milliseconds. This can be done by using equation below with the applicable data for each task in Table 1.

$$\%\ utilization = \frac{\Sigma(\frac{13500\ ms}{freq} \times runtime)}{13500\ ms}$$

By doing this it was found that the program utilizes 7.34% of the CPU, therefore the rest of the time the CPU is idle. Most of the runtime was spent updating the LCD.

3.3.2. Remote Station

The remote station had a total of 6 tasks to run to perform all of the necessary duties to provide control of the servos, laser, and Roomba, while also checking for light. By using the measurement method described earlier in section 3.3, we were able to obtain the following graph of the running tasks, shown in Figure 15, with more detail provided in Table 2.

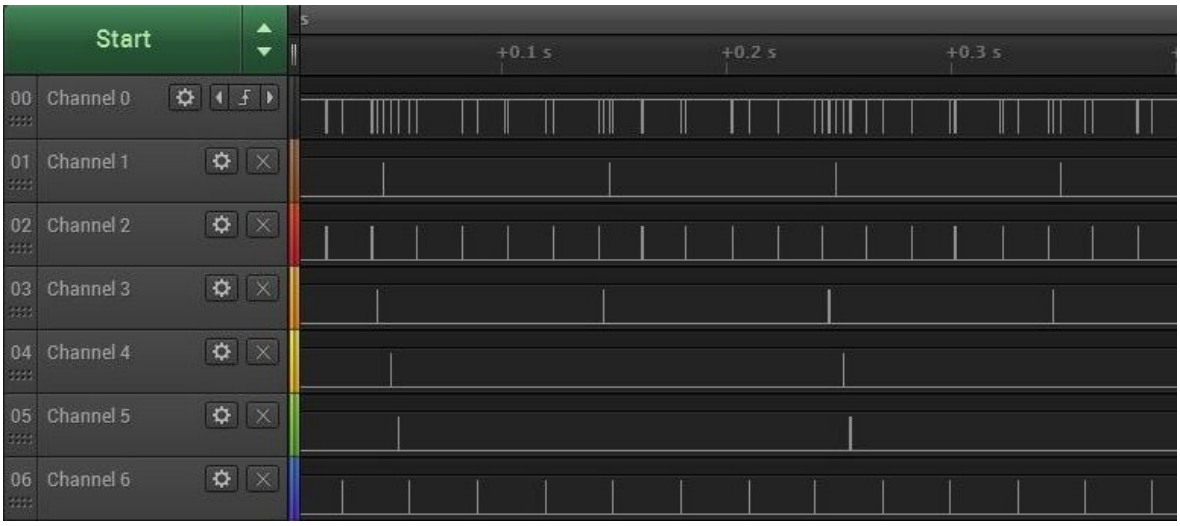

Figure 15. Show the output of the digital analyzer on the remote station for a sample of 0.4 seconds.

Channel #	Task	Offset (ms)	Frequency (Hz)	Measured Runtime (ms)
0	Idle	-	-	-
1	Update Laser	9	100	0.0045
2	Update Servos	3	20	0.0460
3	Update Roomba	6	100	0.1258
4	Check Light Sensor	12	200	0.1168
5	Transmit	15	200	0.1315
6	Receive	0	30	0.0110

Table 2. Shows the offset, frequency, and measured runtime of each task running on the remote station.

As you can see in Figure 15, channel zero shows that for the majority of the time the system is idling, and that the rest of the tasks that need to be ran are nicely distributed to avoid any conflicts or issues from jitter. The actual runtime for each of the tasks on the different channels can be seen in Table 2.
As you can see in Table 2, the runtime of each task is extremely fast which is why it is possible to run each task together with such a small offset.

Furthermore, using the frequency and runtime data in Table 2, it is possible to get a close estimated CPU utilization by using a standard time frame of 600 milliseconds. This can be done by using equation below with the applicable data for each task in Table 2.

$$\%\ utilization = \frac{\Sigma(\frac{600\ ms}{freq} \times runtime)}{600\ ms}$$

By doing this it was found that the program utilizes 0.52% of the CPU, therefore the rest of the time the CPU is idle.

4. Appendix A

4.1. Source Code Links

4.1.1. Phase 1

https://github.com/JoshuaPortelance/CSC460-Spring2017/blob/master/project_1/phase_1/phase1.ino

4.1.2. Phase 2

4.1.2.1. Base station

https://github.com/JoshuaPortelance/CSC460-Spring2017/blob/master/project_1/phase_2/base_station.ino

4.1.2.1. Remote station

https://github.com/JoshuaPortelance/CSC460-Spring2017/blob/master/project_1/phase_2/remote_station.ino