Project 3 Report

CSC 460

April 7, 2016

GROUP 18

Joshua Portelance - V00824005

Jakob Roberts - V00484900

Table of Contents

1. Introduction 3

2. Hardware 3

3. Phase 1 System Design 3
- 3.1. Base Station 3
  - 3.1.1. System Overview 4
  - 3.1.2. Tasks 4
    - 3.1.2.1. BlueTooth Send Data 5
    - 3.1.2.2. Check Joysticks 5
    - 3.1.2.3. Get Speeds 6
    - 3.1.2.4. Right Joystick Switch 6
  - 3.1.3. Prioritization 7
- 3.2. Remote Station 7
  - 3.2.1. System Overview 7
  - 3.2.2. Tasks 8
    - 3.2.2.1. Receive Transmission 8
    - 3.2.2.2. Update 9
  - 3.2.3. Prioritization 10
- 3.3. Problems 11
  - 3.3.1. Controlling Servos Through PWM 11
  - 3.3.2. Not Realizing We Needed to Create Our Own UART Buffer in Software 11
  - 3.3.3. Analog to Digital Conversions for Joysticks 11
  - 3.3.4. Handling Corrupt or Incomplete Packets 11

4. Phase 2 System Design 12
- 4.1. Subsumption Architecture 12
- 4.2. Tasks 12
  - 4.2.1. Check For and Handle Physical Wall Collision 12
  - 4.2.2. Check For and Handle Virtual Wall Collision 13
- 4.3. Prioritization of Remote Station Tasks 14
- 4.4. Problems 15
  - 4.4.1. Incorrect Data Being Received From the Roomba 15
  - 4.4.2. Hidden Wall Detection FSM 15

5. Testing 16
- 5.1. Manual Control Over Roomba 16
- 5.2. Manual Control Over Servos 16
- 5.3. Manual Control Over Laser 16
- 5.4. Physical Wall Detection 17

5.5. Virtual Wall Detection 17

6. Appendix 18

6.1. Code 18

6.2. Hardware 18

6.2.1. ATMega2560 18

6.2.2. Bluetooth Radio 18

6.2.2.1. HC-05 18

6.2.2.2. HC-06 18

6.2.3. Analog Joysticks 18

6.2.4. Servo Motors 18

6.2.5. Laser 18

6.2.6. iRobot Roomba Create 2 18

6.3. Pictures 19

6.3.1. Base Station 19

6.3.2. Remote Station 20

1. Introduction

The main objective of this project was to take the RTOS created in Project 2 and re-implement the system that was created in Project 1 with some more additions/modifications. The physical system that we re-created during this project was a remote control Roomba and pan-tilt servo that accepted commands via BlueTooth from a base station with two joysticks. This project was divided into two different phases: the first phase was to be able to control all objects on the Roomba with the remote station using the base station via bluetooth, and the second phase was to implement semi-autonomous capabilities in the Roomba. The goal of phase 2 was to have the remote station takeover control of the robot in the event that the Roomba detected a collision with either a physical wall or a virtual wall.

2. Hardware

The hardware that we used for project 3 was very similar to what we used in project 1; however, in project 3 we had no use for LCD monitors, photoresistors, or AC adapters. The components we used for project 3 are:

- Two Arduino Mega 2560
- One Laser pointer
- Two breadboard
- Two analog joysticks
- Two servo motors
- Two BlueTooth modules
- One Roomba Create 2
- Miscellaneous wires
- Miscellaneous resistors

This hardware was split between two different stations, a remote station, and a base station. Links to many of the major components listed above can be found in the Appendix in Section 6.2.

3. Phase 1 System Design

The first phase consisted of re-implementing the same functionality as in project 1: the ability to remotely control a Roomba and a pan/tilt servo setup with a mounted laser. In part we split the workload into two separate stations: the base station to take care of user input and then to transmit to the remote station to control the Roomba and laser mechanism. The following sections detail the two different stations and their functionality.

3.1. Base Station

The base station is the main control unit for user input. It was in charge of collecting all the input data and sending it to the remote station to be used. The base station processed and parsed the input information into the appropriate quantities before formatting it into a packet and sending it over a BlueTooth serial connection.

3.1.1. System Overview

The base station consisted of two joysticks: one to control the movements of the Roomba, and another to control the movements of the pan/tilt servos and the laser. All this was connected to the ATMega2560 board and it used a BlueTooth module to transfer information. The overall architecture of the base station can be seen below in Figure 1.

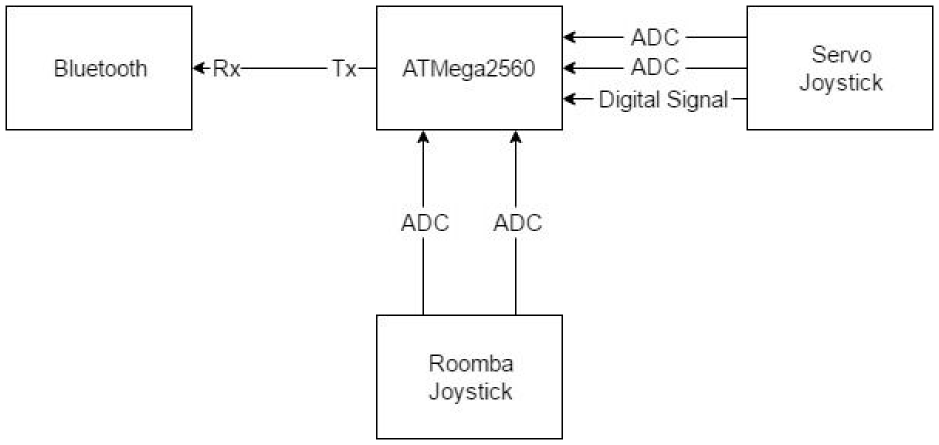


Figure 1. Shows the system architecture of the base station.

As you can see in Figure 1 above, the BlueTooth module was only used for sending information from the Roomba and servo joysticks to the base station.

3.1.2. Tasks

Five tasks were created to collect, manage, and send the data from the base station to the remote station. There were a few hiccups in managing all the tasks as 4 of them were periodic, and one was a system startup task. The one system task was used for initialization purposes and was run only once.

3.1.2.1. BlueTooth Send Data

The BlueTooth Send Data task was used to create a packet that is then sent over serial BlueTooth to the remote station. As we had no access to Arduino libraries, it took a lot of lines to parse through and create a properly formed packet.

Parameter	**Time (10ms TICKs)**
Period	7
Worst Case Runtime	2
Offset	200

Table 1. Shows the period, worst case runtime, and offset of the BlueTooth Send Data task.

Ideally, the period for the periodic tasks would have been shorter, but based on the constraints to factors of 14 (explained in section 3.1.2.4), 7 was used. Worst case runtime of 2 was used because space was available in the scheduling. A 200 TICK (2 seconds) offset was used to ensure that the initialization task had plenty of time to run.

3.1.2.2. Check Joysticks

The purpose of this task was simply to read in data from the ADC for all the joystick inputs. Yet again, because we had no Arduino library access, understanding and configuring the ADC was difficult and confusing.

Parameter	**Time (10ms TICKs)**
Period	7
Worst Case Runtime	2
Offset	202

Table 2. Shows the period, worst case runtime, and offset of the Check Joysticks task.

Ideally, the period for the periodic tasks would have been shorter, but based on the constraints to factors of 14 (explained in section 3.1.2.4), 7 was used. Worst case runtime of 2 was used because space was available in the scheduling. A 202 TICK (~2 seconds) offset was used to ensure that the initialization task had plenty of time to run and that it was offset from the BlueTooth Send Data task.

3.1.2.3. Get Speeds

The task Get Speeds was to take all the input information from both joysticks and adjust the values to match what is required by the remote station. The point of this was to attempt to keep as much computation on the base station side and to just have variable utilization done on the remote station side.

Parameter	**Time (10ms TICKs)**
Period	7
Worst Case Runtime	2
Offset	204

Table X. Shows the period, worst case runtime, and offset of the Get Speeds task.

Ideally, the period for the periodic tasks would have been shorter, but based on the constraints to factors of 14 (explained in section 3.1.2.4), 7 was used. Worst case runtime of 2 was used because space was available in the scheduling. A 204 TICK (~2 seconds) offset was used to ensure that the initialization task had plenty of time to run and that it was offset from the Check Joysticks task.

3.1.2.4. Right Joystick Switch

The purpose of this task was to read in the button press from the right joystick and to trigger a switch in the laser's state (on/off). This task was very similar to what we used in project 1, but due to the RTOS that was used in this project, we were limited to 10ms TICKs for our timing. In project 1, we determined that the ideal delay time for button press was around 135 ms.

Parameter	**Time (10ms TICKs)**
Period	14
Worst Case Runtime	1
Offset	206

Table 3. Shows the period, worst case runtime, and offset of the Right Joystick Switch task.

The period of 14 was chosen because we had to decide between 130 ms or 140 ms as we couldn't get the ideal 135ms. A period of 13 seconds is difficult to work with as it is a prime number and we want the other tasks to run much faster, the next best option was using 14 that is only divisible by 7 and 2. The reason 7 was used for the other periodic tasks was because it was the only usable divisor of 14 to make scheduling simpler to deal with. Worst case runtime of 1 was used because it is the smallest possible that we could use and it is a very simple task. A

206 TICK (~2 seconds) offset was used to ensure that the initialization task had plenty of time to run and that it was offset from the getSpeeds task.

3.1.3. Prioritization

The tasks were prioritized to have fairly equal weights, except priority was given for the initialization task.

Task	Priority
Initialization	System
BlueTooth Send Data	Periodic
Check Joysticks	Periodic
Get Speeds	Periodic
Right Joystick Switch	Periodic

Table 4. Shows all of the tasks for the phase 1 base station and their priorities.

Initialization contained all the required functions to start up the ADC, UART, onboard LED, the joystick digital switch, and getting the centers of the joysticks for movement calibration. It had to run first because nothing would function correctly without it. Each of the other tasks was equally as important and all given the periodic task designation. Unfortunately, due to user input, there were some timing issues, but data was still read and transmitted quickly enough.

3.2. Remote Station

The remote station was in charge of receiving information from the base station and then updating the states of the Roomba, laser, and servos accordingly. The following sections will go into detail about the remote stations including its overview, the tasks it was running, and the prioritization of the tasks.

3.2.1. System Overview

The remote station consisted of five components which included the ATMega2560, Roomba, Bluetooth module, laser, and servos. The remote station was responsible for receiving data from the Bluetooth module, and then updating the Roomba, laser, and servos based on the data that it received. The overall architecture of the remote station can be seen below in Figure 2.

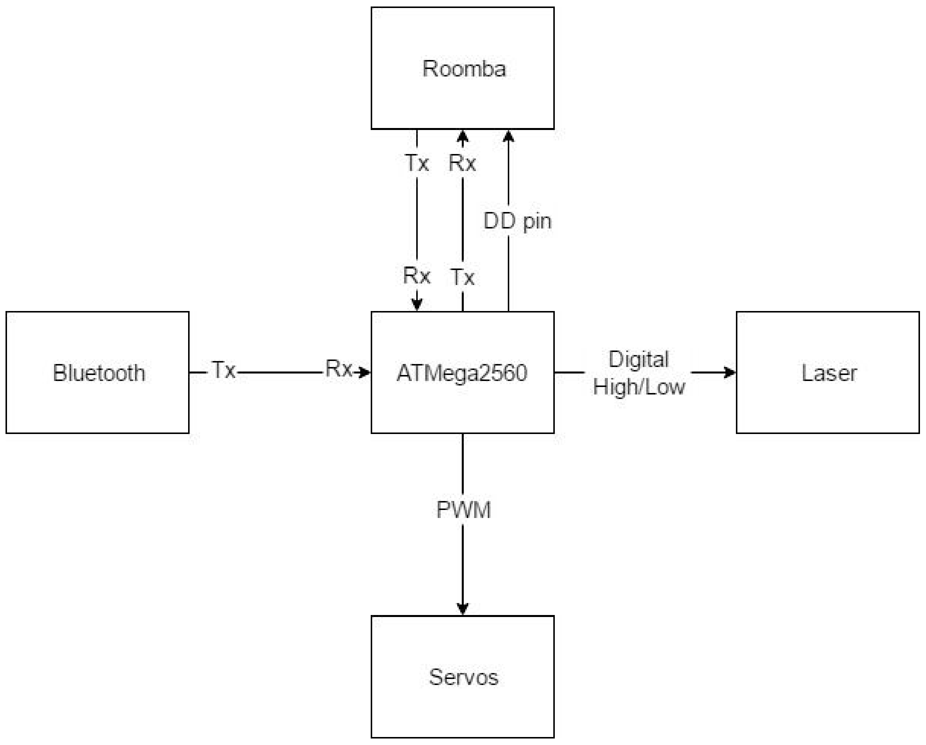


Figure 2. Shows the system architecture of the remote station.

As you can see in Figure 2 above, the BlueTooth module was only used for receiving commands which it then used to send either digital high or low signal to the laser, PWM signals to the servos, or serial data to the Roomba.

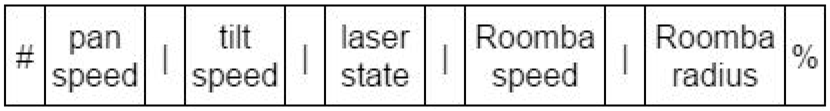
3.2.2. Tasks

Since the remote station had a very specific purpose in phase 1, getting commands and fulfilling them, there were only two tasks needed. The two tasks that the remote station used were Receive Transmission and Update; these tasks will be explained in detail in the following sections.

3.2.2.1. Receive Transmission

The Receive Transmission task was responsible for reading packets that were sent from the base station over BlueTooth and then extracting the information from them.

The structure of the packets that were being received is in the same format as those used in project 1, as shown in Figure 3 below.



#	pan speed	|	tilt speed	|	laser state	|	Roomba speed	|	Roomba radius	%

Figure 3. Structure of the packet that was used to send information from the base station to the remote station.

As you can see in Figure 3 above, the control data for the pan and tilt servo, the speed and radius of the Roomba and the laser were framed between a # and % character, and then each piece of data was separated by a | character. This allowed the remote station to receive all of the information about the states of the Roomba, servos, and laser in a single run of the Receive Transmission.

The Receive Transmission will be a periodic task as it needed to perform its job at a certain rate and does not require a System level priority. The periodic information for the Receive Transmission task is shown below in Table 5.

Parameter	Time (10ms TICKs)
Period	4
Worst Case Runtime	1
Offset	500

Table 5. Shows the period, worst case runtime, and offset of the Receive Transmission task.

As you can see in Table 5 above, the period is 4 TICKs, the worst-case runtime is 1 TICK, and the offset is 500 TICKs. The period was chosen to be 4 TICKs because data was being sent from the base station at a rate of 7 TICKs and it needed to be consumed faster than it was being sent to ensure no packets were missed. The offset was 500 TICKs because the setup sequence for the remote station which involved initializing everything and setting the baud rate on the Roomba should take a little less than 4.5 seconds.

3.2.2.2. Update

The Update task was responsible for updating and adjusting the servos, powering on/off the laser, and sending drive commands to the Roomba, based off of information that was gathered by the Receive Transmission which was previously explained. The Update task will be a periodic task as it should run periodically at the same rate as the Receive Transmission task so that once a transmission is received the Update task runs right after to apply the data from the transmission. The periodic creation details of the Update task are shown below in Table 6.

Parameter	Time (10ms TICKs)
Period	4
Worst Case Runtime	1
Offset	502

Table 6. Shows the period, worst case runtime, and offset of the Update task.

As you can see in Table 6 above, the period is 4 TICKs, the worst-case runtime is 1 TICK, and the offset is 503 TICKs. This specific period, worst case runtime, and offset was chosen so that the Update task will always run one time right after the Receive Transmission task has run. This ensured that we were only updating the servos, laser, and Roomba with fresh data.

3.2.3. Prioritization

At during the startup of the remote station three tasks will be created, one for setting up and initializing all of the necessary hardware and software, and the two periodic tasks which we described in Sections 3.2.2.1 and 3.2.2.2. These three tasks and their priorities can be seen below in Table 7.

Task	Priority
Setup	System
Receive Transmission	Periodic
Update	Periodic

Table 7. Shows all of the tasks for the phase 1 remote station and their priorities.

The Setup task which is only run once on startup is a System level task because it needs to finish before the offset of the Receive Transmission and Update tasks run out, therefore it cannot and should not be preempted by anything and should run one time to completion. The Receive Transmission and Update were both periodic tasks because we needed to be able to control both the rate at which they ran, as well as the order in which they ran in. With them being periodic we were able to interleave their runtimes in a way that ensured that Receive Transmission ran once to gather incoming data, and then Update ran once to apply this data, and if anything were to go wrong the RTOS would enter an error state.

3.3. Problems

The following sections outline the problems that we ran into when working on Phase 2 of the project.

3.3.1. Controlling Servos Through PWM

When creating code to control the servos we ran into issues on two fronts. The first is that the provided RTOS was using timers that were required for us to use the PWM pins that we needed to control the servos; this required us to drive into the RTOS code to try and change their use of the timer to another timer without breaking anything in the system. The second issue was with trying to figure out the mapping between integer values on the target register and the actual microsecond output pulse of the PWM pin. This touch quite a while to figure out, as simple differences in the initialization of the timer interrupt would drastically change how the integer values in the target register we mapped.

3.3.2. Not Realizing We Needed to Create Our Own UART Buffer in Software

One major issue we ran into which lost us around 5 hours of work was that we assumed there was a circular hardware buffer on the UART receive ports which would handle incoming data. This issue didn't come up until we started trying to send packets over BlueTooth and they were coming in broken and misconfigured. In the end we figured out that we needed to create a software based circular buffer for receiving packets over BlueTooth which was triggered by an interrupt on the Rx pin.

3.3.3. Analog to Digital Conversions for Joysticks

The problem arose because we had no access to the built-in Arduino libraries that allowed for easy sensor reading. Having no access to the analogRead() function meant that we had to do the data collection in a bit more difficult manner. During the initialization of the ADC, the prescaler was set to 128 having a 125KHz sample rate at the 16MHz clock speed. In the ADC input buffer, there are 10 bits used, but in 2-byte chunks: 2 from one and 8 from the other. The 2 bits are in the ADCH (high) and the 8 are by default in the ADCL (low), and in order to simplify information reading, the data got shifted so there were 8 in the high instead. Because we were reading only 8 of the 10 bits, we got values from 0-256 instead of 0-1024. When the data from the ADCH buffer is read, it invalidates the information and the ADC timer is reset. After all of this was done, it was just a matter of using the values from 0-256 that we were given.

3.3.4. Handling Corrupt or Incomplete Packets

Unfortunately, despite proper planning for BlueTooth packet design, there were malformed or incomplete packets during transmission. This caused some loops during the packet reading process on the remote station to lock up or fail. It took a significant amount of time to diagnose, but once the issue was identified, it was fixed by re-working the BlueTooth data receiving loop and allowing it to break out if a packet contained malformed data or if a new packet starts before a current packet had given its closing state.

4. Phase 2 System Design

Phase 2 of project 3 was to introduce semi-autonomous behavior to the robot. The goal of this phase is to have the remote station takeover control of the robot in the event that the Roomba detected a collision with either a physical wall or a virtual wall. The following sections will explain how this was done.

4.1. Subsumption Architecture

To achieve the semi-autonomous control for Phase 2 we used subsumption architecture, which allowed our system to behave differently based on external sensors. This was done by having two two separate FSMs which with their combined state governed if the Roomba motors were being controlled manually or if the remote station would be taking over temporarily. This subsumption architecture can be seen below in Figure 4.

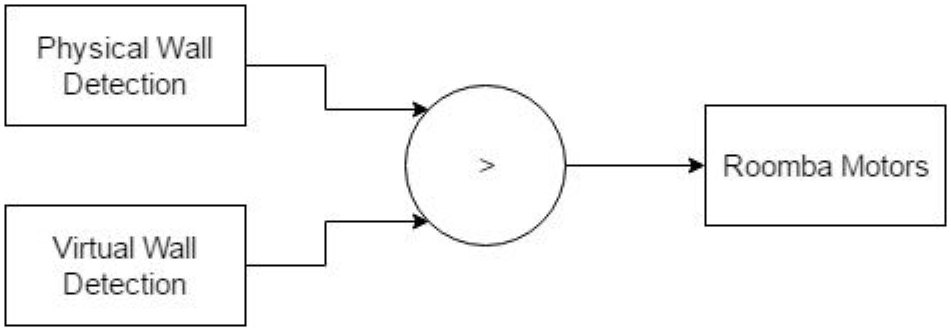


Figure 4. Shows the subsumption architecture of our phase 2 system.

As you can see in Figure 4, there are two FSMs which combine to dictate the behavior of the motors. The FSMs for both the Physical Wall Detection and the Virtual Wall Detection were implemented in periodic tasks which will be explained in Section 4.2. The FSM diagram for each task is also available in Section 4.2.

4.2. Tasks

As mentioned in Section 4.1, there are two additional tasks for Phase 2 of this project. One is for checking if a collision with a physical wall has occurred and then handling it, and the other is for checking if a collision with a virtual wall has occurred and then handling it; these tasks will be explained in the following sections.

4.2.1. Check For and Handle Physical Wall Collision

The task for checking if a collision with a physical wall has occurred and then handling it will be a periodic task which implements the FSM shown below in Figure 5.

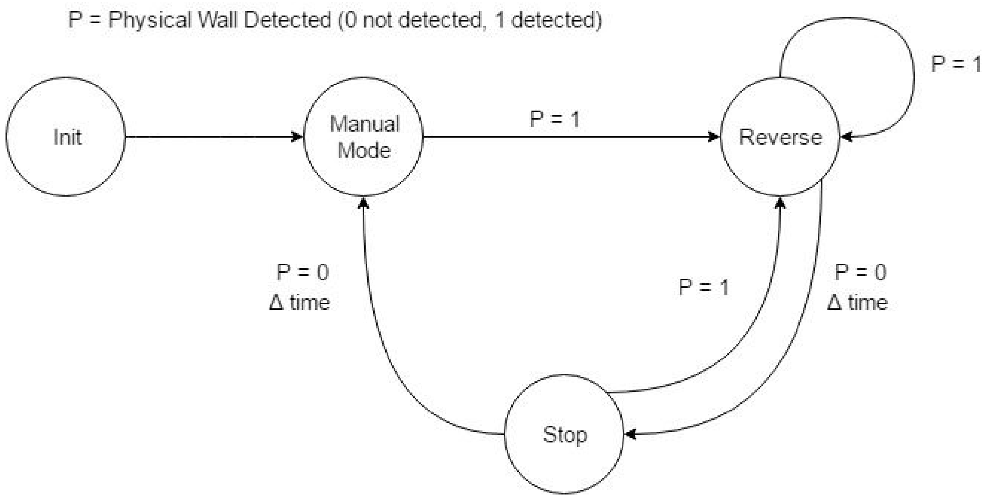


Figure 5. Shows the FMS of the physical wall collision detection and handling task.

As you can see in Figure 5 above, after initialization the state of the remote station will remain in manual mode where it will accept and respond to commands from the base station. In the event that a collision with a physical wall is detected the remote station will change its behavior by taking over control, reversing the robot, stopping the robot, and then finally returning control back to the base station.

This task will be a periodic task where each time it runs it will check to see if a collision with a wall has occurred and then take over briefly if one has. The periodic details for this task can be shown below in Table 8.

Parameter	Time (10ms TICKs)
Period	8
Worst Case Runtime	1
Offset	504

Table 8. Shows the period, worst case runtime, and offset of the Check For and Handle Physical Wall Collision task.

As you can see in Table 8 above, the period is 8 TICKs, the worst-case runtime is 1 TICK, and the offset is 504 TICKs. A period of 8 was chosen so that it would fit in with the rest of the tasks, which will be shown in Section 4.3, while also running at the fastest possible rate to ensure a fast response to physical wall collisions.

4.2.2. Check For and Handle Virtual Wall Collision

The task for checking if a collision with a virtual wall has occurred and then handling it will be a periodic task which implements the FSM shown below in Figure 6.

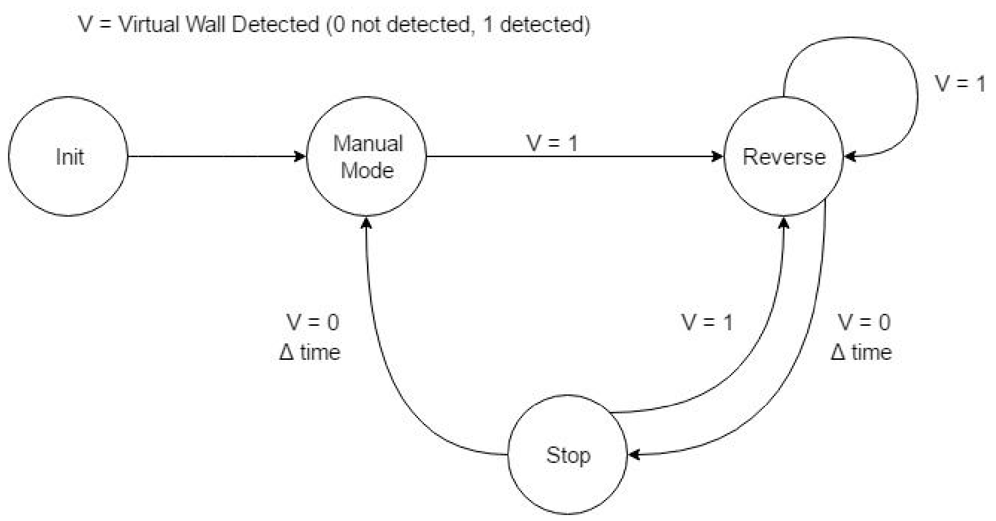


Figure 6. Shows the FMS of the virtual wall collision detection and handling task.

As you can see in Figure 6 above, after initialization the state of the remote station will remain in manual mode where it will accept and respond to commands from the base station. In the event that a collision with a virtual wall is detected the remote station will change its behavior by taking over control, reversing the robot, stopping the robot, and then finally returning control back to the base station.

This task will be a periodic task where each time it runs it will check to see if a collision with a wall has occurred and then take over briefly if one has. The periodic details for this task can be shown below in Table 9.

Parameter	Time (10ms TICKs)
Period	8
Worst Case Runtime	1
Offset	506

Table 9. Shows the period, worst case runtime, and offset of the Check For and Handle Virtual Wall Collision task.

As you can see in Table 9 above, the period is 8 TICKs, the worst-case runtime is 1 TICK, and the offset is 506 TICKs. A period of 8 was chosen so that it would fit in with the rest of the tasks, which will be shown in Section 4.3, while also running at the fastest possible rate to ensure a fast response to physical wall collisions.

4.3. Prioritization of Remote Station Tasks

With the addition of the two new periodic tasks in Phase 2 for detecting and handling collision with physical or virtual walls, there is a total of four periodically running tasks in the system. The periods, worst case runtime, and offset of each task are shown below in Table 10.

Task	Period (10ms TICKs)	Worst Case Runtime (10ms TICKs)	Offset (10ms TICKs)
Receive Transmission	8	1	500
Update	8	1	502
Detect Physical Wall	8	1	504
Detect Virtual Wall	8	1	506

Table 10. Shows all of the concurrent tasks for the phase 2 remote station and their priorities.

Since there will be a total of four periodic tasks which all require at most 1 TICK to run while also running as fast as possible the period and offset shown above in Table 10 must be used. This will ensure that all of the tasks get an equal opportunity to run. Also, as previously mentioned in Section 3.2.3, the offset of the tasks starts at 500 TICKs because the initialization of all of the hardware and software one startup requires around 4.5 seconds to complete, so this offset of 5 seconds ensures that we will not enter an error state in the RTOS on boot unless something went wrong with the setup.

4.4. Problems

The following sections outline the problems that we ran into when working on Phase 2 of the project.

4.4.1. Incorrect Data Being Received From the Roomba

The main issue that was occurring was that when we were requesting front bumper and virtual wall collision sensor data from the Roomba, we would receive a physical wall detection for both physical and virtual walls. We were not able to diagnose the issue as we had no idea why the Roomba was failing to give us correct values when we prompted it for specific data.

4.4.2. Hidden Wall Detection FSM

In addition to the data acquisition problems as described in section 4.4.1, despite having drawn out and organized functions to handle data input from the Roomba, there was an FSM that was being created that we had no control over. The Roomba front bumper seemed to work no problem, but the virtual wall detection had no input. It wasn’t until both the virtual wall was in place, and the bumper was hit would the system “switch states” in which the front bumper failed to continue functioning properly and only the virtual wall detection worked. Unfortunately, this issue was not completely diagnosed and likely could have been remedied by the creation of a system task associated with Roomba data causing an ISR to trigger.

5. Testing

Phase 1 and 2 were tested initially by data analysis on our inputs and then by physical verification of the movements/actions of the Roomba. The following are some examples of tests that we worked on.

5.1. Manual Control Over Roomba

Objective:

The manual control over the Roomba was tested to ensure that we have accurate measurements coming from both of the controls on the base station and that they were being properly translated to output on the Roomba. It took tweaking to ensure that the controls were nicely usable for the user.

Description:

We initially tested the manual controls be ensuring that the data being transferred in the packets was as expected and they were being read out by the remote station correctly. Once the data was being successfully transferred to Roomba control, it was just a matter of getting in some “wheel time” practicing controlling the Roomba and making tweaks to the data being sent from the joysticks to fit the preferences of the user.

5.2. Manual Control Over Servos

Objective:

This was tested to optimize the aiming capabilities of the laser mounted to the Roomba. This was very user dependent like in section 5.1.

Description:

We tested this initially by observing a min and max values were being received from the base station to the remote station. The pan and tilt speeds were then tuned to match user preference based on their max possible speed.

5.3. Manual Control Over Laser

Objective:

The button press on one of the joysticks on the base station turned the laser on and off, the timing on this was critical and took some adjustment. Even though we had received good values from the Project 1 at 135ms, we had to test it again as the RTOS that we were using only supported increments of 10ms.

Description:

This was tested by repeatedly pressing the button and initially observing the change in packets on the base station, then by the laser being turned on and off on the Roomba itself.

5.4. Physical Wall Detection

Objective:

The objective of testing this was to ensure that when the front bumper was pressed in any location, the Roomba would react and act according to the automation programmed with this event.

Description:

Initially, the testing for this was simple as we had to just press the bumper and receive values. Unfortunately, we realized quickly that the sensor had a long activation time and would trigger the FSM multiple times on a single press. Future testing was done by simply running the Roomba into either feet or walls to ensure that everything was working as intended.

5.5. Virtual Wall Detection

Objective:

This, like the Physical Wall Detection testing, was to ensure that the Roomba would act accordingly to outside source inputs and enter its designated autonomous state.

Description:

We tested this by driving the Roomba back and forth to enter and leave the beam created by the virtual wall. This was able to give us an idea of under what conditions the sensor would be triggered and how we could account for multiple sensor triggers.

6. Appendix

6.1. Code

Our code for project 3 is available on GitHub, and contains the correct files so that it can be directly imported into Atmel Studio 7 or used directly on Linux.

- https://github.com/JoshuaPortelance/CSC460-Spring2017/tree/master/project_3

6.2. Hardware

6.2.1. ATMega2560

- https://www.arduino.cc/en/Main/ArduinoBoardMega2560

6.2.2. Bluetooth Radio

6.2.2.1. HC-05

- http://www.electronicaestudio.com/docs/istd016A.pdf

6.2.2.2. HC-06

- https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf

6.2.3. Analog Joysticks

- https://tkkrlab.nl/wiki/Arduino_KY-023_XY-axis_joystick_module

6.2.4. Servo Motors

- https://www.heinpragt.com/techniek/robotica/images/SG90Servo.pdf

6.2.5. Laser

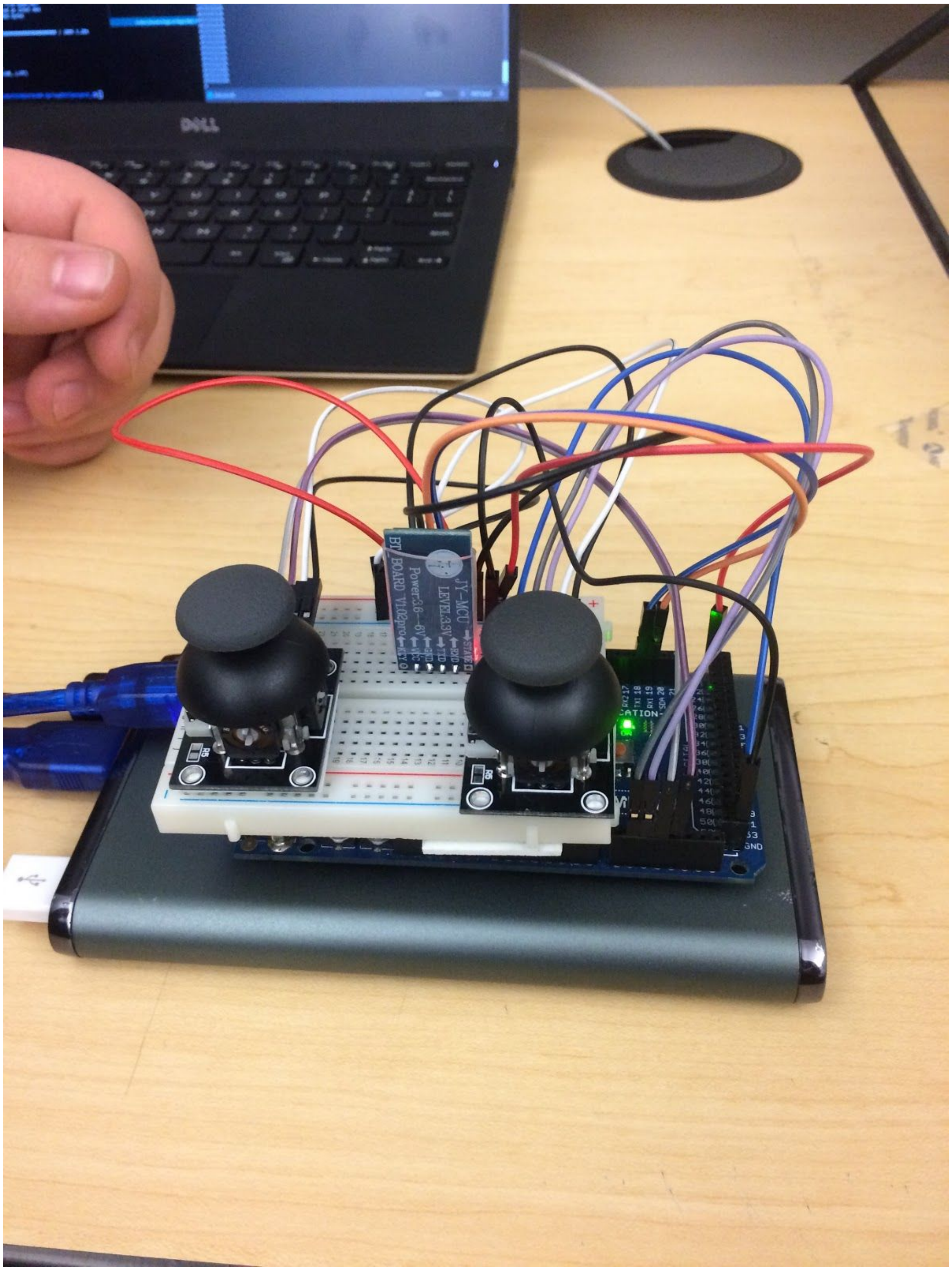
- http://henrysbench.capnfatz.com/henrys-bench/arduino-output-devices/ky-008-arduino-laser-module-guide-and-tutorial/

6.2.6. iRobot Roomba Create 2

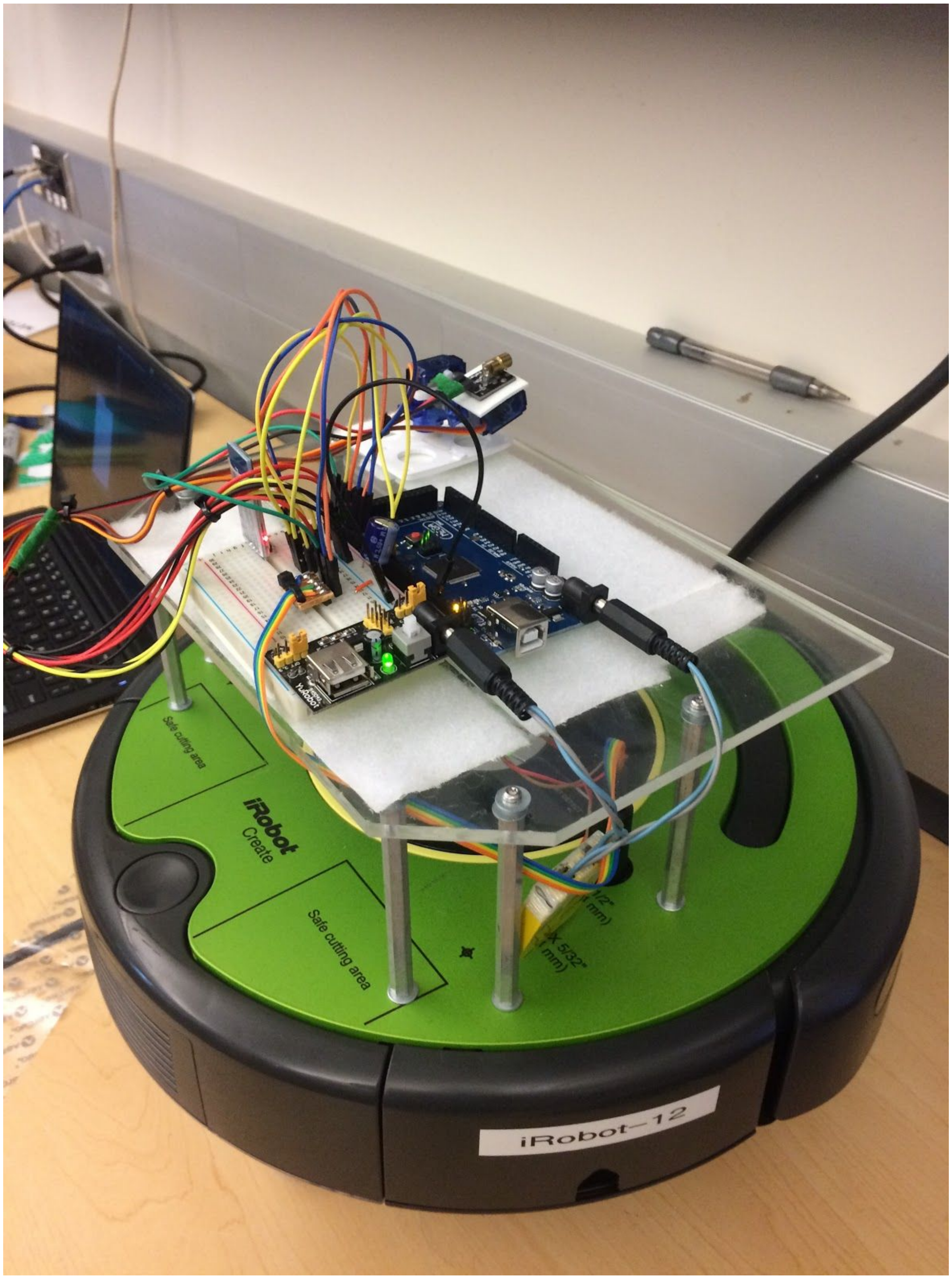
- http://www.irobot.com/~/media/MainSite/PDFs/About/STEM/Create/create_2_Open_Interface_Spec.pdf

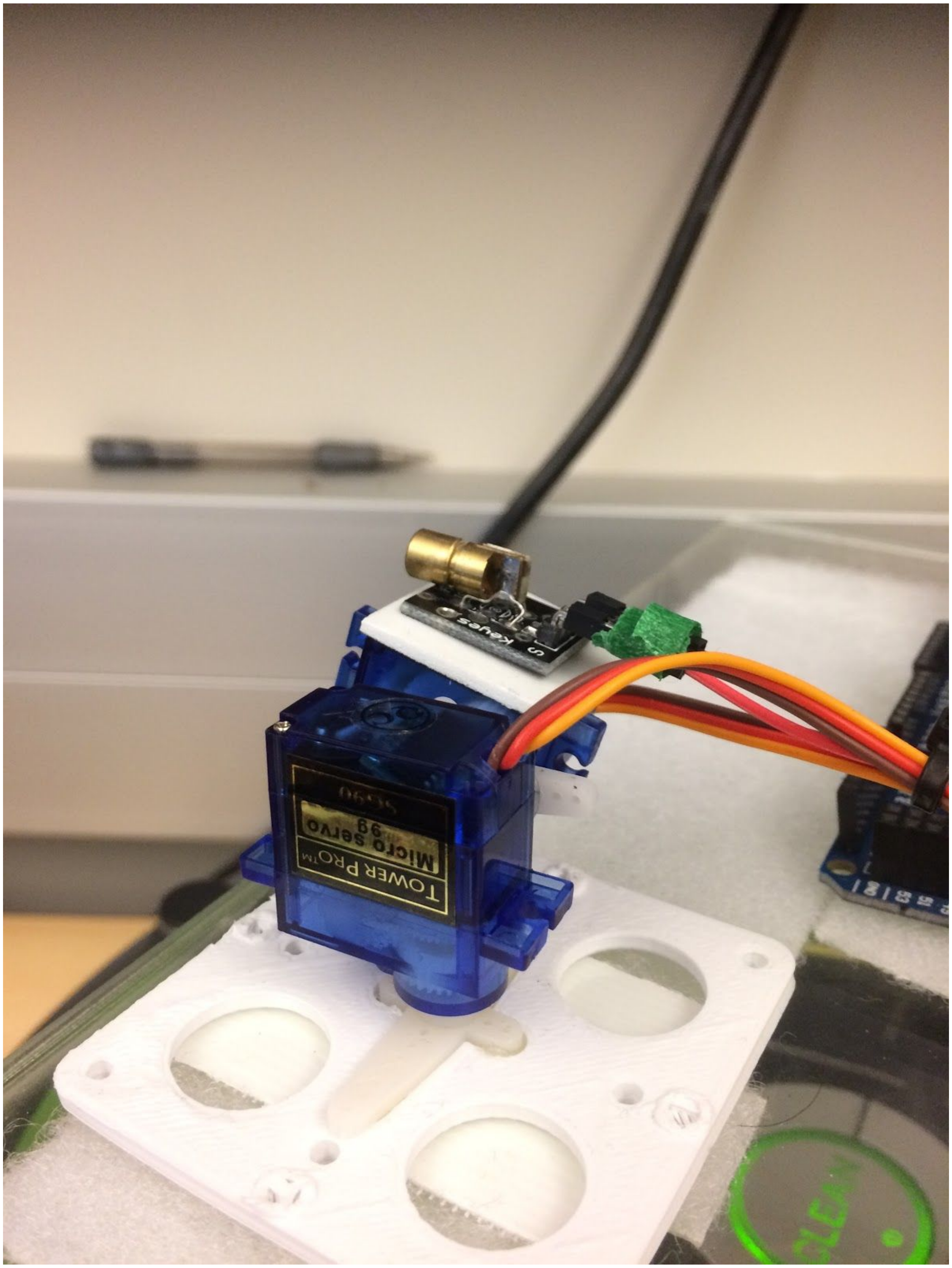
6.3. Pictures

6.3.1. Base Station



6.3.2. Remote Station




TOWER PRO™
Micro servo
9g
SG90
keyes
S
CLEAN