CSC 460: Design and Analysis of Real Time Systems

Project 3, Team 7

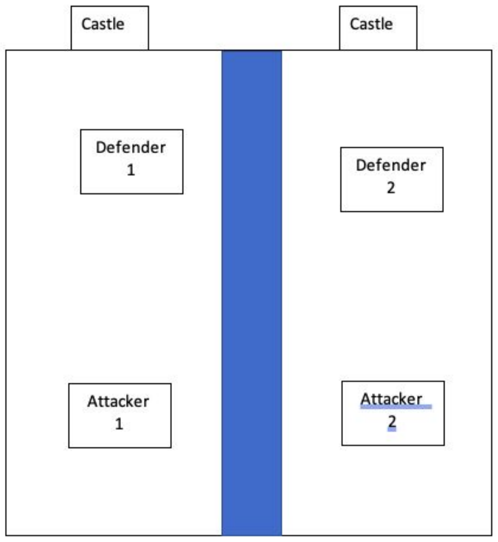
Andrew Meijer & Kenzie Wong

Submission: April 7th, 2019

Abstract

The goal of project 3 is to use what we learned from project 1 and project 2 to compete in a tournament against other students using a roomba. The goal of the game is to design a roomba to operate within the rules and to compete against other teams. To begin, we will describe the rules and outline of the tournament.

The game entitled "Defend Your Castle" involves two teams of two roombas each. Each roomba is equipped with a laser, shield, and light sensor to detect laser fire. The lasers are mounted onto pan-and-tilt servo motors so that when controlled, the roombas can aim and shoot laser beams. The teams play on a square battlefield that is divided vertically by a river, represented by a beam of infrared light. On each side of the river (IR-beam), there is a castle equipped with a light-sensor to detect laser fire. The goal of the attacking team is to shoot the castles using the laser on their roomba while the defending team protects the castles. Points are accumulated by the number of castles defeated/defended and roombas killed. Going out of bounds, getting hit by a laser, or crossing the IR-beam counts as a killed roomba. The roombas are controlled by a remote that communicate via bluetooth or wifi. Even though the roombas are controlled via remote, they must have some autonomous behaviour to respond to collisions and avoid the IR-beam.



Furthermore, each player must operate in two alternating modes. The first mode gives the player full control over the roomba and the second mode disables forward and backward movement such that the roomba can only turn in place. The laser and servo motor are operational in both modes, but, the laser can only shoot for a total of 10 seconds. If a light-sensor on any roomba or castle in the game is activated by a laser for 2 seconds continuously, it is considered to be defeated/killed.

The primary learning objective (workload) of this project is to make a real-time operating system for the roomba that handles commands from the remote, autonomous behavior, and input from the roomba's sensors. Using our work from project 1 and 2, we use a Time Triggered Architecture (TTA) scheduler that allows for interrupts, dynamic tasks, and task states to make to roomba behave accordingly.

Table of Contents

Materials List

- IRobot Create 2 (Roomba)
- Arduino ATmega2560 microcontroller (x2)
- UART cable for roomba arduino
- Bluetooth Radio (x2)
- Pan and Tilt servo
- Analog Joystick with push button (x2)
- Light Sensor
    - Red solo cup
- Laser
- LED
- Various wires

IRobot Create 2
We refer to the IRobot Create 2 as the roomba for simplicity. It is one of the core components of this project along with the ATmega2560 microcontroller. The roomba's specifications can be found on IRobot's website [2], but Open Interface Specifications for the roomba which detail how the roomba can receive commands and transmit data are detailed in a separate manual [3]. The roombas for this project have an elevated platform for us to mount all of our equipment without blocking the buttons and sensors that are on its surface.

Arduino
The Arduino ATmega2560 is one of the core components in this project because it is where we upload our code. One is used for the roomba, and the other for the remote.

Microcontroller	**ATmega2560**
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA

Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Figure 1: ATmega2560

More information on the Arduino MEGA 2560 can be found at the Arduino website referenced below [1].

UART cable

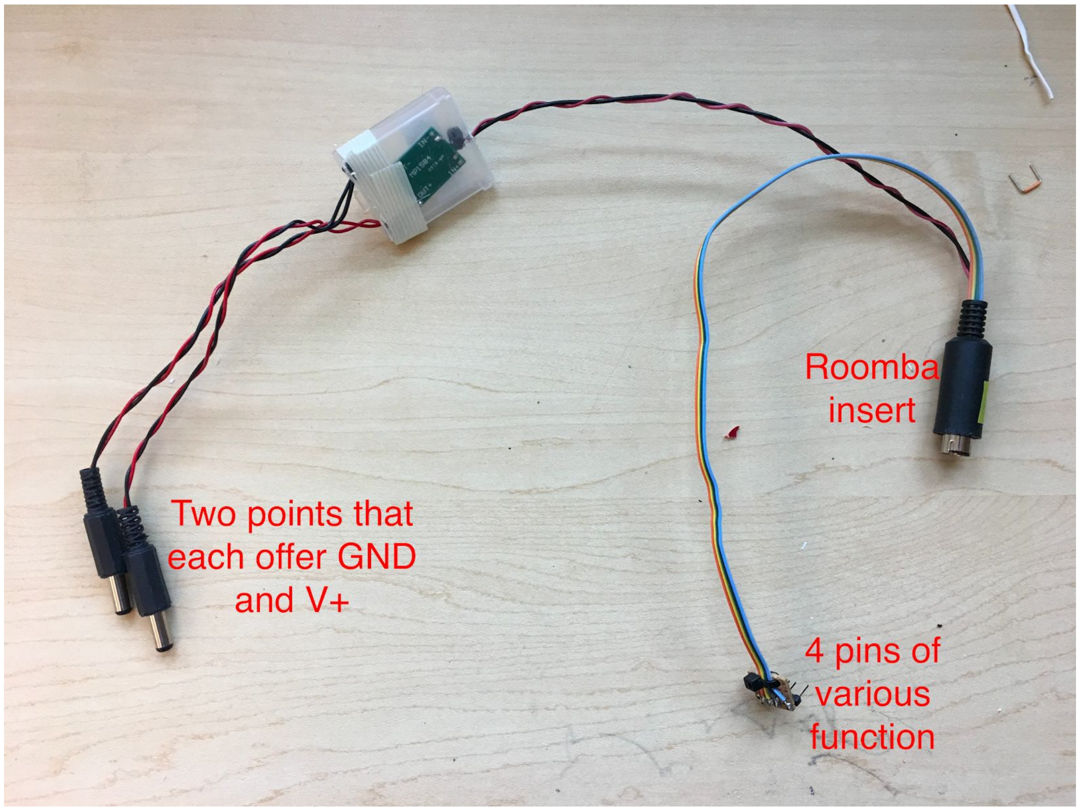

Roomba insert
Two points that each offer GND and V+
4 pins of various function

Figure 2: UART Cable

The UART cable joins the roomba to all of the other components. As seen in the image above, "Roomba insert" is the mini-DIN plug that connects to the roomba. On the left of the image, there are two power plugs that can supply the ATmega2560 or other devices with 5 volts. The 4 pins on the bottom right of the image are used for their own functions as shown in the following table:

Wire	Name	Function
Blue	Device Detect (DD pin)	Connecting the Roomba and turning it on.
Green	UART Receive pin	The roomba reads data sent along this pin.
Yellow	UART Transmit pin	The roomba can write data along this pin.
Orange	Ground (GND)	The Roombas ground terminal.

Figure 3: Roomba Pin Functions

Bluetooth
This project requires two bluetooth components, one for the remote side and one for the roomba side. Specifications for the JY-MCU bluetooth chips we use can be found in this user guide [4].

Pan and Tilt servos
The servo is mounted to the roomba and used to move the laser around. The Pan and Tilt both use an SG-90 motor and its specifications can be found on this datasheet [5].

Analog Joystick with push button
This project requires one joystick to control roomba movement and one to control the pan and tilt movement. The joystick specifications are referenced at the end of this report [6].

Light Sensor
The light sensor for this project sends data to an analog pin on the ATmega2560, but we found in our testing that it is quite sensitive to ambient light. Covering the sensor with a red solo cup (or similar translucent cup) is necessary to reduce sensitivity to ambient light. This also gives an appropriately sized target for enemy lasers. When lasers hit the cup, the light is dispersed enough to affect the sensor inside. Figure 4 shows how we set-up the light sensor . Once the cup is placed on top, we secure it with tape to further reduce the amount of ambient light that can get in.

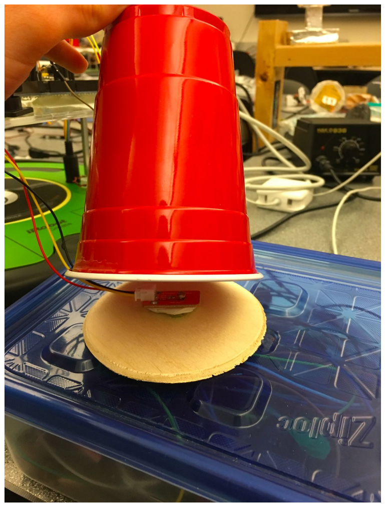


Figure 4: Light Sensor

Laser
We use a KY-008 laser on our pan and tilt servo. The light must be bright enough so that when pointed towards the light sensor, it changes the readings enough to differentiate between ambient light and the light from the laser. Some of the specifications for our laser are referenced at the end of this report [7].

LEDs
In this project we use two LED lights to give some visual feedback to the user about what mode the roomba is currently in. The ATmea2560 is equipped with on-board LEDs, but external LEDs are much easier to see. One is used on the roomba, the other on the remote.

Wires
We use the wires to connect components together as needed.

Concepts

A large portion of the concepts for project 3 are carried over from the first two projects. Project 1 introduces interacting with hardware and communicating over bluetooth, while project 2 focuses on expanding the capabilities of our TTA scheduler by allowing for interrupts, dynamic tasks, and task states. The only new technology for project 3 is the IRobot Create 2 or roomba, and there are concepts relating to the roomba that we discuss in this section. There are also new concepts for this project that come from connecting everything together in a way that is robust and reliable for tournament competition. Some driving questions to consider for this project are as follows:

- How does gameplay strategy affect our implementation of roomba control?
- What are the different useful signals we can send to and receive from the roomba?
- How do we need to customize our Time Triggered Scheduler for this project?
- How should we format data that we send across the bluetooth UART?
- How should we secure the hardware to the roomba and controller respectively?

To reliably control our roomba for Defend Your Castle, we use an intentionally simple design. We have movement controls for the roomba using a single joystick; the roomba can either drive or turn, but not at the same time. Instead of having gradual speed changes, our controller has only two speeds called LOW and HIGH that we can switch between. Our second joystick controls the pan and tilt of the servo motor. Buttons on the joysticks operate the laser and the speed-boost respectively.

This project involves programming the arduino to interact with the roomba by receiving sensor data and sending command data. The Device Detect pin requires a time-sensitive sequence of data in order to power-on the roomba. Fortunately, there are libraries available on connex that provide the code for this. To communicate with the roomba after it is turned on, we refer to the specification manual. On page 36, the manual explains the format of each different sensor packet. It has a list of the packets that can be received from the roomba by sending a command. We send and receive commands through UART, just like with the bluetooth chips.

Our scheduler for project 2 allows for interrupts, dynamic tasks, and task states, but for this project, we do not use dynamic tasks. On the roomba, we have periodic tasks for the bluetooth communication with the controller, the pan and tilt servo, the light sensor, the roomba sensors, and the mode switching between full-control and stationary mode. The sensor tasks can generate interrupts if there is a collision, but otherwise our tasks are all periodic.

In our bluetooth connection, to ensure that we do not miss any data from the controller, our periodic input task on the roomba station runs 4 times as fast as the periodic output task on the controller station. This is because when the controller station writes to the bluetooth chip through UART, it sends 4 characters, one for the laser, one for the roomba movement, one for the servo pan, and one for the servo tilt. Originally, our plan was to use 1 character to control all of the peripherals and the driving, but that approach requires excessive branching in the code.

The concepts for this project leave room for our own design and creativity. The most challenging aspect of understanding how the system works as a whole is the roomba interfacing because it is completely new. Though we have experience with bluetooth UART, sending our commands across bluetooth offers new challenges that we did not have in previous projects.

Procedure

Roomba

The first step of this project is to connect the arduino board to the roomba and understand how the roomba receives commands. The ordinary power button for the roomba starts a built-in cleaning program that out-prioritizes or overwrites our code from the ATmega2560. To bypass the power button, we access the roomba using a mini-DIN plug shown in the materials list of this report. The same cable also provides two power jacks that we use for the arduino and the breadboard power supply. The four additional wires from the cable allow us to connect the Device Detect pin, the transmitting port, the receiving port, and the electrical ground.

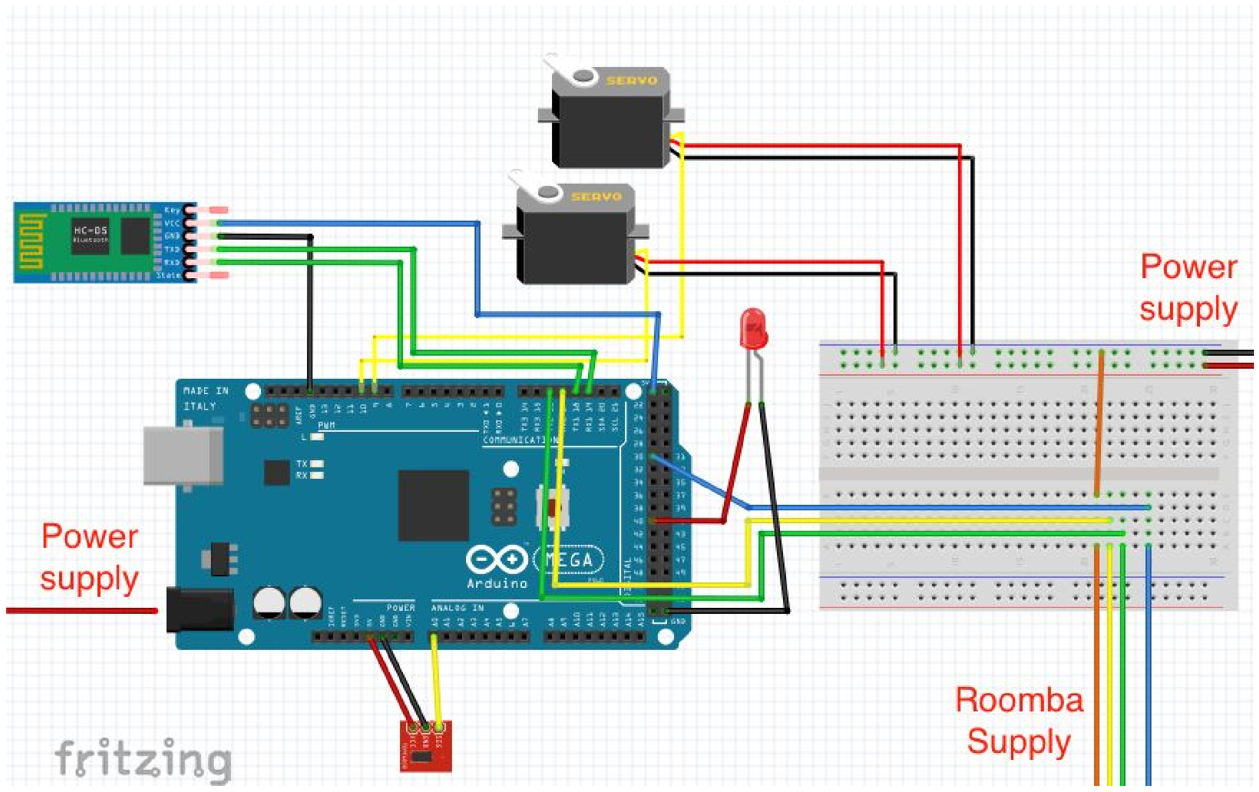


Figure 5: Roomba Wiring Diagram

Our first implementation of a control system for the roomba is much simpler than what is shown in our current wiring diagram. At the beginning of the project, our first step is to read the header file for the roomba library and understand the following two lines:

```
Roomba r(2, 30);
r.drive(100,0x8000);
```

The first line initializes a roomba object on the arduino that reads on UART channel 2, and has a Device Detect connection on pin 30. Reading the header files is a necessary step in understanding how to plug in the wires that come from the roomba. The roomba drive function takes two parameters. The first is the speed, and the second is the direction. Again, by reading the header files for the included library, we determined that a value of "0x8000" drives the roomba in a straight line. Putting a negative integer as the first argument drives the roomba backwards. Once we understood the roomba library and uploaded code to drive forward and backward, it was time to use our TTA scheduler and get input from a user. We attached a joystick button to the roomba and made a periodic task that creates interrupts whenever the joystick button is pressed. In this initial experiment, we had the roomba move forwards for 1 second and then backwards for 1 second in an endless cycle until interrupted by the joystick, at which point the roomba turns off.

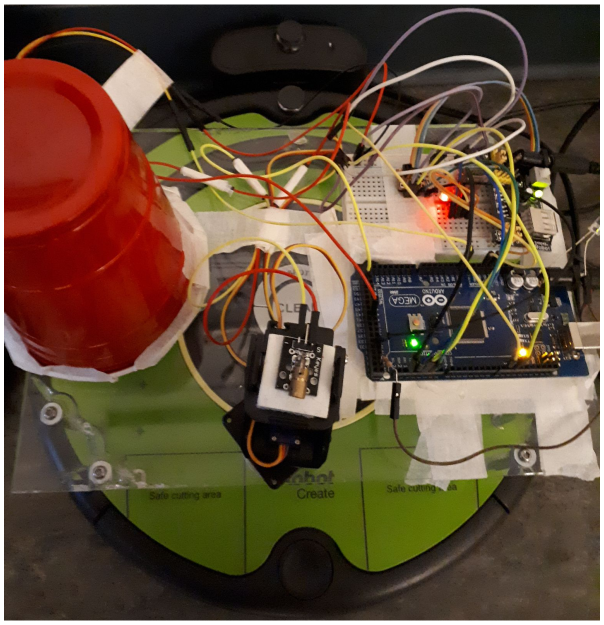


Figure 6: Roomba Station

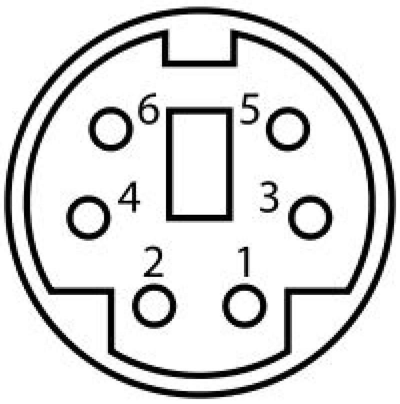
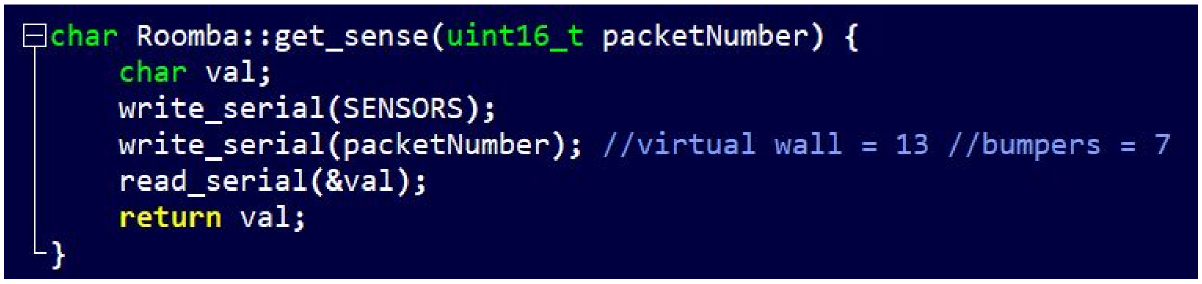


Figure 7: mini-DIN plug

At this point, we solved the bluetooth communication problem and returned to the roomba once we were able to send movement commands. Our next task was to read two sets of sensors on the roomba; the front bumper sensors, and the IR sensors. Along the way, we tackled various problems such as the light sensor, the servo motor, and the mode-switching functionality using periodic tasks. The light sensor is calibrated according to the frequency of its task, meaning that if it senses sufficient light for 100 tasks in a row, 2000ms will have passed because it is scheduled to run every 20ms. Compared to the light sensor, sensing from the roomba is complicated because it has a total of 26 sensor packets, such as the drop sensor. We are only interested in 2 sensor packets, one for the bumper and one for the infrared. Our library for this project includes a function to get data from the roomba, but it reads all of the packets available, and this costs too much time for our scheduler. To get only the 2 sensor packets we need, we wrote an additional library function called get_sense, shown in figure 8.



```
char Roomba::get_sense(uint16_t packetNumber) {
    char val;
    write_serial(SENSORS);
    write_serial(packetNumber); //virtual wall = 13 //bumpers = 7
    read_serial(&val);
    return val;
}
```

Figure 8: Reading Sensor Packets

The get_sense function takes a packet number as a parameter. To read specific packets from the roomba, we first must send to the roomba the "SENSORS" command so that it knows we are requesting sensor packets, and then we must send a number that corresponds to the packet we want. In this case, we want the virtual-wall (infrared) sensor and the bumper sensor. The specification manual for the roomba includes tables that shows the meaning of each bit in each packet [3]. The infrared sensor uses 1 bit for detection, but the bumper uses 2, one for each side. In our final project, we have a periodic task that reads this data and then creates interrupts that cause the roomba to automatically back away from the obstacle, whether it be a solid object or an IR-beam.

As we developped our project, we made various improvements to the speed at which we could deploy updates. For example, our light sensor requires a threshold in order to register a hit. Depending on the time of day, ambient sunlight from the windows in the lab will create different readings from the sensor. Instead of hard-coding a new threshold value that works with the ambient light, we include a calibration phase when our roomba powers-on. In the calibration, the sensor reads in ambient light over a 1 second period, takes an average, and then chooses a good threshold based on those values.

Remote
The goal of the remote side of the setup is to send user commands to the roomba over bluetooth. Expanding on project 1 we incorporated some additional features to ensure accuracy and reliability.

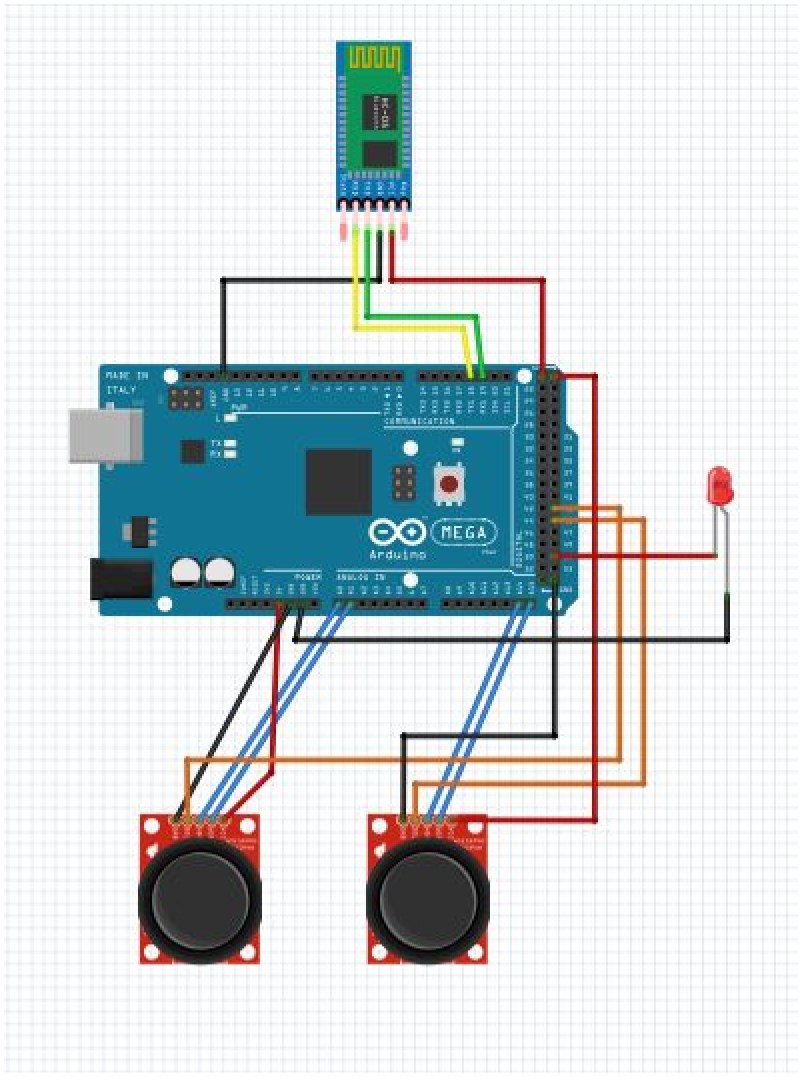


Figure 9: Controller Wiring Diagram

Here we have the setup of our remote. One joystick controls the commands for the roomba direction. We decided to have the button for this joystick be the on/off switch for the laser as the other joystick would control the pan and tilt. This way, we could move the pan and tilt servos and operate the laser at the same time. We connected an LED to one of the digital pins to provide some live feedback to the user. Finally, the bluetooth was connected to the TX and RX pins to send and receive commands.

The first question we needed answered was how we were going to send commands over bluetooth. Ideally we would send the full command onto the bluetooth connection then receive updates of servo positions, sensor status, and so forth. However, sending large amounts of data required processing on both ends and specific timing. The timing became a difficult issue to tackle as we could not manually sync the timing on both sides. Instead we would have to create some handshake that would match the sending and receiving on both ends. Instead of dealing with those issues, we decided to send code words that summarize specific commands. There were four types of commands we wanted to send; roomba movement, pan movement, tilt movement, and laser status. The roomba station has 5 possible commands: Forward, backwards, clockwise turn, counterclockwise turn, and no movement (still). Both the pan and the tilt each had three commands: left/down, right/up, and remain still. The laser had two states: laser on or laser off. After brainstorming we came up with the following table.

Command	Code
Roomba: Still	a
Roomba: Forward	b
Roomba: Backwards	c
Roomba: counterclockwise	d
Roomba: clockwise	e
Pan: Left	f
Pan: Right	g
Pan: Still	h
Tilt: Down	i
Tilt: up	j
Tilt: Still	k
Laser: On	X
Laser: Off	Y

Figure 10: Bluetooth Commands

This gave us a total of 13 commands to have. Each time the controller station arduino reads the user input, it sends them over to the roomba station. Receiving the command became easy as there was no header to read and the order did not matter. The bluetooth sends 4 commands every 100ms. For example, if the user does not move any of the joysticks, 'a', 'h', 'k', 'Y' is sent over. The arduino receiving these commands would be reading as fast or faster than 4 commands every 100ms. The roomba station continues the received command until it receives a new one that overrides it. It is on the controller side that we determine how long the laser can stay on for. As a laser command could only be sent every 100ms, we made the 'X' command only able to be sent 1000 times, limiting the laser to be on for 10 seconds as per the rules of the game. We then expanded the commands to include a "speed-boost" mode for the roomba. In this mode the roomba moves at a higher speed. This mode is activated by pushing the second joystick. To visually alert the user, the LED on the remote lights up. However, the light was incorporated due to the inconsistency of the joystick. After several rounds of testing we found the joystick select button was not reliably reading input. To make things as simple as possible, a capital letter of the command was sent to distinguish between normal speed and boosted speed. 'B', 'C', 'D', 'E' became fast modes of their lowercase versions.

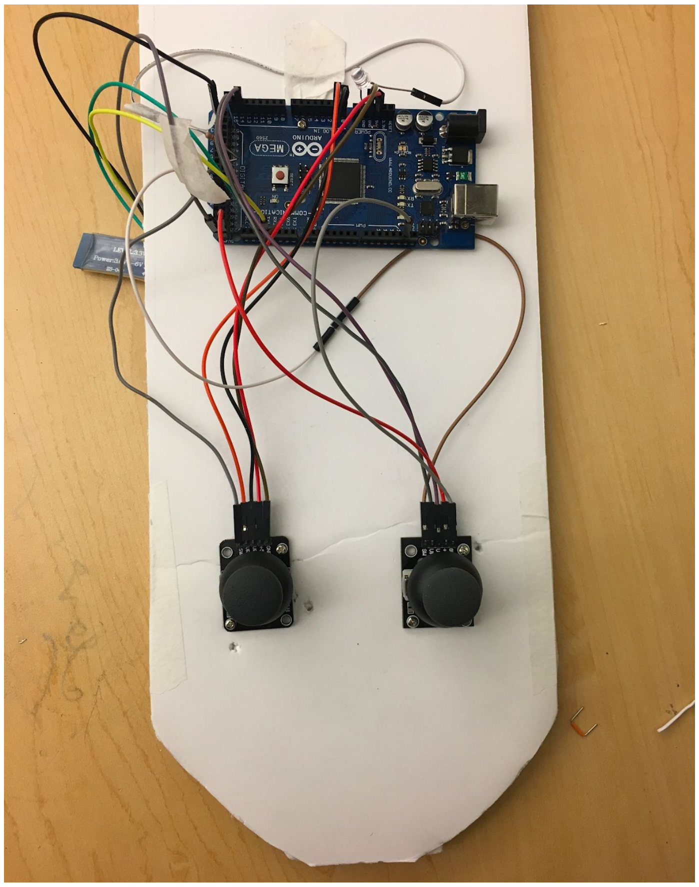
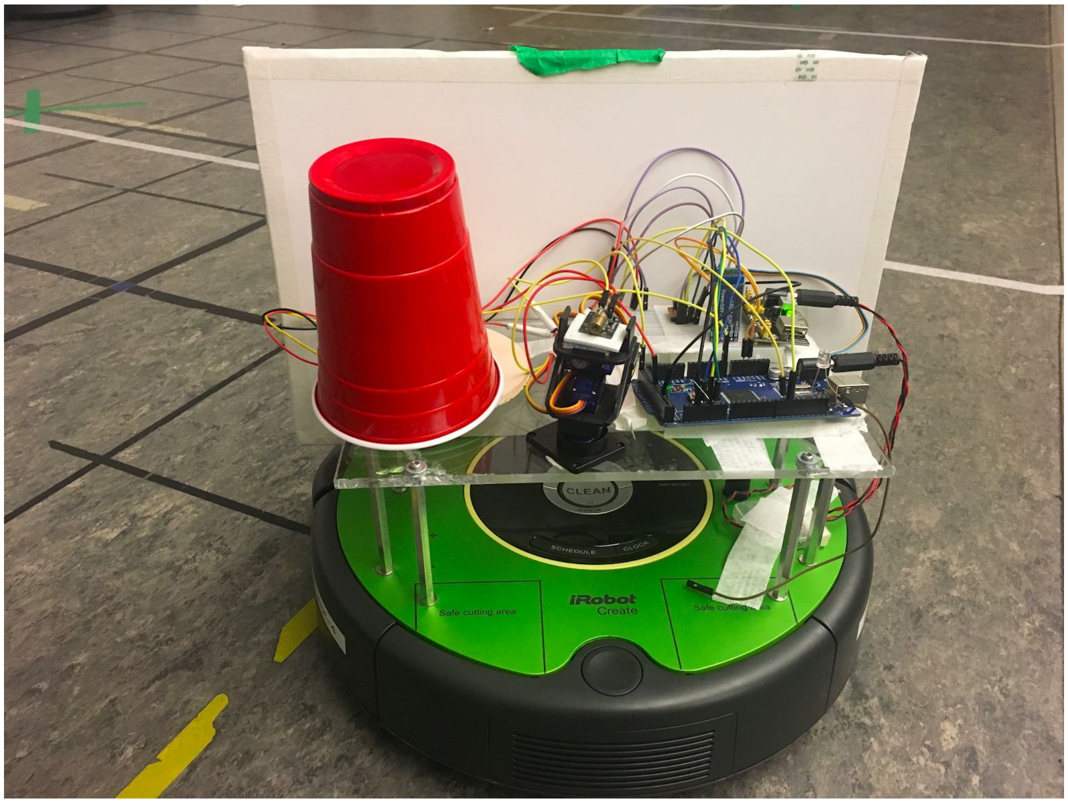


Figure 11: Controller Station

Above we have the controller we actually used. On the left sticking out of the side is the bluetooth used to send commands. The left joystick controls roomba movement, with the select button operating the laser. The right joystick controls the pan and tilt movement, with the select button changing the speed of the roomba from normal mode to speed-boost mode. The LED seen above the arduino lights up if speed-boost mode is on.

Conclusion



Despite our best efforts, we did not win the Defend Your Castle tournament. During our testing and demonstration our roomba behaved as expected. Our setup passed each test in a controlled environment. However, we frequently encountered unexpected behaviour that we could not explain. On occasion, our roomba appeared to get locked in a single command, refusing new incoming commands and ignoring interrupts such as engaging the bumper. Other times our interrupts would cause the wrong reaction; when bumped it would move towards the wall instead of away. Some of this may be due to mistakes in our coding but the behaviour demonstrated by the roomba was often perplexing. In the tournament, we managed to engage in some movement and shooting before our roomba seized up. We were not killed by another player, but we guess that the ambient light set off the death interrupt. In our second round we encountered a new bug that made the roomba unresponsive to the remote. Either way it was a disappointing end to our project. Looking back at our work it would have been helpful to explore the interrupts using the logic analyzer. With the logic analyzer we could have reviewed our system as a whole just like we did in project 2. Unfortunately due to time constraints we were unable to explore this area. Following project 2, we could have seen exact timing behavior of the roomba but this time we would also get hardware reactions too. With this information we might have seen the root cause of the unexplained behaviour of the roomba.

References

[1] Store.arduino.cc. (2019). *Arduino Mega 2560 Rev3*. [online] Available at: https://store.arduino.cc/usa/mega-2560-r3 [Accessed 12 Mar. 2019].

[2] i. Robot and i. Robot, "iRobot Create® 2 Programmable Robot", *Store.irobot.com*, 2019. [Online]. Available: https://store.irobot.com/default/create-programmable-programmable-robot-irobot-create-2/RC65099.html. [Accessed: 06- Apr- 2019].

[3] *Irobot.com*, 2015. [Online]. Available: https://www.irobot.com/~/media/MainSite/PDFs/About/STEM/Create/create_2_Open_Interface_Spec.pdf. [Accessed: 28- Mar- 2019].

[4] *Core-electronics.com.au*, 2019. [Online]. Available: https://core-electronics.com.au/attachments/guides/Product-User-Guide-JY-MCU-Bluetooth-UART-R1-0.pdf. [Accessed: 06- Apr- 2019].

[5] *Ee.ic.ac.uk*, 2019. [Online]. Available: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf. [Accessed: 06- Apr- 2019].

[6] "Joystick Module Pinout, Features, Arduino Circuit & Datasheet", *Components101.com*, 2018. [Online]. Available: https://components101.com/modules/joystick-module. [Accessed: 06- Apr- 2019].

[7] "Arduino KY-008 Laser sensor module - TkkrLab", *Tkkrlab.nl*, 2016. [Online]. Available: https://tkkrlab.nl/wiki/Arduino_KY-008_Laser_sensor_module. [Accessed: 06- Apr- 2019].