CSC 460 Project 3

Spring 2019

Team 13

Hamzah Mansour	V00812108
Christina Araya	V00781339

Table Of Contents

List of Figures

List of Tables

1 Abstract

This report is a part of project 3 in CSC 460 as offered in the spring term at the University of Victoria. It describes the design and implementation of a remote and base system used in a tournament, where the finished projects of several students will compete. The implementation is required to use a modified TTA scheduler from project 2 in the same course and various hardware components as specified in the hardware components section of the report. Some of these elements include programmable boards for the programming of the system and an iRobot Create® 2 Roomba robot to control.

The resultant system needed to be capable of moving, rotating, firing and positioning a turret as well as other functions. This was in order to play a castle defence game where two systems would compete as a defensive guard of the tower or as an attacker of the tower. Both systems would have the same limitations, firing times of the laser turret was limited to ten seconds, and every thirty seconds the robot would switch between an active movement mode to a cruising rotation only mode.

There were some issues with the implementation of this project, the UART communications posed issues in that the definitions were specific, or the lack of a definition would keep communications from occurring normally. At times in the implementation, due to a small modification in the UART code signals from the serial ports that seemed to be functioning would no longer properly communicate with the roomba. Another issue that was resolved differently for the base and remote stations in the implementation was the issue of the UART codes sent. Because of the way commands were sent, the base and remote stations would sometimes go out of sync, however this could be solved by either reading the whole buffer, or clearing it at specific intervals.

2 Project Introduction

The objective of this project[1] is to make a system with a base station (a controller) and a remote station (a robot to be controlled). The system will be used to play a tournament, involving four separate teams forming pairs of robot-controller systems. The two teams are designated as either attack or defence teams. Defence teams will be required to defend a castle and their remote station. Attack teams will be required to defend themselves and attack the defence roomba or the castle. Each member of the teams will individually face a member from the other team and have access to a castle system. Winning criteria for game in the tournament include the following results:

- If the castles are shot, the attack team wins
- If the defence teams robots are dead, the attack team wins
- If the attack robots are both dead, the defence team wins

- If the time is up and the castles are not shot, the team with the most surviving robots wins

In the game of two teams with two robot-controller systems, each robot is built by a team of two. Each round has a time limit of five minutes and during the round two matches would occur involving two robots and a castle is separated by a river made of virtual IR walls.

In phase one of the system building, the controller is required to send commands to the robot allowing the robot to move, shoot a laser, pan and tilt the laser. The robot should also have a target made from a light sensor in a cup, that kills the robot when hit by a continuous, two-second shot from a laser hitting the cup. Each robot has a laser firing limit of ten seconds and are required to cycle between states of full motion and rotation-only movement of the roomba every thirty seconds.

For phase 2, additional semi-autonomous functionality should be built in to the roomba, allowing it to sense and react to a physical barrier or an IR ‘river’ and back away. For both base and remote systems, the time-triggered scheduler from project 2 must be used. Any additional work mentioned in this report was not required to complete the phases and was optional.

3 Implementation

3.1 Hardware Components

Equipment needed to complete the design and implementation as described in this report, similar to project 1[2], excluding some wires and materials to secure the systems:

- Two Arduino ATMEGA 2560 REV3 Boards
- Two Bluetooth radio modules (HC-06 or HC-05)
- Two analog Joysticks
- Two servo motors (SG-90 or SG-91)
- A Pan and Tilt kit
- A laser pointer
- A digital light sensor
- An Arduino LCD KeyPad shield
- Two breadboards and
- A Roomba

The Arduino Mega 2560 board is a microcontroller, with 16 analog pins, 4 serial ports, and 54 digital pins, 14 of which can be used to output PWM. The board also has a USB connector that can be used to load programming onto the board through the first serial port as well as to power the board and any peripherals.

The HC-06 and HC-05 bluetooth radio modules are designed for transparent wireless communication. HC-05 can be used as both a master and a slave module, but HC-06 may only be used as a slave. These modules are used to communicate between the remote and base station

The SG-90 or SG-91 servo motors are rotational actuators that can be sent a PWM signal to set a rotational position, these motors have ranges that take in PWM signals with duty cycles ranging from 500μs to 2500μs.

A digital light sensor is a light sensor mounted on a board that converts the analog light input to a digital output. The sensitivity of the output can be controlled by adjusting a potentiometer on the board.

The LCD KeyPad Shield is an Arduino-compatible shield that has a mounted LCD and a series of digital input buttons. The display allows for 2 rows of 16 characters each to be written to the screen at a time.

The iRobot Create® 2 is a programmable vacuum cleaner robot that can be passed commands using a serial port to move, play music, or change state. It also has a series of sensors that can detect walls, virtual walls, and bumper pushes among others.

3.2 Design

3.2.1 System block diagram

The system is split up into two major components: the remote station and the base station. The system block diagram on the following page outlines the process, data structures, and hardware components involved within each of these two subsystems.

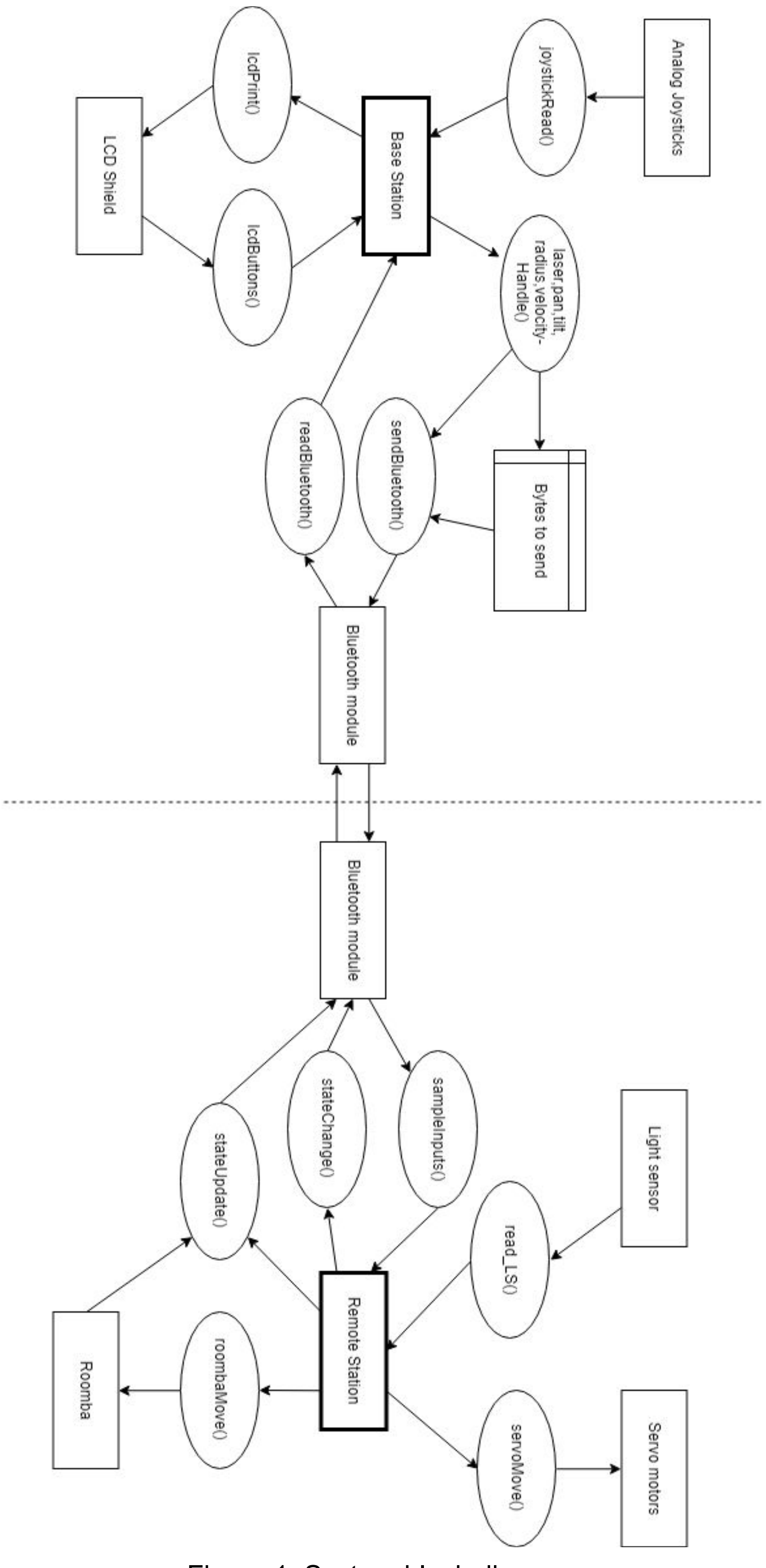


Figure 1: System block diagram

3.2.2 State diagram

The base station has some automatic and semi-autonomous behaviours outlined in the following state flowchart.

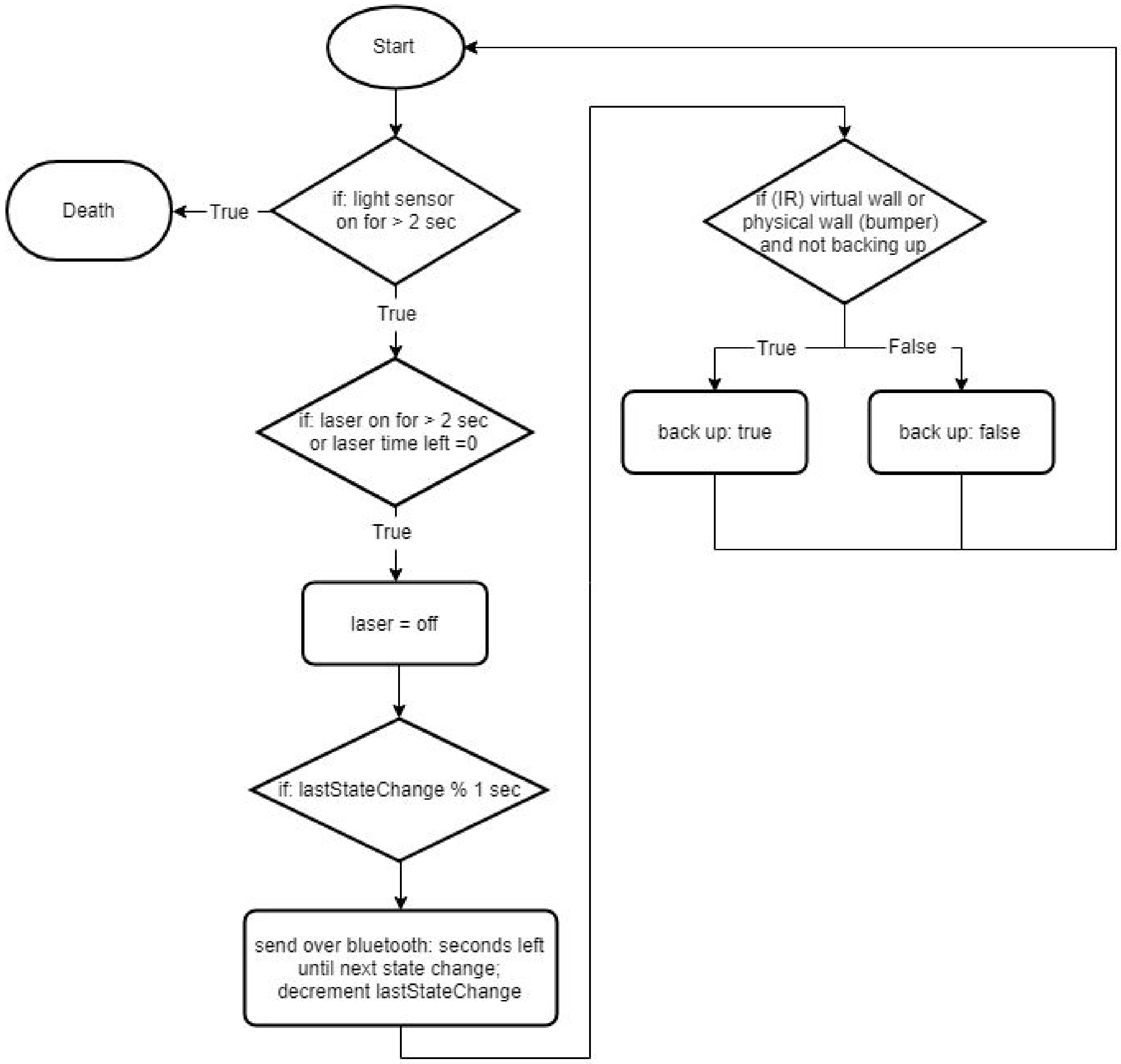

Start
Death
True
if: light sensor on for > 2 sec
True
if: laser on for > 2 sec or laser time left =0
True
laser = off
if: lastStateChange % 1 sec
send over bluetooth: seconds left until next state change; decrement lastStateChange
if (IR) virtual wall or physical wall (bumper) and not backing up
True
False
back up: true
back up: false

Figure 2: System state diagram

3.2.3 Bluetooth communication

A bluetooth protocol was used to send commands and information between the base station and remote station sub-systems. In general, the first byte defined the type of data being sent and the second byte defined the value of the data; this data was either a range of values (for hardware control) or binary values (to represent a state).

The base station received information in the following format:

Table 1: Remote station UART messages

First byte	Data type	Second byte	Data value
0	State	0:1	0: stand-still, 1: cruising
1	Time left to state change	0:29	Time left in seconds
2	Laser ‘energy’ (time) left	0:100	Time left in tenths of a second
3	Light sensor state	0:1	0: safe, 1: light sensor being shot
4	Death	0:1	0: robot is alive, 1: robot is dead

The remote station received information in the following format:

Table 2: Base station UART messages

First byte	Data type	Second byte	Data value
0	Laser	0:1	0: turn laser off, 1: turn laser on
1	Pan servo	0:81	Mapped to servo position -80:80
2	Tilt servo	0:21	Mapped to servo position -20:20
3	Roomba velocity	0:100	Mapped to velocity 1:2000 (mm/s)
4	Roomba radius	0:101	Mapped to -2000:2000 (with special cases handled at -1,0,1)

A UART library was used to make reading this information two bytes at a time from the ATMega2560’s serial communication port (RX2) two bytes at a time simpler. This library utilized a buffer array (that could hold up to 32 bytes at a time), which also provided a way to find out how many bytes were still in the buffer. If there were an odd number of bytes in the buffer, all of the elements in the buffer were read in pairs, then the last lone byte was pushed to the front of the buffer, while the rest of the buffer was cleared. This made communication over bluetooth more stable. In addition to handling this particular case, the remote station also reset the buffer every two seconds to ensure both subsystems stayed in sync.

3.3 Execution

3.3.1 Final Design

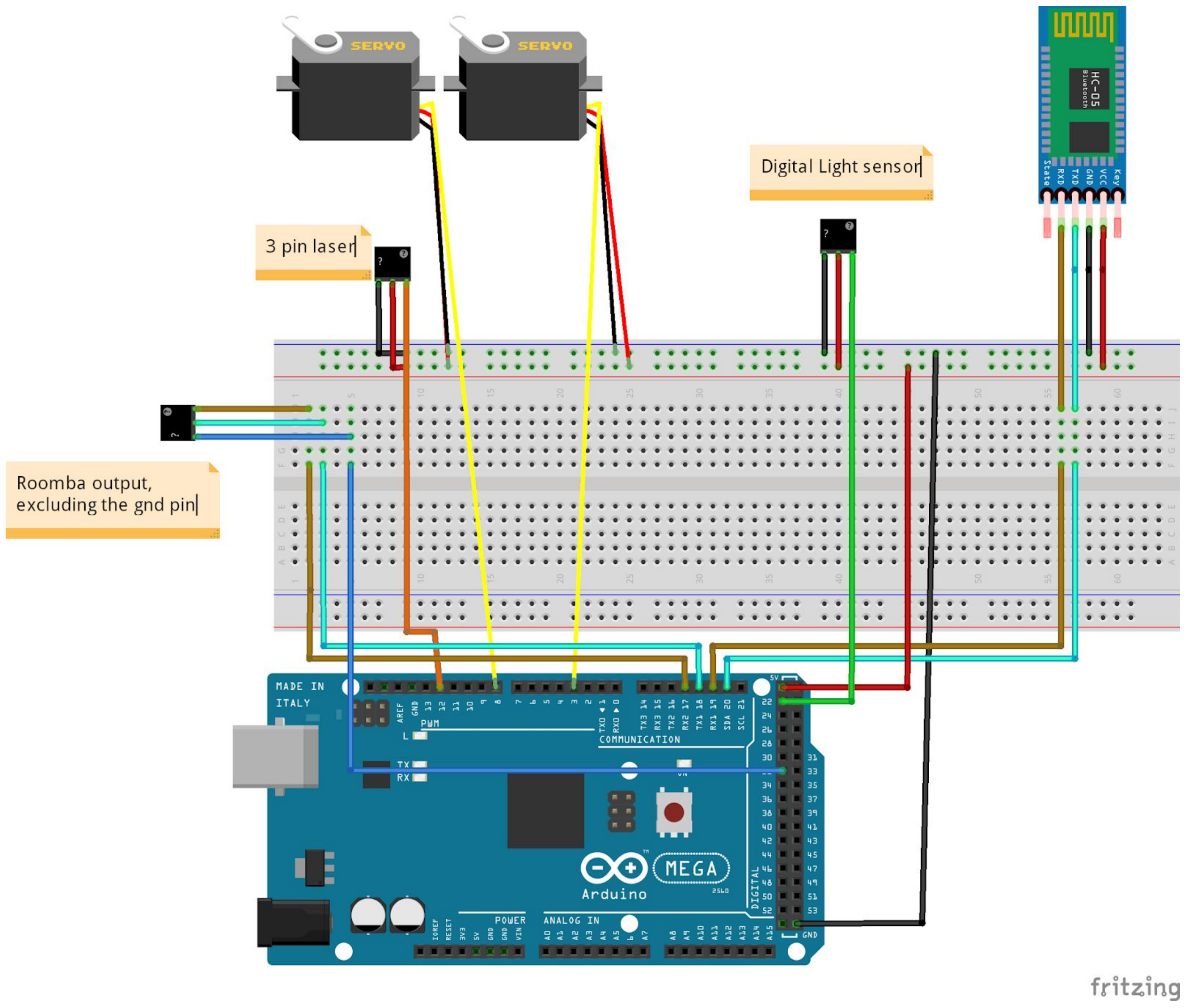


Figure 3: Remote station wiring diagram

The above wiring diagram contains a few dummy parts, representing the roomba interfacing, digital light sensor and laser from the completed design. Each of the wires are colour coordinated where the black wires are ground, red are power(VCC) cyan is RX outputs, brown are TX inputs, yellow are servo PWM drivers, green is the light sensor input, orange is the laser output and blue is the DD pin for the roomba. The TX1 and RX1 connections are used for the roomba communication, TX2 and RX2 is used for bluetooth communication. From left to right of the arduino pins used PWM pin 12 drives the laser, PWM pins 8 and 3 drive tilt and pan respectively, the digital pin 22 reads the Light Sensor output and digital pin 32 is used for the roombas DD pin.

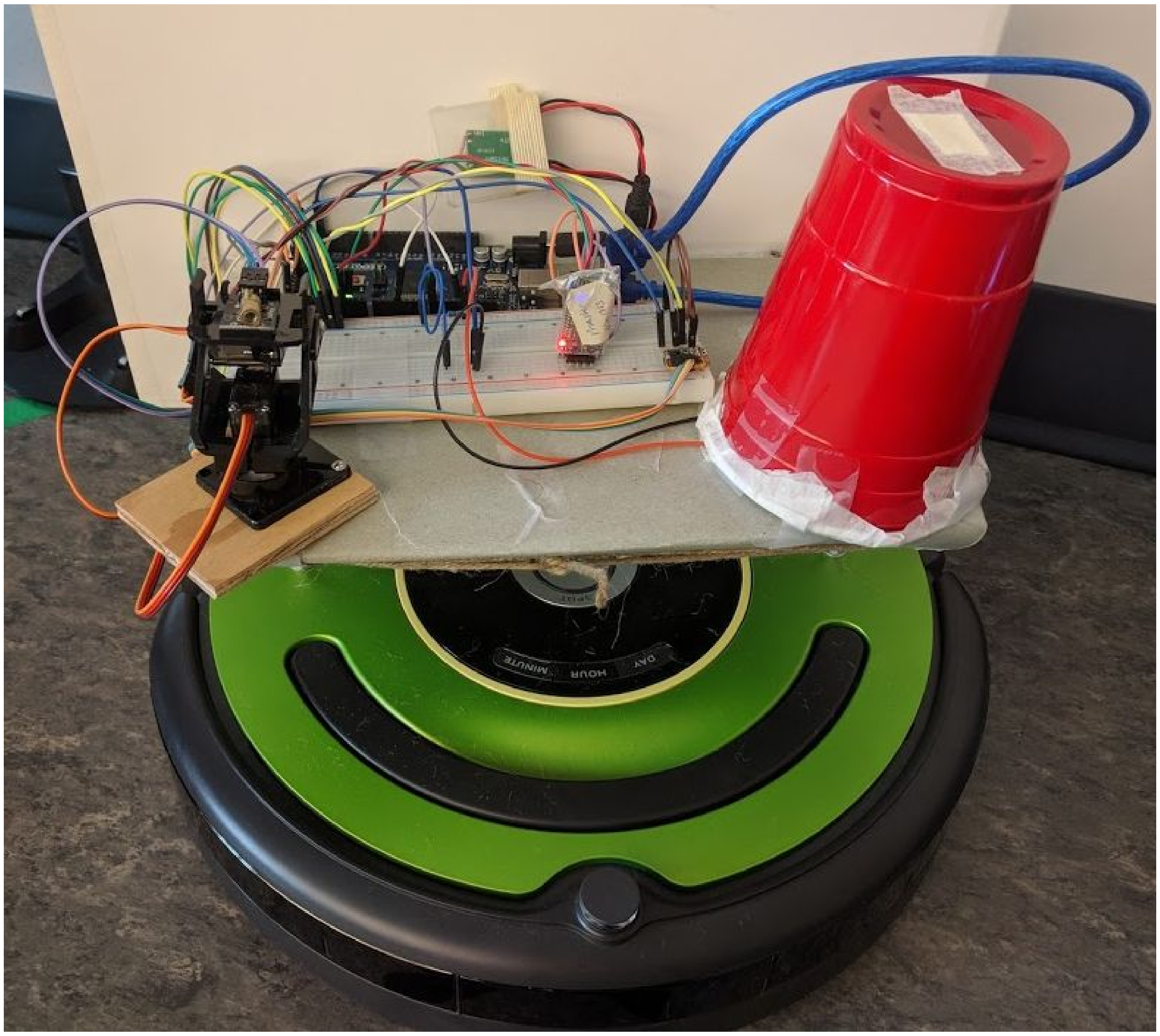


Figure 4: Remote station image

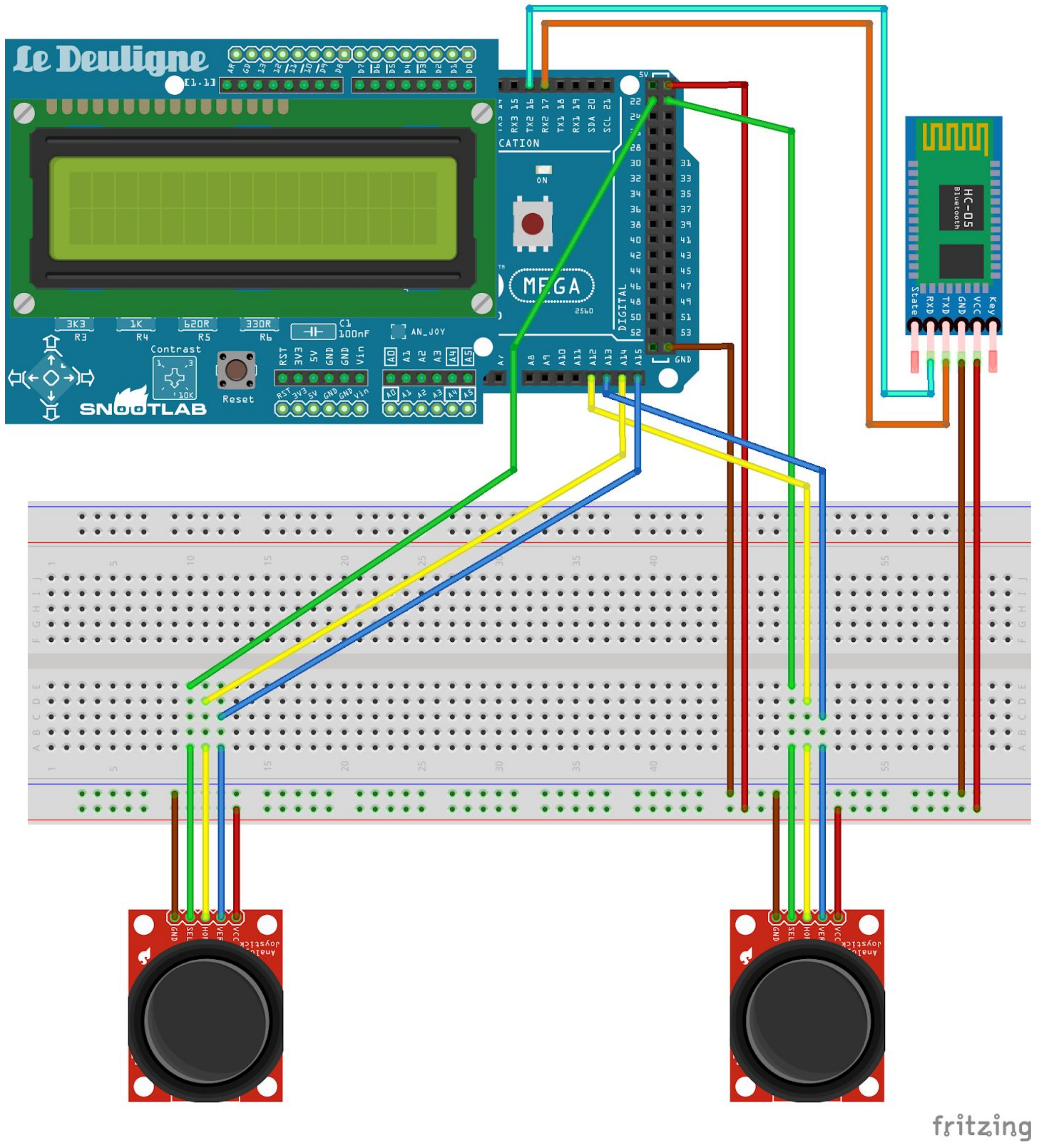


Figure 5: Base station wiring diagram

The wiring diagram above indicates how the base station (controller) was put together and how each component interfaced with the ATMega2560 board. The LCD shield was placed directly on the board, occupying the pins on the left side of the board for output information to the operator of the robot. The LCD shield was also used for input via the buttons on the shield, using analog pin 0. The base station communicated to the remote station via RX2/TX2, which were connected to the bluetooth module. The two joysticks were used as input, with the horizontal and vertical controls of joystick 1 using analog pins A14 and A15, as well as A12 and A13 for the two axes of joystick 2. The digital input buttons of each joystick were also read using digital pins 22 and 23. These components were all securely placed in a cardboard box, as shown in the image below.

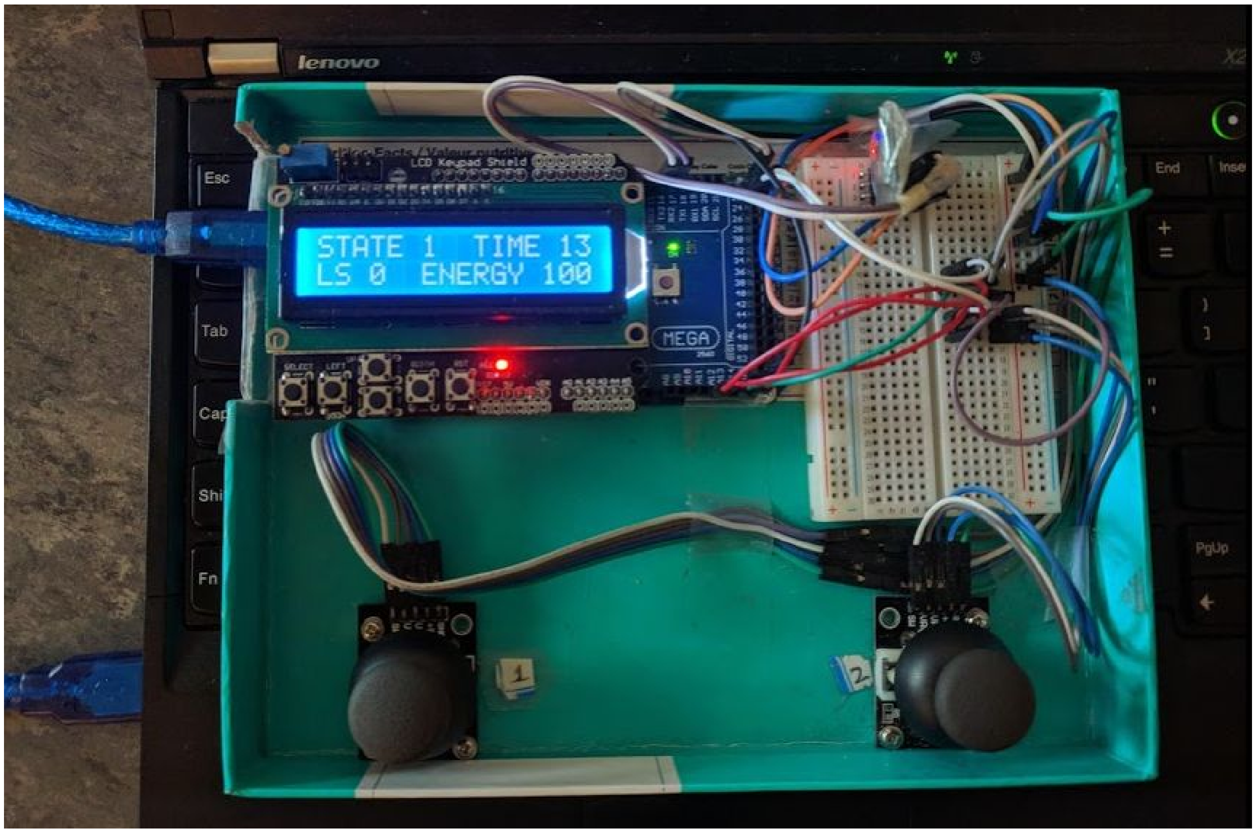
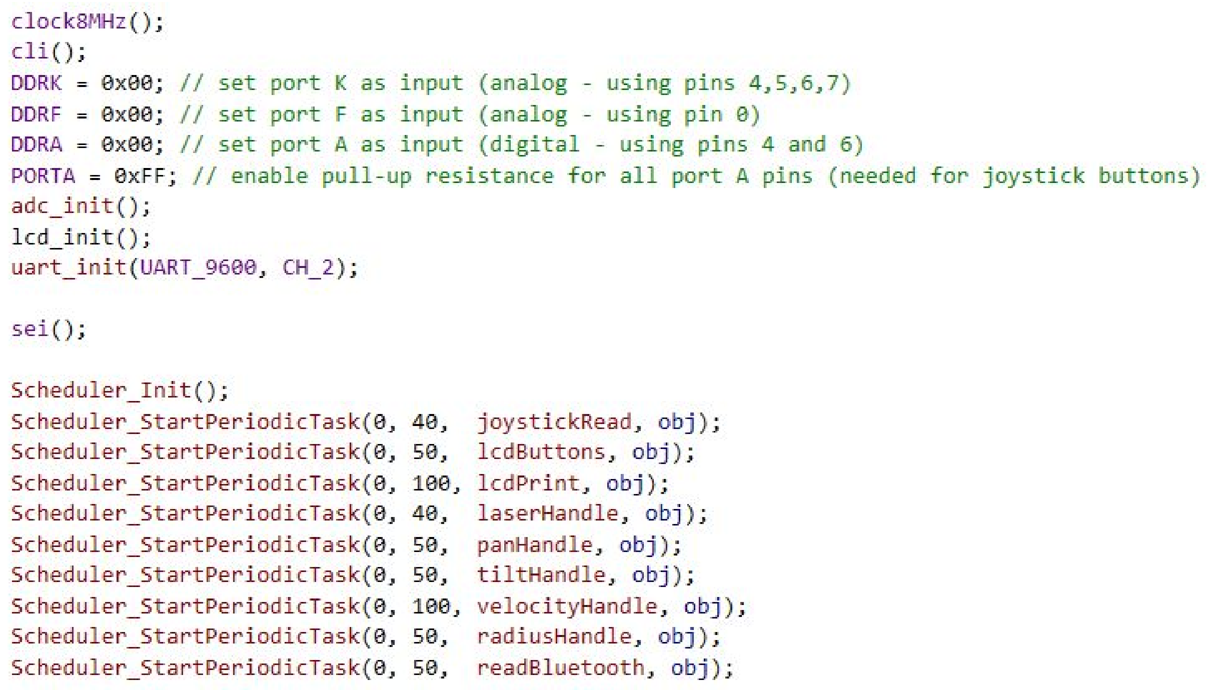


Figure 6: Base station image

3.3.2 Initialization

The initialization of the base station (Fig. 7) required analog and digital pins to be set as inputs, in this case K, F and A while input pullup was enabled by setting all pins in A as on before starting execution. Then the analog reader and lcd code is initialised, along with the uart communication onto a baud rate of 9600 across bluetooth; this baud rate is used on the remote station for valid communication. Then after interrupts are restored the scheduler is initialized along with all of the periodic tasks.

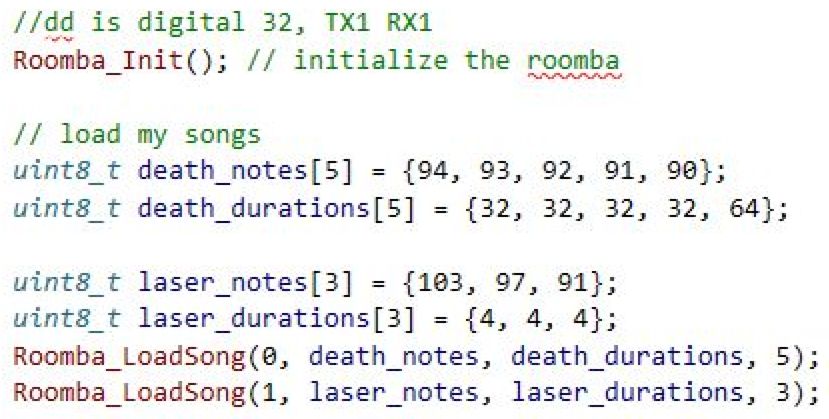


```
clock8MHz();
cli();
DDRK = 0x00; // set port K as input (analog - using pins 4,5,6,7)
DDRF = 0x00; // set port F as input (analog - using pin 0)
DDRA = 0x00; // set port A as input (digital - using pins 4 and 6)
PORTA = 0xFF; // enable pull-up resistance for all port A pins (needed for joystick buttons)
adc_init();
lcd_init();
uart_init(UART_9600, CH_2);

sei();

Scheduler_Init();
Scheduler_StartPeriodicTask(0, 40,  joystickRead, obj);
Scheduler_StartPeriodicTask(0, 50,  lcdButtons, obj);
Scheduler_StartPeriodicTask(0, 100, lcdPrint, obj);
Scheduler_StartPeriodicTask(0, 40,  laserHandle, obj);
Scheduler_StartPeriodicTask(0, 50,  panHandle, obj);
Scheduler_StartPeriodicTask(0, 50,  tiltHandle, obj);
Scheduler_StartPeriodicTask(0, 100, velocityHandle, obj);
Scheduler_StartPeriodicTask(0, 50,  radiusHandle, obj);
Scheduler_StartPeriodicTask(0, 50,  readBluetooth, obj);
```

Figure 7: Base station initialization

In order to start the remote station execution some elements needed to be initialized in the code. In setup, the ports would need to be set up for inputs or outputs for different purposes, the uart channels CH_1 for serial 1 and CH2 for serial 2 would need to be initialized, music be loaded , PWM initialized and scheduler to be set up.

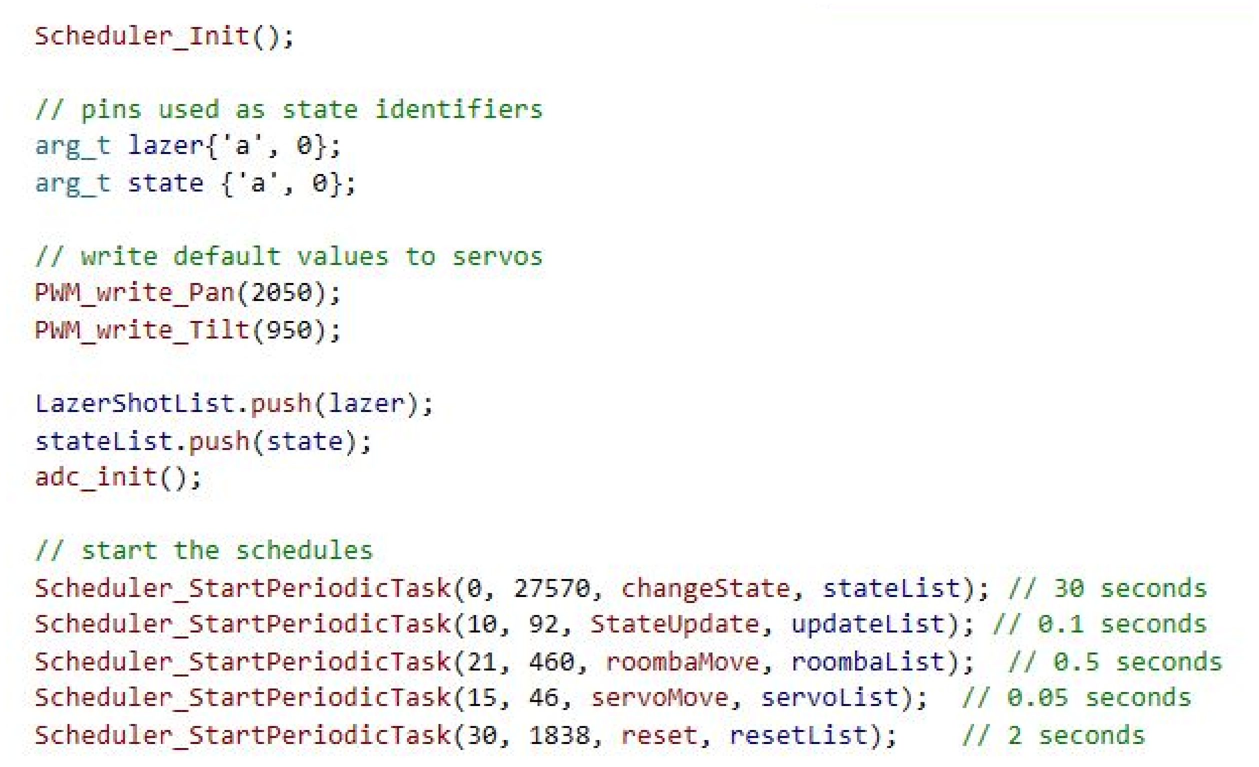
For testing and laser firing purposes the L and B ports are set as outputs, while in order to get the input from the light sensor port A was set as an input. Using an existing library modified with a new uart implementation, the roomba is initialized using channel 1 and the songs are loaded.



```
//dd is digital 32, TX1 RX1
Roomba_Init(); // initialize the roomba

// load my songs
uint8_t death_notes[5] = {94, 93, 92, 91, 90};
uint8_t death_durations[5] = {32, 32, 32, 32, 64};

uint8_t laser_notes[3] = {103, 97, 91};
uint8_t laser_durations[3] = {4, 4, 4};
Roomba_LoadSong(0, death_notes, death_durations, 5);
Roomba_LoadSong(1, laser_notes, laser_durations, 3);
```

Figure 8: Roomba initialization, laser and death cry

The serial 2 uart is initialized for use with the bluetooth module, and the PWM outputs are initialized and set to starting values, and the adc is initialized. Finally the scheduler is defined and periodic tasks pushed as in Figure 9.



```
Scheduler_Init();

// pins used as state identifiers
arg_t lazer{'a', 0};
arg_t state {'a', 0};

// write default values to servos
PWM_write_Pan(2050);
PWM_write_Tilt(950);

LazerShotList.push(lazer);
stateList.push(state);
adc_init();

// start the schedules
Scheduler_StartPeriodicTask(0, 27570, changeState, stateList); // 30 seconds
Scheduler_StartPeriodicTask(10, 92, StateUpdate, updateList); // 0.1 seconds
Scheduler_StartPeriodicTask(21, 460, roombaMove, roombaList);  // 0.5 seconds
Scheduler_StartPeriodicTask(15, 46, servoMove, servoList);  // 0.05 seconds
Scheduler_StartPeriodicTask(30, 1838, reset, resetList);    // 2 seconds
```

Figure 9: Roomba scheduler initialization

3.3.3 Tasks

Tasks are divided into a few categories for the base and remote stations, there are periodic, oneshot and idle functions. Periodic tasks happen with a specific frequency, and are consistent, oneshot tasks are not urgent, but are scheduled when there is available time. Idle functions are not tasks at all, but are instead functions that are run in the idle loop and will run whenever there is available time for them to run.

Base station tasks were all handled as periodic tasks of varying frequencies, in the TTA scheduler used each tick takes approximately 1.08 ms each. The most frequent tasks for the base station took 40 ticks, joystickRead and laserHandle, which read the analog inputs of the joystick x and y positions, as well as the joystick button presses. There are then 5 periodic tasks scheduled at 50 ticks each, lcdButtons, panHandle, tiltHandle, radiusHandle and readBluetooth. LcdButtons reads the LCD Shield button presses and stores the value. The tilt, pan and radius handle functions map the x and y values from joystick 1 and 2 to an appropriate speed and sends the appropriate code to the remote station if the value had changed. The range of the joysticks typically goes from 0 to 1024 and needed to be mapped to speed ranges -10 to 10 for tilt, -40 to 40 for pan and 0 to 101 for radius. The readBluetooth task scans the input buffer for communications from the remote station, and updates the states based on the values received using the parseBytes function. The least frequent tasks are the velocityHandle and lcdPrint tasks at a 100 tick period, velocityHandle maps the output from joystick 2 x to a velocity range between 0 and 100 and sends the result over bluetooth to the remote station. The lcdPrint function, selects a print type based on the state of the select button press and displays messages on the LCD shield based on the selected output type. Print types are input, output, compete or death, input displays the joystick inputs, output displays the speeds being sent by the base station, compete displays the states returned from the robot for competition purposes and death displays a message indicating the dead state of the remote station.

Remote system tasks from order of most to least frequent start with the idle tasks. Idle tasks must be short, as long idle tasks may interfere with the scheduling of periodic tasks. The functions called in idle are read_LS() and sampleInputs(). The function read_LS reads the input from digital pin 22, the reading from the digital light sensor, this is used to determine if the remote station is in danger of dying, and how long it's been in danger. The other function sampleInputs reads UART from the bluetooth module, this is where all the commands are read from the base station. The values gathered from the UART reading are used to determine the desired speeds of the servos, roomba and roombas rotation, as well as the status of the laser. There is one function called by sample Inputs that doesn't have a fixed frequency as it depends on the UART message received. If the 00 or 01 message is received, a oneshot task laser shot is called this task is started the next available idle period, it changes the state of the laser, samples the time and plays a song when the laser starts to fire.

Periodic tasks for the remote system starting with the most to the least frequent tasks can be seen in Figure 9. In order to give the most precise movements in the pan and tilt servos, a periodic function called servoMove that controls the servos is the most frequent at a 0.05 second period. This function calls a secondary function for the pan and tilt systems that limits the change in speed and sets the position of the servos. The pan servo has a wider range and its PWM velocity can be changed by 5 as a maximal adjustment, and the PWM range is between 2100 and 750 micro seconds. The tilt system has a smaller range and the speeds can be adjusted by 2, and PWM ranges are between 1000 and 500. The speed for the pan and tilt servos are retreived from the sampleInputs function and are mapped from input ranges to 80 to -80 and -20 to 20 respectively. The second most frequent periodic task for the roomba, was the state update function every 0.1 seconds which updated the base station with the remote station states, including how much laser fire was left, how much time left between movement states, if the roomba was in danger from laser fire and whether the remote station died. It also detects obstacles for the roomba, sets a backup flag and backs up the roomba when an obstacle is sensed. The task limits the amount of laser fire started at a time and shuts down the system after playing a jingle upon death. Every 0.5 seconds the roombaMove task is scheduled, its ranges determined from sampleInputs are between -300 to 300 for velocity and from -2000 to

2000 for radius where larger values are smaller turns. The roombaMove function doesn't regulate its velocity rate change, instead it checks for edge cases and sends the modified velocity and rotation values to the roomba. If the backup flag is set nothing occurs, if the remote station is in standstill mode, only rotation is allowed, and if there is a rotation value and no velocity, the velocity is mapped to the rotation value. The reset task was introduced to solve issues to do with bluetooth command ordering, since two bits are being sent through bluetooth for commands sometimes strange behaviours would occur due to off by one errors. The reset task simply resets the UART buffer for channel 2 every 2 seconds. The least frequent function change state occurs every 30 seconds, changing the roomba movement controls between full movement and rotation only states, it also updates the base station with the current state and updates a counter for the state update functions calculations.

3.4 Observations

The biggest issue encountered in this project was the UART library implementation and communication to the bluetooth and roomba channels. This is the part of the implementation that took the longest and the part that most often caused problems. When starting the implementation of the project, a sample project was used that handled roomba communication, but the UART implementation was for only one serial port and wasn't implemented for the atmega2560 specifications. In order to make sure that it functioned, the correct F_CPU value needed to be specified along with the correct baud rates, it needed to have the ability to use 2 serial ports for the addition of bluetooth and the initialization needed to have the correct values. If the incorrect flags are set to the receiver and transmitter of serial 0, serial 0 may block transmission, and because the port is used to upload code, it would prevent uploading of code, it's safer to use the other serial ports 1, 2 or 3 for the bluetooth and roomba commands. This implementation used serial ports 1 and 2, if 0 is used for either the bluetooth or the roomba and is initialized correctly, it would still be necessary to disconnect the wiring to those devices in order to upload programming.

Another minor issue was one to do with our mode of communication, the bluetooth would send 2 bytes per command, based on the two bytes sent different functions would be affected, however if the bytes went out of sync, undefined behaviour would occur. The syncing issue was also observed in project 1 of this course and was solved in the same way for the remote station by resetting the uart communication every 2 seconds.

Also the implementation of the light sensor on the roomba as a target was later swapped to use a digital light sensor as the original implementation would cause the remote station to die at random intervals. This was due to the reliability and sensitivity of the light sensor, this issue occurred less frequently with the digital light sensor and made the finished system more workable.

4 Conclusions

In our system design[3], commands were sent using uart between a remote and base station in order to control a roomba and return its state. This ended up being a major issue with the project and often hindered progress. Other issues encountered included communication syncing issues between the roomba and controller setups. As well as light sensor issues that caused us to resort to using a digital light sensor that was more reliable.

The roomba and attached servos were capable of being piloted by the x and y joystick positions on the controller. The laser turret mounted on the pan and tilt servos was controllable by a joystick button press. The light sensor target cup could be shot for 2 seconds which would result in the death of the roomba, and the message would be broadcast to the controller as a song played. The roomba read sensor data and would back up in case of a collision with the IR wall, a physical wall or a bumper push. The state of the roomba would be sent at a maximum rate of 1/10th of a second and would switch the roomba between full and rotation only movement. The laser could only fire for a little over 2 seconds a shot, and over the lifetime of the roomba could fire for a total of 10 seconds. This information along with the seconds left until the next state change, available fire time left, light sensor state, current movement state and death state was sent to the controller for display for the sake of the tournament match[4].

5 References

[1] M. Cheng, "Project 3", Webhome.csc.uvic.ca, 2019. [Online]. Available: http://webhome.csc.uvic.ca/~mcheng/460/spring.2019/p3.html . [Accessed: 05- Apr- 2019].
[2]M. Cheng, "Project 3", Webhome.csc.uvic.ca, 2019. [Online]. Available: https://webhome.csc.uvic.ca/~mcheng/460/spring.2019/p1.html . [Accessed: 06- Apr- 2019].
[3] github.com, 2019. [Online]. Available: https://github.com/HamzahMansour/csc460/tree/master/project3 . [Accessed: 06- Apr- 2019].
[4] H. Mansour, “CSC460”, [Online]. Available: https://photos.app.goo.gl/ZLSzDHPTCWxY7wQ36 . [Accessed: 06- Apr- 2019].