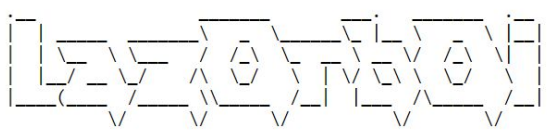

Project Report

String to laser display

Team Quartz
Team # 74
Members Omar Alashqar, Connor Barker, Yathartha Panigrahy

Table of Contents

//lazOrbOi™

Project Report - String to laser display

Project Overview

What is the Project about?

Lazorboi™ is able to receive a string and showcase the visual representation of it. It analyses the characters and converts it to information which the Omega understands (as described later).

What would you have liked it to do if it could do everything?

If the lazorboi™ could do everything we would've liked it to do, then we would have wanted it to analyse different forms of data (frequency, string, sound) and show the visual representation of it in the form of a laser.

What subset does it do?

It analyses a string and showcases that as a laser display by controlling the speed and direction of the motors. It can also project a changing design based on sine and cosine waves, the amplitude of which have been provided by the user.

System design

System components

Here are the components used in this projects:

- Omega2
- Wires
- Motor driver (Sparkfun Dual TB6612FNG)
- Laser pointer
- 2x 6V DC motor
- 2x small mirror
- 6V battery pack with batteries

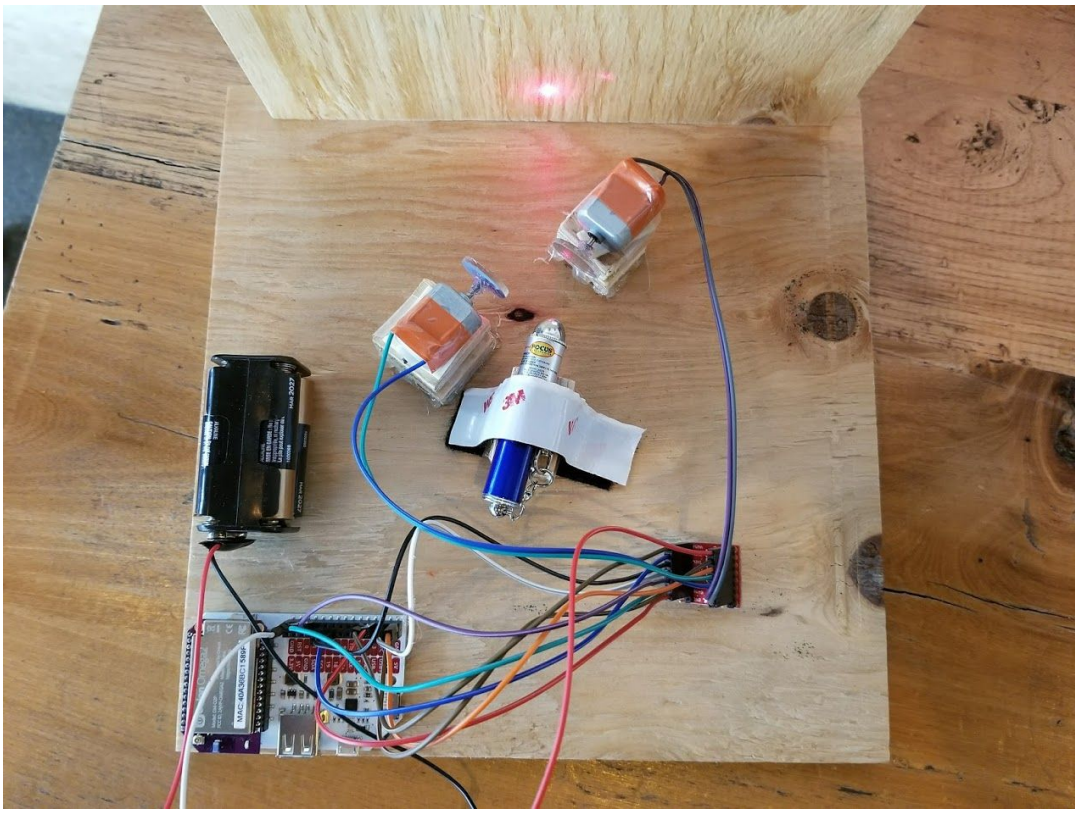

Figure 1 - The complete setup for lazorboi™

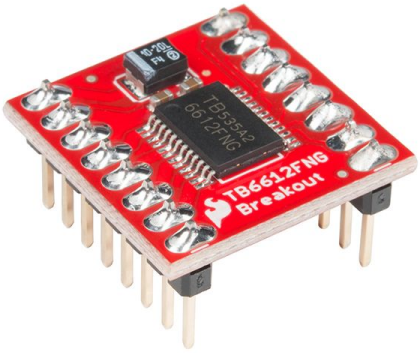

Figure 1 - Motor driver (Sparkfun Dual TB6612FNG)

In order to achieve different shapes with a single laser pointer, the mirrors can be glued onto the motors on a slight angle. As the mirrors spin, the laser projection moves around. As shown in **Figure 1**, the laser pointer is oriented such that it reflects off of one mirror, onto the other mirror, then onto a wall onto which the display will be projected.

The motor driver, shown in **Figure 2**, is being used to supply the motors with the appropriate voltage, speed, and direction that they should be at. This is an important component since it regulates the voltage such that the motors don't burn. Furthermore, the motor driver is necessary for controlling the direction of a motor, along with a PWM based speed. A motor takes in two inputs: one being a PWM speed with different duty cycles mapping to different speeds, and the other being a digital signal of high or low mapping to different directions.

The 6V battery pack provides power to the motor driver, which provides a varying voltage to the motors. The PWM frequency should be set to a value between 6,000Hz to 20,000Hz and a duty cycle that controls speed that has a percentage value.

Table 1 lists all the mapping for the wires for the whole project. Note that the polarity on the motors doesn't matter since the direction they spin at is arbitrary.

Table 1 - The mapping of all the required connections

From	To
Omega2 GPIO 0	(Motor Driver) Pin BIN2
Omega2 GPIO 1	(Motor Driver) Pin BIN1
Omega2 GPIO 11	(Motor Driver) Pin STBY
Omega2 GPIO 19	(Motor Driver) Pin AIN1
Omega2 GPIO 18	(Motor Driver) Pin AIN2
Omega2 GPIO 3	(Motor Driver) Pin PWMB
Omega2 GPIO 2	(Motor Driver) Pin PWMA
Omega2 5V	(Motor Driver) Pin VCC
Omega2 GND	(Motor Driver) Pin GND
Omega2 GND	(Battery Pack) GND (-)
Battery pack VCC (+)	(Motor Driver) Pin VM
Motor 1 Connector 1*	(Motor Driver) Pin A01
Motor 1 Connector 2*	(Motor Driver) Pin A02

Motor 2 Connector 1*	(Motor Driver) Pin B01v
Motor 2 Connector 2*	(Motor Driver) Pin B02
Omega2 Micro-USB	Computer USB

Software design

Program structure

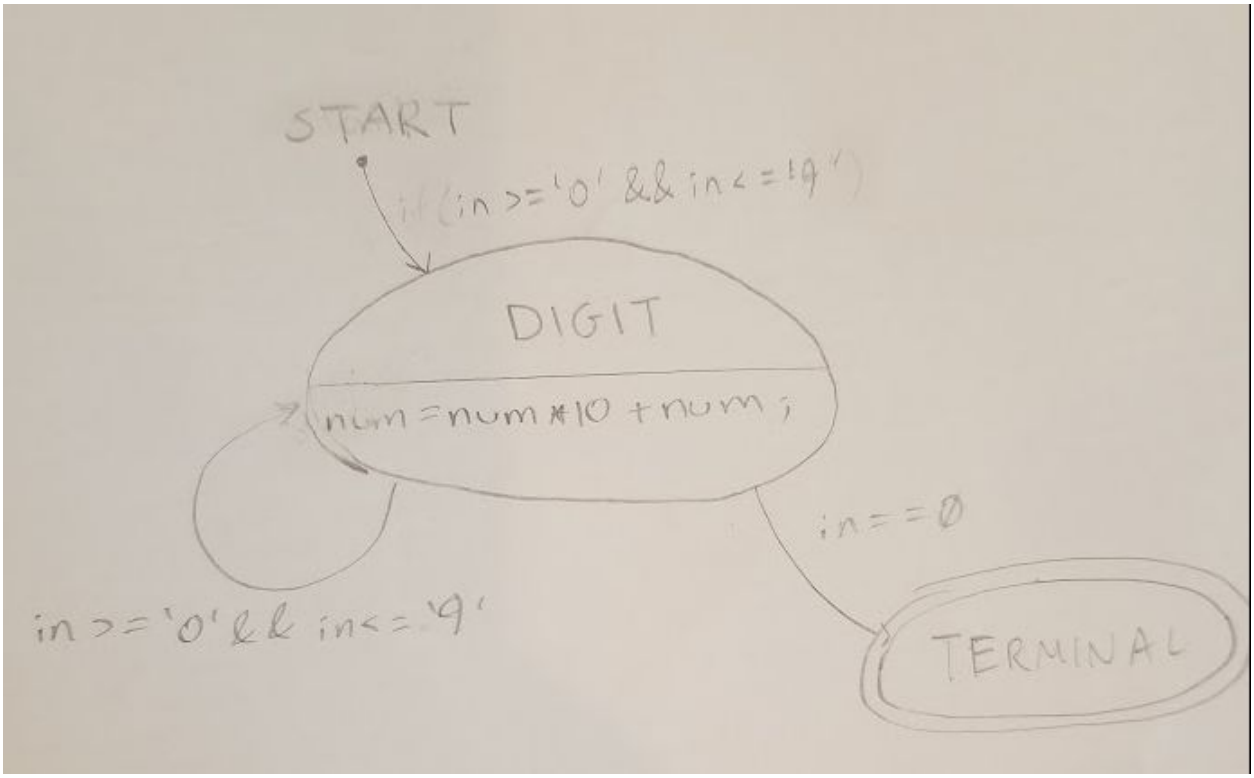

Figure 3 - String to integer state machine

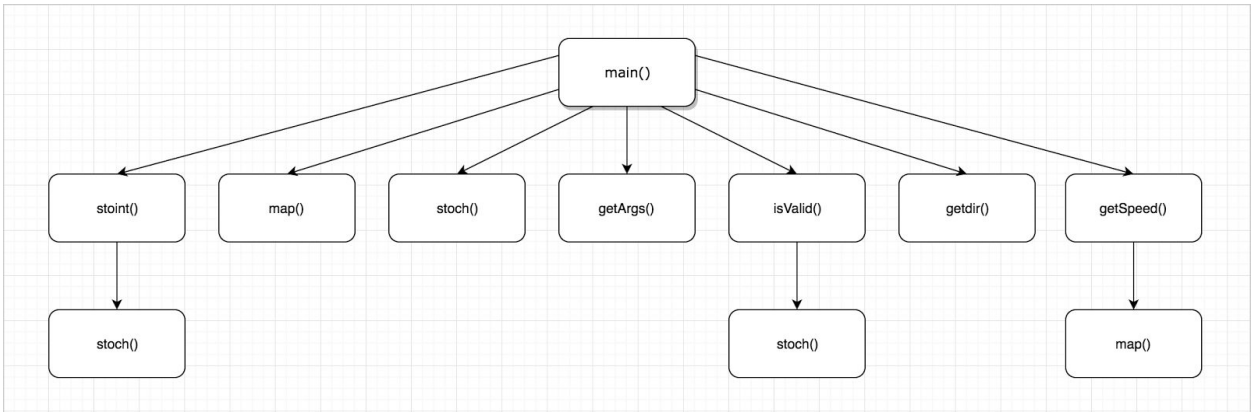

Figure 4 - Function call tree

The program doesn't have any classes or structures. In order to have a sense of modularity based on functionality, some code is written in different c++ files. Those files are then included in the main file when compiling.

Analyzing a sinusoid

One of the features of lazorboi™ is taking in two amplitudes as input and displaying a changing design based on sinusoidal functions with those amplitudes. The sinusoids used to obtain speed and direction information are as follows:

$$\alpha = Asin(0.1x) \text{ and } \beta = -Bcos(0.2x)$$

where A and B are integer amplitudes taken from user input, and x is a constantly increasing index in order to run through all the values. Note that each equation is controlling one motor.

The direction is based on the sign on the computed value. To determine speed, the computed value is linearly mapped to a range of [10, 100].

Since small changes in the duty cycle of the PWM going into the motors has very minimal effect on the design, a quick fix had to be implemented. After a very short amount of time with the motors running with those values, they're set to stop for an even shorter amount of time, then run with the new values.

Since it takes some time for the motors to actually slow down, they effectively never end up stopping. The resulting effect is that the motors are constantly accelerating and decelerating.

Analyzing a string

The main feature of the lazorboi™ involves the program decomposing a string, a word, sentence, etc. into its visual laser representation. The process sounds like it would be arbitrary, and it is, but the beauty of it is that art can be produced out of anything.

It's best to use an example in order to explain the algorithm. Suppose the string being analyzed is **sadreacc**. From this, multiple data points that correspond to the speed and direction of each motor must be extracted. Lazorboi™ iterates through the whole string and extracts pairs of letters in order to determine the speed and direction of the motors. For every pair, the motors would run with the determined values for two seconds.

Following the example, the first pair of letters that would be analyzed would be **[s, a]**. The first character gets processed with two different algorithms, which are described below, in order to extract the corresponding speed and direction for the first motor. Similarly, the second character is processed for the second motor.

Two seconds later, the next pair **[a, d]** is analyzed, and so on. Notice that the same character was used twice. This simply ensures that more character pairing possibilities are explored and visualized.

Algorithm for determining speed

Given a single character, duty cycle value between 20 and 95 must be determined. The values are restricted to that range such that the motors don't spin too slowly and not at their maximum speed.

The calculation for ω, a constant for every letter, is defined as follows:

$$\omega = \frac{A \cdot N}{26\pi}$$

where A is the ASCII decimal representation and N is the character's position in the alphabet. After that, ω is mapped onto the range [20, 95] with a basic linear mapping.

Algorithm for determining direction

Given a single character, a direction of clockwise or counterclockwise must be determined. First, the character is converted to its decimal representation in ASCII. Then, the number of 1s in the binary representation of that decimal is counted. This is done with binary shifting. With that, an even number of 1s corresponds to a certain direction, while an odd number of 1s corresponds to another.

Process control

The software design process took place in three main stages. Firstly, the correct sequence of system commands was discovered, primarily through trial and error. The library being used to control the GPIO pins of the Omega2, `fast-gpio`, operated somewhat differently than its documentation suggested. Namely, each pin's control state had to be set to `output` before the actual output of the pins could be set. According to the documentation, `fast-gpio` should have done this implicitly when the argument `set` was used to change the output of a pin.

Secondly, a command-line tool was written to operate the motors more easily, and without calling the commands manually. For ease of testing, this was done using only standard libraries, meaning compiling would be slightly easier. The tool, which will be hereafter referred to as *ground-control*, used cin to take constant user input to run, stop, reverse, or change the speed of the motors, either independently or simultaneously. The purpose of this function was twofold; it allowed for much easier testing of the complete project during the hardware setup phase, and it provided a base upon which to construct the rest of the project's code.

Finally, the core functions - meaning, those actually used to complete the project's objective - were created and implemented. The work was done modularly, with each team member building their functions with the assumption of some kind of input and output from the other team members' code. For example, code written in *ground-control* was designed around 'placeholder' functions that had not yet been written, but would accept an input and return a specific output. This design process was efficient, as each team member could focus exclusively on their own task, while treating their teammates' work as black box functions.

System independent components

1. Lazorboi.cpp
2. Utils.cpp

System dependent components

1. Ground-control.cpp

Logging

The project logging was done in *ground-control*, using the `<fstream>` library and the `ofstream` object. Time logging was accomplished using `localtime` objects

from the `<time.h>` library. The `outfile` object was created and opened at the beginning of the `main()` function, and lines were passed to `outfile` every time an action which needed logging occurred. Each line was headed by a new timestamp. At the end of the `main()` function, `outfile.close()` was called to save the output to a .log file.

Source code files

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <cmath>
#include <fstream>
#include <time.h>

#include "utils.cpp"
#include "lazorboi.cpp"
```

Testing

The main testing of the hardware in the project was completed after the creation of *ground-control*, which made running specific pwm commands much easier. The project's hardware was assembled using trial and error, as the positioning of the mirrors demanded much testing.

Critically, nothing was assembled permanently until all testing had been completed, which allowed for several iterations of the hardware setup. After discovering that the tilt angle of the mirrors on the motors was slightly larger than anticipated, the motors were mounted higher using small platforms so that the laser light was projected completely onto the wooden display screen.

Limitations

What it could have been

Initially, lazorboi™ was meant to analyze audio frequency signals and convert them into a laser display. Due to hardware complications regarding streaming audio in from a computer, the project was re-designed and has become a string-to-laser display.

Streaming audio into the Omega2 via a mic-in from a different computer or from a plug-in mic, even with the help of external libraries such as PortAudio, is not trivial.

Hardware limitations

As mentioned before, most of our functionality utilizes software based PWM in order to control the motors at different speeds. This introduced an obstacle since PWM control, in this case, was very unstable. The duty cycle being outputted was not very consistent. Because of this limitation, when a new design is meant to be displayed, the motors abruptly change their speed and direction instead of smoothly transitioning between them. This is not as pleasing to the eye.

Lessons learned

- Dealing with hardware components and IoT devices proved to be more difficult than initially thought. Not only does the software have to work, but maintaining the hardware, regulating the voltage going in, ensuring the wires are mapped properly, etc. all factor into the project.
- Starting the software implementation earlier is a very good idea since complications with hardware or software components start appearing. The more time is allocated for debugging and figuring out these complications, the better the project would turn out.

Work cited

Python Playground - by Mahesh Venkitachalam

Onion Omega2 documentation
https://docs.onion.io/omega2-docs/

Motor driver (Sparkfun Dual TB6612FNG)
https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide?_ga=2.30301241.761689168.1512412039-767338479.1509830418&_gac=1.117322100.1509833471.Cj0KCQjwyvXPBRD-ARIsAIeQeoG3KQ0yFTC-ZGrkSI-ZOJ5qMIP8mVmRkw0deVJSuu7NEWU0IQuzjCkaAjU3EALw_wcB

Appendix

Peer contributions

Connor - Wiring, main command-line tool (except sinusoid), logging
Omar - Building project base, code for analyzing strings and sinusoids, implemented error checking for the command-line tool, string to int SM.
Yathartha - Functions that are dependencies for analysing the inputted string and converting it to instructions for the Omega2.

Source code

1. ground-control.cpp

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <cmath>
#include <fstream>
#include <time.h>
using namespace std;

#include "utils.cpp"
#include "lazorboi.cpp"

const int DIR_CW = 0;
const int DIR_CCW = 1;
const int FREQ = 6000;
const string delim = " ";

int main() {

     // Initialize output states
     system("fast-gpio set-output 0");
     system("fast-gpio set-output 1");
     system("fast-gpio set-output 2");
     system("fast-gpio set-output 3");
     system("fast-gpio set-output 11");
```

```
     system("fast-gpio set-output 18");
     system("fast-gpio set-output 19");
     system("fast-gpio set 0 0");
     system("fast-gpio set 1 1");
     system("fast-gpio set 2 0");
     system("fast-gpio set 3 0");
     system("fast-gpio set 11 1");
     system("fast-gpio set 19 1");
     system("fast-gpio set 18 0");

     const char* filename = "lazorboi.log";
     ofstream outfile;
     outfile.open(filename);
     if(!outfile.is_open()) {
          return -1;
     }

     string input;
     int delimIndex = 0;
     int start = 0;
     int end;
     int dirA = DIR_CW;
     int dirB = DIR_CW;

     cout << "Welcome to" << endl;
     cout << ".__                                        ___.
_______    .__ " << endl;
     cout << "|  |  _____   ________  ____  _______ \\_ |__  \\
_  \\  |__|" << endl;
     cout << "|  |  \\__  \\  \\___   / /  _ \\ \\_  __ \\ | __
\\ /  /_\\  \\ |  |" << endl;
     cout << "|  |__ / __ \\_ /    / (  <_> ) |  | \\/ | \\_\\
\\\\  \\_/   \\|  |" << endl;
     cout << "|____/(____  //_____ \\ \\____/  |__|    |___  /
\\_____  /|__|" << endl;
     cout << "           \\/       \\/                    \\/
\\/     " << endl;
     cout << "Type 'help' for usage information." << endl;

     // Command-line tool
     while (true) {
          cout << endl;
          cout << "> ";
          getline(cin, input);
```

```
          delimIndex = input.find(delim);
          string command = input.substr(0, delimIndex);
          string arg;
          string arg2;
          if (command.length() == input.length()) {
               arg = "\0";
               arg2 = "\0";
          } else {
               string args = input.erase(0, delimIndex +
delim.length());
               string *iargs = getArgs(args, delim);
               arg = iargs[0];
               arg2 = iargs[1];
          }


          // HELP
          if (command == "help") {
               cout << "[+] run <motor_number> // Starts motor
number indicated" << endl;
               cout << "[+] reverse <motor_number> // Reverses
motor number indicated" << endl;
               cout << "[+] speed <motor_number> <speed> // Runs
motor at a speed between 0 and 100" << endl;
               cout << "[+] string // Initiates a request to
input a string to be visualized";
               cout << "[+] stop <motor_number> // Stops motor
number indicated" << endl;
               cout << "[+] quit // Quits program and stops
motors if running" << endl;
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
               outfile << "[" << timeinfo->tm_hour << ":" <<
timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Called
'help' command" << '\n';
          }

          // RUN
          else if (command == "run") {
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
```

```
               if (arg == "1") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Ran
motor 1" << '\n';
                    system("fast-gpio set 2 1");
               } else if (arg == "2") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Ran
motor 2" << '\n';
                    system("fast-gpio set 3 1");
               } else if (arg == "all") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Ran
all motors" << '\n';
                    system("fast-gpio set 2 1");
                    system("fast-gpio set 3 1");
               } else {
                    cout << "Usage: run <motor_number>" << endl;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'run'" << '\n';
               }
          }

          // REVERSE
          else if (command == "reverse") {
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
               if (arg == "1") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Reversed motor 1" << '\n';
                    if (dirB == 0) {
                         system("fast-gpio set 0 1");
                         system("fast-gpio set 1 0");
                         dirB = 1;
                    } else {
                         system("fast-gpio set 0 0");
                         system("fast-gpio set 1 1");
                         dirB = 0;
                    }
               } else if (arg == "2") {
```

```
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Reversed motor 2" << '\n';
                    if (dirA == 0) {
                         system("fast-gpio set 19 0");
                         system("fast-gpio set 18 1");
                         dirA = 1;
                    } else {
                         system("fast-gpio set 19 1");
                         system("fast-gpio set 18 0");
                         dirA = 0;
                    }
               } else if (arg == "all") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Reversed all motors" << '\n';
                    if (dirB == 0) {
                         system("fast-gpio set 0 1");
                         system("fast-gpio set 1 0");
                         dirB = 1;
                    } else {
                         system("fast-gpio set 0 0");
                         system("fast-gpio set 1 1");
                         dirB = 0;
                    }
                    if (dirA == 0) {
                         system("fast-gpio set 19 0");
                         system("fast-gpio set 18 1");
                         dirA = 1;
                    } else {
                         system("fast-gpio set 19 1");
                         system("fast-gpio set 18 0");
                         dirA = 0;
                    }
               } else {
                    cout << "Usage: reverse <motor_number>" <<
endl;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'reverse'" << '\n';
               }
          }

          // SPEED
          else if (command == "speed") {
```

```
                time_t rawtime;
                struct tm * timeinfo;
                time (&rawtime);
                timeinfo = localtime (&rawtime);

                string strArg;
                stringstream convert;
                convert << arg2;
                strArg = convert.str();

                if (arg == "1") {
                     outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Set
speed of motor 1 to " << arg2 << '\n';
                     string pwm_command = "fast-gpio pwm 2 6000 "
+ strArg;
                     const char *c_command = pwm_command.c_str();
                     system(c_command);
                } else if (arg == "2") {
                     outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Set
speed of motor 2 to " << arg2 << '\n';
                     string pwm_command = "fast-gpio pwm 3 6000 "
+ arg2;
                     const char *c_command = pwm_command.c_str();
                     system(c_command);
                } else if (arg == "all") {
                     outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Set
all motor speeds to " << arg2 << '\n';
                     string pwm_command1 = "fast-gpio pwm 2 6000
" + arg2;
                     string pwm_command2 = "fast-gpio pwm 3 6000
" + arg2;
                     const char *c_command1 =
pwm_command1.c_str();
                     const char *c_command2 =
pwm_command2.c_str();
                     system(c_command1);
                     system(c_command2);
                } else {
                     cout << "Usage: speed <motor_number>
<speed>" << endl;
```

```
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'speed'" << '\n';
               }
          }

          // STOP
          else if (command == "stop") {
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
               if (arg == "1") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Stopped motor 1" << '\n';
                    system("fast-gpio set 2 0");
               } else if (arg == "2") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Stopped motor 2" << '\n';
                    system("fast-gpio set 3 0");
               } else if (arg == "all") {
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Stopped all motors" << '\n';
                    system("fast-gpio set 2 0");
                    system("fast-gpio set 3 0");
               } else {
                    cout << "Usage: stop <motor_number>" <<
endl;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'stop'" << '\n';
               }
          }

          // STRING
          else if (command == "string") {
               cout << "Please input a string: ";
               string userIn;
               getline(cin, userIn);
               char* charArray = stoch(userIn);
               while (!isValid(charArray)) {
                    time_t rawtime;
```

```
                    struct tm * timeinfo;
                    time (&rawtime);
                    timeinfo = localtime (&rawtime);
                    cout << "String entered is invalid. Make
sure it only includes alphabetical characters and spaces." <<
endl;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'string'" << '\n';
                    getline(cin, userIn);
                    charArray = stoch(userIn);
                    continue;
               }
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
               outfile << "[" << timeinfo->tm_hour << ":" <<
timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Started
'string' command motor cycle" << '\n';
               int index = 0;
               int speedA;
               int speedB;
               string strArgA;
               stringstream convertA;
               string strArgB;
               stringstream convertB;
               while (charArray[index + 1] != '\0') {
                    time_t rawtime;
                    struct tm * timeinfo;
                    time (&rawtime);
                    timeinfo = localtime (&rawtime);
                    dirA = getDir(charArray[index]);
                    speedA = getSpeed(charArray[index]);
                    dirB = getDir(charArray[index + 1]);
                    speedB = getSpeed(charArray[index + 1]);
                    convertA << speedA;
                    strArgA = convertA.str();
                    convertB << speedB;
                    strArgB = convertB.str();
                    if (dirA == 0) {
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically reversed motor 2" << '\n';
                         system("fast-gpio set 0 1");
```

```
                        system("fast-gpio set 1 0");
                        dirA = 1;
                    } else {
                        outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically reversed motor 2" << '\n';
                        system("fast-gpio set 0 0");
                        system("fast-gpio set 1 1");
                        dirA = 0;
                    }
                    if (dirB == 0) {
                        outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically reversed motor 1" << '\n';
                        system("fast-gpio set 19 0");
                        system("fast-gpio set 18 1");
                        dirB = 1;
                    } else {
                        outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically reversed motor 1" << '\n';
                        system("fast-gpio set 19 1");
                        system("fast-gpio set 18 0");
                        dirB = 0;
                    }
                    string pwm_commandA = "fast-gpio pwm 3 6000
" + strArgA;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically set speed of motor 2 to " << speedA <<
'\n';
                    const char *c_commandA =
pwm_commandA.c_str();
                    system(c_commandA);
                    string pwm_commandB = "fast-gpio pwm 2 6000
" + strArgB;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'string' automatically set speed of motor 1 to " << speedB <<
'\n';
                    const char *c_commandB =
pwm_commandB.c_str();
                    system(c_commandB);
                    usleep(2000000);
                    index++;
```

```
                }
            }

            // SINUSOID
            else if (command == "sinusoid") {
                outfile << "[" << timeinfo->tm_hour << ":" <<
timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Started
'sinusoid' command motor cycle" << '\n';
                string amp1str;
                string amp2str;
                int amp1;
                int amp2;

                int index = 0;
                int speedA;
                int speedB;
                string strArgA;
                stringstream convertA;
                string strArgB;
                stringstream convertB;

                do {
                    cout << "Please input an integer amplitude:
";
                    cin >> amp1str;
                    amp1 = stoint(amp1str);

                    if (amp1 == -1) {
                        time_t rawtime;
                        struct tm * timeinfo;
                        time (&rawtime);
                        timeinfo = localtime (&rawtime);
                        cout << "Invalid amplitude. Make sure
its an integer and only includes digits." << endl;
                        outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'sinusoid'" << '\n';
                    }
                } while (amp1 == -1);

                do {
                    cout << "Please input another integer
amplitude: ";
                    cin >> amp2str;
                    amp2 = stoint(amp2str);
```

```
                    time_t rawtime;
                    struct tm * timeinfo;
                    time (&rawtime);
                    timeinfo = localtime (&rawtime);
                    if (amp2 == -1) {
                         cout << "Invalid amplitude. Make sure
its an integer and only includes digits.";
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"Incorrect usage of command 'sinusoid'" << '\n';
                    }
               } while (amp2 == -1);

               for (int i = 0; i < 100; i++) {
                    time_t rawtime;
                    struct tm * timeinfo;
                    time (&rawtime);
                    timeinfo = localtime (&rawtime);
                    speedA = amp1 * sin(0.1 * i);
                    speedB = -amp2 * cos(0.2 * i);
                    if (speedA < 0) {
                         speedA = abs(speedA);
                         dirA = 0;
                    }
                    else
                         dirA = 1;
                    if (speedB < 0) {
                         speedB = abs(speedB);
                         dirB = 1;
                    }
                    else
                         dirB = 0;

                    speedA = map(speedA, 0, amp1, 10, 100);
                    speedB = map(speedB, 0, amp2, 10, 100);

                    convertA << speedA;
                    strArgA = convertA.str();
                    convertB << speedB;
                    strArgB = convertB.str();

                    if (dirA == 0) {
                         system("fast-gpio set 0 1");
                         system("fast-gpio set 1 0");
```

```
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically reversed motor 2" << '\n';
                         dirA = 1;
                    } else {
                         system("fast-gpio set 0 0");
                         system("fast-gpio set 1 1");
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically reversed motor 2" << '\n';
                         dirA = 0;
                    }
                    if (dirB == 0) {
                         system("fast-gpio set 19 0");
                         system("fast-gpio set 18 1");
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically reversed motor 1" << '\n';
                         dirB = 1;
                    } else {
                         system("fast-gpio set 19 1");
                         system("fast-gpio set 18 0");
                         outfile << "[" << timeinfo->tm_hour <<
":" << timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically reversed motor 1" << '\n';
                         dirB = 0;
                    }

                    string pwm_commandA = "fast-gpio pwm 3 14000
" + strArgA;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically set speed of motor 2 to " << speedA <<
'\n';
                    const char *c_commandA =
pwm_commandA.c_str();
                    system(c_commandA);
                    string pwm_commandB = "fast-gpio pwm 2 14000
" + strArgB;
                    outfile << "[" << timeinfo->tm_hour << ":"
<< timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " <<
"'sinusoid' automatically set speed of motor 2 to " << speedB <<
'\n';
                    const char *c_commandB =
pwm_commandB.c_str();
```

```
                    system(c_commandB);
                    usleep(1000000);

                    system("fast-gpio pwm 2 14000 0");
                    system("fast-gpio pwm 3 14000 0");
                    usleep(500000);
               }
          }

          // QUIT
          else if (command == "quit") {
               time_t rawtime;
               struct tm * timeinfo;
               time (&rawtime);
               timeinfo = localtime (&rawtime);
               outfile << "[" << timeinfo->tm_hour << ":" <<
timeinfo->tm_min << ":" << timeinfo->tm_sec << "] " << "Quit
program" << '\n';
               system("fast-gpio set 2 0");
               system("fast-gpio set 3 0");
               break;
          }
     }
     outfile.close()
}
```

2. lazorboi.cpp

```
#include <cmath>

bool isValid(string input) {
     for (int i = 0; i < input.length(); i++) {
          char temp = toupper(input[i]);
          if (!(temp >= 'A' && temp <= 'Z') ^ (temp == ' '))
               return false;
     }
     return true;
}

int getDir(char input) {
     int value = (int) input;
     int count = 0;
     while(value) {
          count += value & 1;
          value >>= 1;
     }

     int dir = count % 2;
     return dir;
}

int getSpeed(char input) {
     int value = toupper(input);
     int posInAlphabet = abs(value - 64);

     int speed = (int) ((value * posInAlphabet) / (3.14159 *
26));

     int lowerBound = (int) ((65 * 1) / (3.14159 * 26));
     int upperBound = (int) ((90 * 26) / (3.14159 * 26));

     speed = map(speed, lowerBound, upperBound, 20, 95);
     return speed;
}
```

3. Utils.cpp

```
int map(int value, int preLowerBound, int preUpperbound, int
postLowerBound, int postUpperbound){
     return (value - preLowerBound) * (postUpperbound -
postLowerBound) / (preUpperbound - preLowerBound) +
postLowerBound;
}

string* getArgs(string input, string delim) {
     int delimIndex = 0;
     string *args = new string [2];
     delimIndex = input.find(delim);
     args[0] = input.substr(0, delimIndex);
     args[1] = input.erase(0, delimIndex + delim.length());
     return args;
}

char* stoch(string x) {
     char* array = new char[x.length() + 1];
     for (int i =0; i<x.length(); i++)
          array[i] = x[i];
     array[x.length()] = '\0';
     return array;
}

enum STATE {START, DIGIT, TERMINAL};
int stoint(string x){
     char* input = stoch(x);
     int output = 0;

     STATE state = START;
     char in;
     cout << endl;
     for (int i = 0; state != TERMINAL; i++) {
          in = input[i];

          switch (state) {
          case START:
               if (in >= '0' && in <= '9') {
                    output = (int) (in - '0');
```

```
                    state = DIGIT;
               }
               else
                    return -1;
               break;
          case DIGIT:
               if (in >= '0' && in <= '9')
                    output = output * 10 + (int) (in - '0');
               else if (in == '\0')
                    return output;
               else
                    return -1;
               break;
          }
     }
}
```