Autonomous surf life saving device

Jarod Lam
Supervisor: Matthew Dunbabin

2018-2019 VRES project at Queensland University of Technology

***Abstract*—build boat that saves people**

CONTENTS

I. INTRODUCTION

Surf life savers regularly patrol beaches to help those in danger and are essential to keeping public beaches safe. In Queensland alone, over three thousand are rescued and hundreds are resuscitated by lifesaving services every year. [1] Even so, from 2008-2018, there was an average of 47 drowning deaths per year at Australian beaches–a tragically high number that many organisations are working to reduce. [2]

To supplement the activities of surf life savers and other services at public beaches, a system has been proposed that will allow timely help to be given to people in distress while waiting to be rescued. The surf rescue boat (SRB) aims to deliver help quickly and reduce the risk to which life savers are exposed.

The basic concept of the design is a remotely-controlled floating water vehicle that sits in the surf away from the shore at a beach. At most times, the vehicle is idle and remains stationary in the surf. When a person in distress is seen, a life saver on the shore can remotely direct the vehicle to the person to support them while they wait for help.

A simple water-based robot such as the one proposed can be constructed relatively cheaply and easily with off-the-shelf components. In the future, systems such as these may become widely available and save the lives of many along coastal beaches.

The system is divided into three main sections: the remotely operated water vehicle (the ”boat”), an XBee-based radio communications system between the boat and a base station, and a computer vision-based control system. Out of these, only the vehicle and communications were prototyped in this

project; the control system has been developed separately in the past and time constraints prevented it from being implemented.

Design considerations taken into account include cost of construction, parts, and maintenance; usability and user-friendliness; and effectiveness as a water-based vehicle.

This report describes these systems in detail, the design and testing methodology, and avenues that can be explored for future development of the surf rescue boat.

II. Boat

A. Mechanical

The remotely operated boat uses a standard surfboard as a base, and houses electronics in a watertight hard plastic case attached to the top. Two propellers are mounted to the bottom of the surfboard for movement control.

1) Chassis: Surfboard. Chosen for its stability and familiarity in the surf. The standardness and availability of surfboards is an advantage to encouraging development of such systems. A custom-built chassis may have been designed, but would have taken more time and money.

2) Propellors: 2 Blue Robotics T100 Thrusters. A propeller is mounted each to the left and right of the surfboard's middle underside. Aluminium mounting plates, attached to the board with Sikaflex, were designed to distribute force and allow propellors to be detached easily.

3) Electronics housing: Pelican 1120 Case. A laser-cut acrylic frame was made to mount the electronics in the box. Wire glands on the side of the box allow propellor wires to be fed through the box walls.

B. Electronics

An Arduino Mega 2560 controls the onboard electronics mounted in the case. GPS and IMU modules are used for navigation, and an XBee radio communicates with the base station. Two electronics speed controllers (ESCs) control each of the two propellers. A block connection diagram of the electronics is shown in Figure 1.

1) Microcontroller: Arduino Mega 2560 with Seeedstudio Grove Mega Shield breakout board. This development board is powerful enough to handle relatively simple communication and processing tasks required to control the boat's sensors and motors. More powerful ARM-based boards such as the Raspberry Pi are less suited to rugged environments, and more difficult to recover from failures. The shield provides robust headers to the UART functions of the Arduino.

2) Radio: Digi XBee Pro S1 on a SparkFun XBee Explorer Regulated. This connects to the Arduino via UART, and creates a wireless serial connection to the base station. The protocol is defined in section 3.

3) GPS: LOCOSYS LS20031 GPS receiver. Sends data to the Arduino via UART using the NMEA 0183 protocol found in section 3.

4) IMU: Sparkfun SEN-10736 9DOF Razor IMU. Sends data to the Arduino via UART. Currently, only the compass value from the sensor is used. Flashed with the Razor AHRS firmware: https://github.com/Razor-AHRS/razor-9dof-ahrs

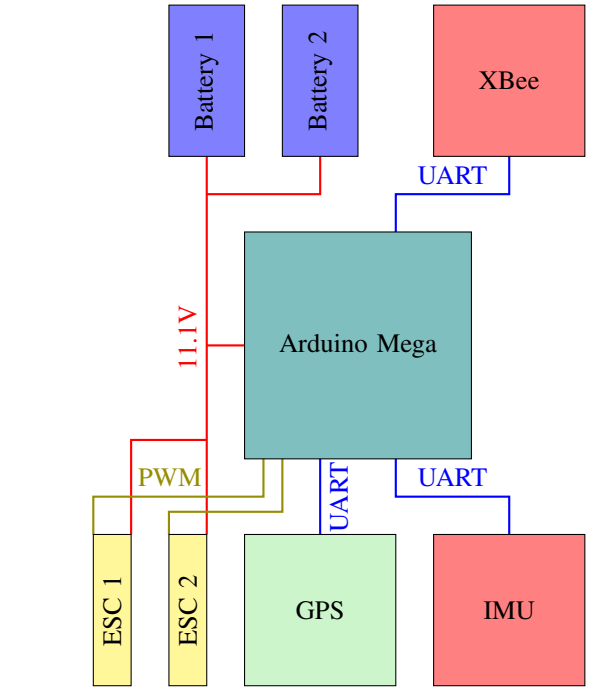


Fig. 1. Connection block diagram for onboard electronics.

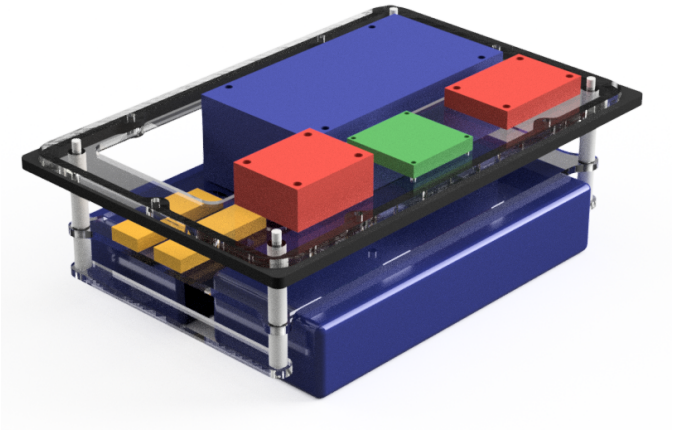


Fig. 2. Rendering of internal electronics frame

5) Motor control: 2 Flycolor Raptor 390 30A ESC. Firmware modified to allow forward and backward propeller movement. Receives PWM control signals from the Arduino with a duty cycle range of 1000 µs to 2000 µs.

6) Battery: 2 Zippy Flightmax Z58003S-30 5800 mAh 3S1P. The two batteries are wired in parallel, with a combined nominal capacity of 11.6 Ah and nominal voltage of 11.1 V.

C. Software

The Arduino Mega 2560 is programmed in C++ on top of Arduino default and custom libraries. Efforts were made to keep the code somewhat portable and reusable.

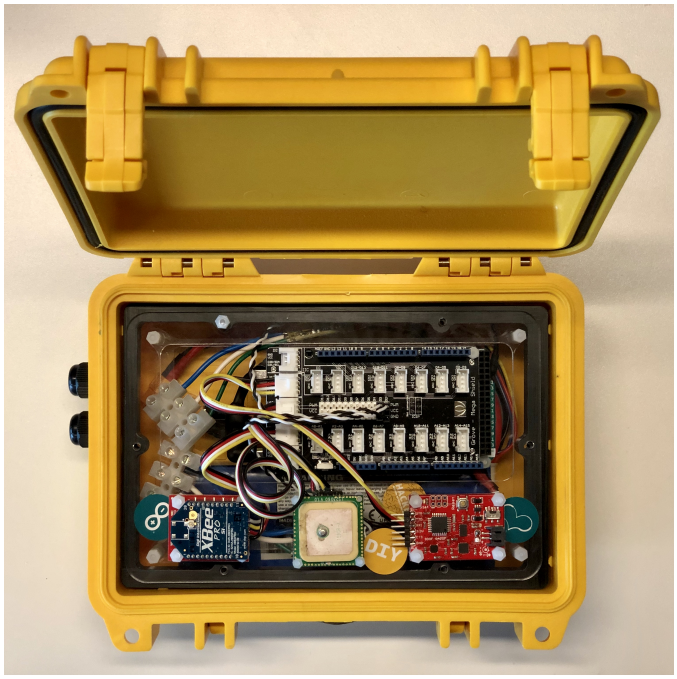


Fig. 3. Boat electronics in Pelican case housing

The code is split up into several modules, each handling a section of robot operation. These are described below, and the full code can be found in appendix A.

1) `srb`: Contains the main loop of the program. Initialises values, classes, etc. Runs update functions for GPS, comms, IMU, nav, and motors. Sends `SRBSM` message at intervals. An AVR watchdog timer is set to reset the microcontroller at the hardware level if the program hangs and reaches a timeout.

2) `nmea`: Defines the `Nmea` class, which contains functions for constructing and parsing NMEA 0183 sentences. This includes generating and validating checksums, counting the number of fields, appending strings and decimal numbers to a sentence, and parsing a sentence by field. All functions use standard C string libraries, so they do not rely on Arduino libraries and will work outside the Arduino environment. Used by `SrbGps` and `SrbComms`.

3) `srb_stats`: Defines the `SrbStats` class, which stores the ID, state, GPS and target location, compass and target heading, and other information related to the current state and navigation of the boat. A pointer to the same instance of this class is passed to most other classes when they are created so that they can read and update this information.

4) `srb_serial`: Defines the `SrbSerial` class, which buffers a hardware serial stream and parses the input when a newline is received. The serial port used is configured when the object is created. This is the base class for `SrbGps`, `SrbComms`, and `SrbImu`.

5) `srb_gps`: Defines the `SrbGps` class, which receives and parses GPS fix data over serial. Latitude, longitude, magnetic variation, and ground speed are parsed from the NMEA `GPRMC` sentence and stored in the `SrbStats` object. Conversions are made from knots to metres per second, and degrees/minutes to decimal degrees.

6) `srb_comms`: Defines the `SrbComms` class, which sends and receives messages to and from the base station via the XBee radio. Contains functions for constructing and parsing the proprietary NMEA sentences defined in section 3. Stores information and instructions received in the `SrbStats` object. Stops the boat if no message is received within a timeout period.

7) `srb_imu`: Defines the `SrbImu` class, which receives data from the Sparkfun IMU over serial. Extracts the compass heading from the serial stream and stores it in the `SrbStats` object.

8) `srb_motor`: Defines the `SrbMotor` class, which controls motor movement. Receives motor power ranges from -100 to 100 and sets the corresponding PWM duty cycle. Accelerates motors to the desired speed at a safe pace.

9) `srb_nav`: Defines the `SrbNav` class, which controls robot navigation. In manual mode, sets motor speed and orients the boat according to target speed and heading sent from the base station. In auto mode, moves the boat toward a set of coordinates sent from the base station. Motors are controlled with the `SrbMotor` class.

III. Communications

Communications between the SRB and the base station are achieved using XBee radios. By attaching a pair of XBee modules to the base station computer and the on-board Arduino, a virtual serial connection is created between the two devices.

A. NMEA 0183 sentences

NMEA 0183 is a communications specification designed to create a standardised serial interface for GPS devices. Every NMEA 'sentence' begins with a `$` and ends with `*CS\r\n`, where `CS` is a two-digit hexadecimal checksum of the sentence. Some advantages of using NMEA sentences are that they are standardised, human-readable, robust, and relatively simple to implement.

A common NMEA sentence type is `GPRMC`, the GPS recommended minimum. This sentence is used to receive information from the onboard GPS module. `GPRMC` sentences are specified as follows: [3]

```
$GPRMC,<Time>,<Status>,<Lat>,<LatDir>,
<Lon>,<LonDir>,<Speed>,<Angle>,<Date>,
<MagVar>,<MagDir>*CS
```

Where:

`<Time>`	UTC timestamp in HHmmss format
`<Status>`	Status `A`=active, `V`=void
`<Lat>`	Latitude in ddmm.mmm format
`<LatDir>`	`N` or `S` hemisphere
`<Lon>`	Longitude in dddmm.mmm format
`<LonDir>`	`E` or `W` hemisphere
`<Speed>`	Ground speed in knots
`<Angle>`	Track angle in degrees from north
`<Date>`	Date in DDMMYY format
`<MagVar>`	Magnetic variation magnitude
`<MagDir>`	Magnetic variation direction

A NMEA sentence parser and constructor was written in C++ and Python for the boat and the base station, respectively. Specified below is a set of custom NMEA sentence types that was created for communication between the boat and the base station over the XBee radios.

1) `SRBSM` - Status Message: The `SRBSM` sentence is sent periodically by the boat to update the base station with status information.

```
$SRBSM,<ID>,<State>,<Lat>,<Lon>,<Speed>,
<Heading>,<BattV>,<FwdPower>,
<TgtHeading>*CS
```

Where:

`<ID>`	ID of target SRB
`<State>`	`0`=disabled, `1`=manual, `2`=auto
`<Lat>`	Latitude in decimal degrees
`<Lon>`	Longitude in decimal degrees
`<Speed>`	Speed in metres per second
`<Heading>`	Compass heading in deg CW from N
`<BattV>`	Current battery voltage
`<FwdPower>`	Forward power from -100 to 100
`<TgtLat>`	Target latitude in decimal degrees
`<TgtLon>`	Target longitude in decimal degrees
`<TgtHeading>`	Target heading in deg CW from N

2) `SRBJS` - Joystick: The `SRBJS` sentence is sent by the base station for manual control of the boat.

```
$SRBJS,<ID>,<FwdPower>,<TgtHeading>*CS
```

Where:

`<ID>`	ID of target SRB
`<FwdPower>`	Forward power from -100 to 100
`<TgtHeading>`	Target heading in deg. CW from N

3) `SRBWP` - Waypoint: The `SRBWP` sentence is sent by the base station to autonomously direct the boat to a set of coordinates.

```
$SRBJS,<ID>,<TgtLat>,<TgtLon>*CS
```

Where:

`<ID>`	ID of target SRB
`<TgtLat>`	Target latitude in decimal degrees
`<TgtLon>`	Target longitude in decimal degrees

B. Hardware

The base station uses an XBee S1 radio on a Sparkfun XBee Explorer Regulated to communicate with the boat. An FTDI232 breakout board allows the computer's USB port to interface with the XBee's UART. This system is shown in Figure 4.

C. Software

The Python program `srb-base.py` was developed to provide a rudimentary graphical interface for the wireless serial connection over XBee. A screenshot of this program is shown in Figure 5.

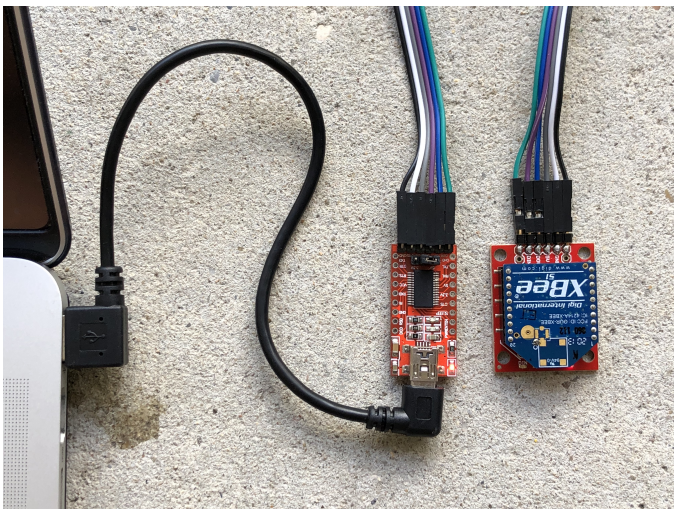


Fig. 4. Communications hardware

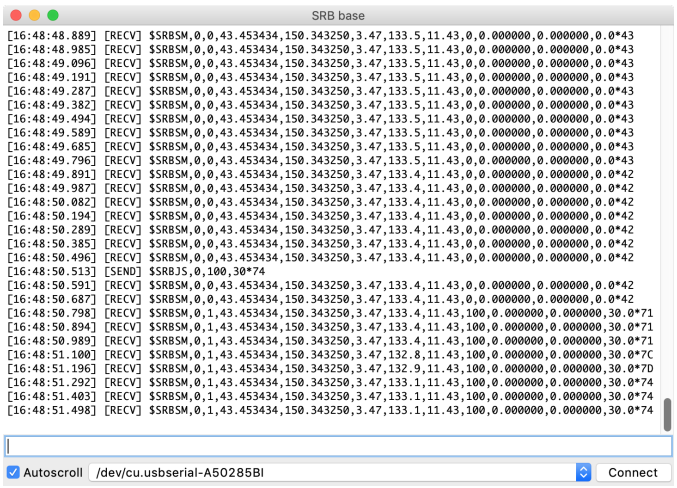


Fig. 5. Screenshot of `srb-base.py`

The Tkinter interface was modelled on the Arduino IDE's Serial Monitor, but is designed to send NMEA sentences instead of arbitrary text over the serial connection.

"Naked" messages without a leading `$` or trailing `*CS` have these added when they are sent, making it easy to send valid sentences. Incoming messages are validated to ensure the correct syntax and checksum, and invalid sentences are shown in red. The NMEA sentence handler class, `Nmea`, is separated into the file `nmea.py`.

All incoming and outgoing messages are logged to the GUI window, terminal window, and a log file in the working directory. The log file includes a comma-separated timestamp at the beginning of each line, allowing the log data to be processed in a makeshift way as a spreadsheet.

The source code for `srb-base.py` and `nmea.py` can be found in Appendix A. However, this code is unstable and should not be used as-is, as it was written for testing purposes only.

IV. Testing

Testing was performed on the power consumption of the boat in a lab setting, as well as the forward speed and turning speed of the boat under typical calm conditions.

A. Power consumption

1) Method: The boat was elevated in midair on two stands and oriented to face 0 degrees North according to the onboard magnetometers. Two multimeters were connected to the battery, one in parallel measuring the voltage under load, and one in series measuring current.

Starting from an idle state, the message

`$SRBJS,0,<FwdPower>,0*<CS>`

was sent from the base station to the boat, with `<FwdPower>` as the throttle value being tested. Voltage and current values were recorded once they stabilised. Then, the message

`$SRBJS,0,0,0*46`

was sent from the base station to halt the motors. This was repeated for the throttle levels 0 (idle), 20, 40, 60, 80, and 100.

Throttle (%)	Voltage (V)	Current (A)
0		
20		
40		
60		
80		
100		

TABLE I
Power consumption test results

Throttle (%)	Power (W)	11.6 Ah runtime (min)
0		
20		
40		
60		
80		
100		

TABLE II
Power consumption analysis

2) Analysis and discussion:

B. Forward speed

1) Method: The boat was placed in the Brisbane River at a clear, calm location and oriented along the shoreline. A rope was attached to aid retrieval.

Starting from an idle state, the message

`$SRBJS,0,<FwdPower>,<TgtHeading>*<CS>`

was sent from the base station to the boat, with `<FwdPower>` as the throttle value being tested and `<TgtHeading>` as the boat's current orientation. After the boat travelled roughly 20 metres, the message

`$SRBJS,0,0,0*46`

was sent back from the base station to halt the motors. The boat was moved back to its starting position. This process was repeated twice for each throttle level being tested.

Messages from the boat were captured into a file using `srb-base.py`. All messages except for `SRBSM` were removed by running the terminal command:

```
$grep "SRBSM" [logfilename] > results.csv
```

which saved the results in a `.csv` file.

Throttle (%)	Accel time (s)			Max speed ($\mathrm{m\,s^{-1}}$)		
	1	2	avg.	1	2	avg.
25						
50						
75						
100						

TABLE III
Forward speed test results

Fig. 6. Forward speed vs. time

2) Analysis and discussion:

V. Future development

A. Boat

B. Communications

C. Computer vision

VI. Conclusion

References

[1] Surf Live Saving Queensland, "Statistics," accessed February 2019. [Online]. Available: http://lifesaving.com.au/about/statistics/

[2] Royal Life Saving Australia, "National drowning report 2018," accessed February 2019. [Online]. Available: https://www.royallifesaving.com.au/__data/assets/pdf_file/0004/23197/RLS_NDR2018_ReportLR.pdf

[3] D. DePriest, "Nmea data," accessed November 2018. [Online]. Available: https://www.gpsinformation.org/dale/nmea.htm

APPENDIX A
CODE

A. srb/srb.ino

```
/*
   srb.ino
   Surf rescue boat control system
   Written by Jarod Lam
*/

#include <avr/wdt.h>
#include "srb_stats.h"
#include "srb_motor.h"
#include "srb_nav.h"
#include "srb_comms.h"
#include "srb_gps.h"
#include "srb_imu.h"

#define LOOP_DELAY 100
#define USE_WATCHDOG

unsigned long prevMillis = 0;
int motorPins[] = {2, 3};
int motorSides[] = {LEFT, RIGHT};

SrbStats stats;
SrbMotor motors;
SrbNav   nav(&stats, &motors);
SrbComms comms(&stats, &Serial1);
SrbGps   gps(&stats, &Serial2);
SrbImu   imu(&stats, &Serial3);

void setup() {

  // Start watchdog
  #ifdef USE_WATCHDOG
  wdt_enable(WDTO_2S);
  #endif

  // Start USB serial
  Serial.begin(9600);

  // Initialise motors
  motors.begin(motorPins, motorSides);

  // Set comms failsafe timeout
  comms.setTimeout(-1);

  // Initialise stats
  stats.ID = 0;
  stats.state = 0;
  stats.lat = 43.4534324;
  stats.lon = 150.3432493;
  stats.speed = 3.4749;
  stats.heading = 174.3;
  stats.battV = 11.434;
  stats.forwardPower = 0;
  stats.targetHeading = 0;

}

void loop() {

  // Reset watchdog timer
  #ifdef USE_WATCHDOG
  wdt_reset();
  #endif

  // Call update functions for everything!
  gps.update();       // Check GPS serial
  comms.update();     // Check XBee serial
  imu.update();       // Check IMU serial
  nav.update();       // Calculate nav based on new
      ↪ position/target
  motors.update();    // Accelerate motors to speeds set
      ↪ by nav

  // Non-blocking loop delay
  if (millis() >= prevMillis+LOOP_DELAY) {
    prevMillis = millis();

    // Send status message
    comms.sendSRBSM();
    Serial.println(stats.heading);

  }

}
```

B. srb/nmea.h

```
/*
   nmea.h
   Library for manipulating NMEA-0183 strings
   Written by Jarod Lam
*/

#ifndef nmea_h
#define nmea_h

#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

#define NMEA_BUFFER_SIZE 1024

class Nmea {
  public:
    Nmea();

    /*
       Returns the sentence.
    */
    const char *read();

    /*
       Clear and set the sentence.
    */
    void write(const char *s);

    /*
       Initialise the sentence with a dollar sign.
    */
    void begin();

    /*
       Append a field to the sentence.
    */
    void append(const char *s);
    void appendInt(int d);
    void appendFloat(float d, int places);

    /*
       Calculate the checksum and append to the sentence
           ↪ .
    */
    void appendChecksum();

    /*
       Checks if an NMEA sentence is valid.

       Needs to:
         - Start with a $
         - End with * and checksum
         - Correct checksum
    */
    int validate();

    /*
       Returns the next argument of the sentence.

       Removes the argument from the string once read.
       Returns NULL if end of the string is reached.
    */
    const char *nextField();

    /*
       Returns the number of fields in the sentence.
    */
    int numFields();

  private:
    /*
       Character buffer working space
    */
    char _buffer[NMEA_BUFFER_SIZE];

    /*
```

```
78          Where the sentence is stored
79      */
80      char _sentence[NMEA_BUFFER_SIZE];
81
82      /*
83          Generate the NMEA checksum.
84
85          Automatically skips $ and terminates at *.
86      */
87      unsigned char generateChecksum(const char *s);
88
89  };
90
91  #endif
```

C. srb/nmea.cpp

```
1   /*
2    * nmea.cpp
3    * Library for manipulating NMEA-0183 strings
4    * Written by Jarod Lam
5    */
6
7   #include "nmea.h"
8
9   Nmea::Nmea(void) {
10    memset(&_buffer, 0, sizeof(_buffer));
11    _sentence[0] = '$';
12  }
13
14  const char *Nmea::read() {
15    return _sentence;
16  }
17
18  void Nmea::write(const char *s) {
19    memset(_sentence, 0, sizeof(_sentence));
20    strcpy(_sentence, s);
21  }
22
23  void Nmea::begin() {
24    memset(_sentence, 0, sizeof(_sentence));
25    _sentence[0] = '$';
26  }
27
28  void Nmea::append(const char *s) {
29    int fieldLen = strlen(s);
30    int sentLen = strlen(_sentence);
31
32    // Abort if the sentence will become too long
33    if ((fieldLen + sentLen) > (NMEA_BUFFER_SIZE + 5))
          ↪ return;
34
35    // Add a comma if necessary
36    if (_sentence[sentLen-1] != '$') {
37      strcat(_sentence, ",");
38    }
39
40    // Concatenate
41    strcat(_sentence, s);
42  }
43
44  void Nmea::appendInt(int d) {
45    char s[16];
46    snprintf(s, 16, "%d", d);
47    //dtostrf(d, 0, 0, s);
48    append(s);
49  }
50
51  void Nmea::appendFloat(float d, int places) {
52    char s[16];
53    dtostrf(d, 0, places, s);
54    append(s);
55  }
56
57  void Nmea::appendChecksum() {
58    char cs_string[2];
59    sprintf(cs_string, "%.2X", generateChecksum(_sentence)
          ↪ );
60    strcat(_sentence, "*");
61    strcat(_sentence, cs_string);
62  }
63
64  int Nmea::validate() {
65    int sentLen = strlen(_sentence);
66
67    // Check if it starts with a dollar sign
68    if (_sentence[0] != '$') return 0;
69
70    // Check if it ends with an asterisk
71    if (_sentence[sentLen-3] != '*') return 0;
72
73    // Check the checksum
74    char cs_recv = strtol(&_sentence[sentLen-2], NULL, 16)
          ↪ ;
75    char cs_calc = generateChecksum(_sentence);
76    if (cs_recv != cs_calc) return 0;
77
78    return 1;
79  }
80
81  const char *Nmea::nextField() {
82    int startInd = 0;
83    int endInd = 0;
84    int len = strlen(_sentence);
85
86    // Copy the string into the buffer
87    if (len > NMEA_BUFFER_SIZE) return 0;
88    memset(&_buffer, 0, sizeof(_buffer));
89    strcpy(_buffer, _sentence);
90
91    // Check for end of sentence
92    if (_sentence[0] == '*') return 0;
93
94    for (int i = 0; i < len; i++) {
95      // Check for dollar sign to start the string
96      if (_buffer[i] == '$') {
97        startInd = i + 1;
98      }
99      // Check for comma or asterisk to end the string
100     if ((_buffer[i] == ',') || (_buffer[i] == '*')) {
101       endInd = i;
102       break;
103     }
104   }
105   if (startInd > endInd) return 0;
106
107   // Truncate the string
108   strcpy(_sentence, &_buffer[endInd+1]);
109
110   // Return the field
111   if (_buffer[0] == ',') {
112     return 0;
113   } else {
114     char returnVal[len];
115     memset(returnVal, 0, len);
116     strncpy(returnVal, &_buffer[startInd], endInd-
          ↪ startInd);
117     strcpy(_buffer, returnVal);
118     return _buffer;
119   }
120 }
121
122 int Nmea::numFields() {
123   int count = 0;
124   for (int i = 0; i < strlen(_sentence); i++) {
125     if (_sentence[i] == ',') {
126       count++;
127     }
128   }
129   return count + 1;
130 }
131
132 unsigned char Nmea::generateChecksum(const char *s) {
133   unsigned char checksum = 0;
134
135   for (int i = 0; i < strlen(s); i++) {
136     char c = s[i];
137
138     if (c == '$') continue;  // Skip the character if
          ↪ dollar sign
139     if (c == '*') break;     // Stop the checksum if
          ↪ asterisk
140
141     checksum ^= c;  // XOR to checksum
142   }
143
144   return checksum;
145 }
```

D. srb/srb_stats.h

```
/*
   srb_stats.h
   Surf rescue boat control system
   Written by Jarod Lam
*/

#ifndef srb_h
#define srb_h

// Number of servos
#define NUM_MOTORS 2

// Left and right motor names
#define LEFT 0
#define RIGHT 1

// Class containing current stats of SRB
class SrbStats {

  public:

    /*
     * ID of the SRB, set at start of program
     */
    int ID;

    /*
     * State where 0=disabled, 1=manual, 2=auto
     */
    int state = 0;

    /*
     * Boolean GPS fix flag
     */
    int gpsFix = 0;

    /*
     * Current latitude and longitude in decimal degrees
     */
    float lat = 0;
    float lon = 0;

    /*
     * Current ground speed in m/s
     */
    float speed = 0;

    /*
     * Current compass heading in degrees from north
     */
    float heading = 0;

    /*
     * Current battery voltage
     */
    float battV = 0;

    /*
     * Forward power percentage (-100-100 manual, 0-100 auto)
     */
    int forwardPower = 0;

    /*
     * Target compass heading in degrees from north
     */
    float targetHeading = 0;

    /*
     * Target lat/lon in decimal degrees
     */
    float targetLat = 0;
    float targetLon = 0;

    /*
     * Magnetic variation (declination) in degrees
     */
    float magVar = 0;
};

#endif
```

E. srb/srb_serial.h

```
/*
   srb_serial.h
   General class for receiving data over serial
   Written by Jarod Lam
*/

#ifndef srb_serial_h
#define srb_serial_h

#define SERIAL_BUFFER_SIZE 1024
#define SERIAL_BUFFER_TIMEOUT 500

#include <Arduino.h>
#include "srb_stats.h"

class SrbSerial {

  public:

    /*
     * Initialise SrbImu with the IMU serial port.
     */
    explicit SrbSerial(SrbStats *stats, HardwareSerial *port, long baud);

    /*
     * Update the values from serial.
     */
    void update();

  protected:

    /*
     * Pointer to the XBee serial port object.
     */
    HardwareSerial *_serial;

    /*
     * Pointer to the stats object given at creation.
     */
    SrbStats *_stats;

    /*
     * Buffers for storing I/O data.
     */
    char _buffer[SERIAL_BUFFER_SIZE];
    unsigned long _bufferClearTime;
    void _clearBuffer();

    /*
     * Update buffer with new serial info.
     */
    void _updateSerial();

    /*
     * Parse the contents of the buffer after a carriage return.
     */
    virtual void _parseBuffer() {};

};

#endif
```

F. srb/srb_serial.cpp

```
/*
   srb_serial.h
   General class for receiving data over serial
   Written by Jarod Lam
*/

#include "srb_serial.h"

SrbSerial::SrbSerial(SrbStats *stats, HardwareSerial *port, long baud) {
  _serial = port;
  _serial->begin(baud);
  _stats = stats;
  _clearBuffer();
}
```

```
15
16 void SrbSerial::_updateSerial() {
17
18   while (_serial->available()) {
19
20     // Read a single character from serial
21     char c = _serial->read();
22
23     // Clear the buffer if line feed
24     if (c == '\n') {
25       _clearBuffer();
26     }
27
28     // Parse the sentence if carriage return
29     else if (c == '\r') {
30       _parseBuffer();
31       _clearBuffer();
32     }
33
34     // Filter out non printable characters
35     else if (!isPrintable(c)) {
36       _clearBuffer();
37     }
38
39     // Else, add character to the buffer
40     else {
41       if (strlen(_buffer) >= SERIAL_BUFFER_SIZE) {
42         _clearBuffer();
43       }
44       strncat(_buffer, &c, 1);
45     }
46   }
47
48   // Timeout and clear the buffer
49   if (millis() - _bufferClearTime >
       ↪ SERIAL_BUFFER_TIMEOUT) {
50     _clearBuffer();
51   }
52
53 }
54
55 void SrbSerial::_clearBuffer() {
56   memset(_buffer, 0, sizeof(*_buffer));
57   _bufferClearTime = millis();
58 }
```

G. srb/srb_gps.h

```
1  /*
2     nmea_gps.h
3     Receive and parse GPS NMEA strings
4     Written by Jarod Lam
5  */
6
7  #ifndef nmea_gps_h
8  #define nmea_gps_h
9
10 #include <Arduino.h>
11 #include <Time.h>
12 #include "nmea.h"
13 #include "srb_stats.h"
14 #include "srb_serial.h"
15
16 class SrbGps : public SrbSerial {
17   public:
18
19     /*
20      * Initialise SrbGps with the GPS serial port.
21      */
22     SrbGps(SrbStats *stats, HardwareSerial *port);
23
24     /*
25      * Update the values from serial.
26      */
27     void update();
28
29   private:
30
31     /*
32      * Parse the sentence in the buffer.
33      */
34     void _parseBuffer();
35
36     /*
37      * Convert NMEA lat/lon string to decimal degrees.
38      *
39      * "degLen" is number of degrees digits (2 for lat,
       ↪ 3 for lon).
40      */
41     float _degToDec(const char *s, int degLen);
42
43     /*
44      * Convert knots string to m/s.
45      */
46     float _knotsToMps(const char *s);
47
48     /*
49      * Milliseconds since the buffer was last cleared.
50      */
51     unsigned long _bufferClearTime;
52 };
53
54 #endif
```

H. srb/srb_gps.cpp

```
1  /*
2     nmea_gps.cpp
3     Receive and parse GPS NMEA strings
4     Written by Jarod Lam
5  */
6
7  #include "srb_gps.h"
8
9  SrbGps::SrbGps(SrbStats *stats, HardwareSerial *port) :
       ↪ SrbSerial::SrbSerial(stats, port, 57600) {
10
11   // Configure to send RMC commands only
12   _serial->print("$PMTK314
         ↪ ,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n"
         ↪ );
13   // Configure to send every 200ms
14   _serial->print("$PMTK220,200*2C\r\n");
15
16 }
17
18 void SrbGps::update() {
19
20   _updateSerial();
21
22 }
23
24 void SrbGps::_parseBuffer() {
25
26   // Create a new Nmea object and put the sentence in it
27   Nmea nmea;
28   nmea.write(_buffer);
29
30   // Abort if the sentence is invalid
31   if (!nmea.validate()) {
32     Serial.println("Invalid GPS sentence");
33     return;
34   }
35
36   // 1. Type, abort if not GPRMC
37   if (strcmp(nmea.nextField(), "GPRMC") != 0) return;
38
39   // 2. Time, discard this
40   nmea.nextField();
41
42   // 3. Status, abort if inactive
43   if (strcmp(nmea.nextField(), "A") == 0) {
44     _stats->gpsFix = 1;
45   } else {
46     _stats->gpsFix = 0;
47     return;
48   }
49
50   // 4-5. Latitude
51   _stats->lat  = _degToDec(nmea.nextField(), 2);
52   _stats->lat *= (strcmp(nmea.nextField(), "N") == 0) ?
         ↪ 1 : -1;
53
54   // 6-7. Longitude
55   _stats->lon  = _degToDec(nmea.nextField(), 3);
56   _stats->lon *= (strcmp(nmea.nextField(), "E") == 0) ?
         ↪ 1 : -1;
57
```

```
58    // 8. Speed (in knots)
59    _stats->speed = _knotsToMps(nmea.nextField());
60
61    Serial.print(_stats->lat, 6);
62    Serial.print(",");
63    Serial.print(_stats->lon, 6);
64    Serial.println("");
65
66    // Discard these fields:
67    // 9. Track angle
68    nmea.nextField();
69    // 10. Date
70    nmea.nextField();
71
72    // 11-12. Magnetic variation
73    _stats->magVar = strtod(nmea.nextField(), NULL);
74    _stats->magVar *= (strcmp(nmea.nextField(), "E") == 0)
          ↪ ? 1 : -1;
75
76  }
77
78  float SrbGps::_degToDec(const char *s, int degLen) {
79
80    // Convert degrees portion
81    char degStr[degLen];
82    strncpy(degStr, s, degLen);
83    float deg = strtod(degStr, NULL);
84
85    // Convert minutes portion
86    float mins = strtod(&s[degLen], NULL);
87
88    // Minutes to decmial using integer maths
89    long minsInt = mins * 10000;
90    minsInt = minsInt * 100 / 60;
91    mins = minsInt / 1000000.0;
92
93    // Add the two
94    return deg + mins;
95
96  }
97
98  float SrbGps::_knotsToMps(const char *s) {
99
100   float knots = strtod(s, NULL);
101   return knots / 1.944;
102
103 }
```

I. srb/srb_comms.h

```
1   /*
2      srb_comms.h
3      Communication functions for SRB
4      Written by Jarod Lam
5   */
6
7   #ifndef srb_comms_h
8   #define srb_comms_h
9
10  #include <Arduino.h>
11  #include "nmea.h"
12  #include "srb_stats.h"
13  #include "srb_serial.h"
14
15  class SrbComms : public SrbSerial {
16    public:
17
18      /*
19       * Initialise SrbGps with the XBee serial port.
20       */
21      SrbComms(SrbStats *stats, HardwareSerial *port);
22
23      /*
24       * Set the failsafe timeout in milliseconds.
25       * Run during setup.
26       */
27      void setTimeout(int ms);
28
29      /*
30       * Update the values from serial.
31       */
32      void update();
33
34      /*
35       * Send a message over serial
36       */
37      void sendMessage(const char* s);
38
39      /*
40       * General function for parsing SRB sentences.
41       */
42      void parseSentence(char *s);
43
44      /*
45       * Send SRBSM message
46       */
47      void sendSRBSM();
48
49    private:
50
51      /*
52       * Parse contents of buffer.
53       */
54      void _parseBuffer();
55
56      /*
57       * Failsafe timeout length. -1 for disabled.
58       */
59      int _failsafeTimeout = -1;
60      unsigned long _lastRecvMillis = 0;
61
62      /*
63       * Functions for parsing sentences.
64       */
65      void _readSRBJS(Nmea *nmea);
66      void _readSRBWP(Nmea *nmea);
67  };
68
69  #endif
```

J. srb/srb_comms.cpp

```
1   /*
2      srb_comms.cpp
3      Communication functions for SRB
4      Written by Jarod Lam
5   */
6
7   #include "srb_comms.h"
8
9   SrbComms::SrbComms(SrbStats *stats, HardwareSerial *port
        ↪ ) : SrbSerial(stats, port, 9600) {
10
11  }
12
13  void SrbComms::setTimeout(int ms) {
14    _failsafeTimeout = ms;
15  }
16
17  void SrbComms::update() {
18
19    // Check for failsafe timeout
20    if (_failsafeTimeout > 0 &&
21        _lastRecvMillis + _failsafeTimeout < millis()) {
22      _stats->state = 0;
23    }
24
25    _updateSerial();
26
27  }
28
29  void SrbComms::sendMessage(const char* s) {
30    _serial->println(s);
31  }
32
33  void SrbComms::_parseBuffer() {
34    parseSentence(_buffer);
35  }
36
37  void SrbComms::parseSentence(char *s) {
38    // Create a new Nmea object to parse the sentence
39    Nmea nmea;
40    nmea.write(s);
41
42    // Check for valid sentence
43    if (!nmea.validate()) {
44      Serial.print("Invalid sentence received: ");
45      Serial.println(s);
```

```
    return;
  }

  // 1. Sentence type
  const char *type = nmea.nextField();

  if (strcmp(type, "SRBJS") == 0) {
    _readSRBJS(&nmea);
  }
  else if (strcmp(type, "SRBWP") == 0) {
    _readSRBWP(&nmea);
  }
  else {
    Serial.print("Sentence type '");
    Serial.print(type);
    Serial.println("' not recognised.");
    return;
  }
}

void SrbComms::sendSRBSM() {
  // Create NMEA object
  Nmea nmea;
  nmea.begin();

  // 1. Sentence type
  nmea.append("SRBSM");

  // 2. SRB ID
  nmea.appendInt(_stats->ID);

  // 3. State
  nmea.appendFloat(_stats->state, 0);

  // 4. Latitude
  nmea.appendFloat(_stats->lat, 6);

  // 5. Longitude
  nmea.appendFloat(_stats->lon, 6);

  // 6. Speed
  nmea.appendFloat(_stats->speed, 2);

  // 7. Heading
  nmea.appendFloat(_stats->heading, 1);

  // 8. Battery voltage
  nmea.appendFloat(_stats->battV, 2);

  // 9. Forward power
  nmea.appendInt(_stats->forwardPower);

  // 10. Target latitude
  nmea.appendFloat(_stats->targetLat, 6);

  // 11. Target longitude
  nmea.appendFloat(_stats->targetLon, 6);

  // 12. Target heading
  nmea.appendFloat(_stats->targetHeading, 1);

  // Checksum
  nmea.appendChecksum();

  // Send the message
  sendMessage(nmea.read());
}

void SrbComms::_readSRBJS(Nmea *nmea) {
  // Check number of fields
  if (nmea->numFields() != 3) return;

  // 2. SRB ID, abort if doesn't match
  int recvID = strtod(nmea->nextField(), NULL);
  if (recvID != _stats->ID) return;

  // 3. Forward power
  _stats->forwardPower = strtod(nmea->nextField(), NULL)
      ↪ ;

  // 4. Heading
  _stats->targetHeading = strtod(nmea->nextField(), NULL
      ↪ );

  // Set state
  _stats->state = 1;
}

void SrbComms::_readSRBWP(Nmea *nmea) {

  // Check number of fields
  if (nmea->numFields() != 3) return;

  // 2. SRB ID, abort if doesn't match
  int recvID = strtod(nmea->nextField(), NULL);
  if (recvID != _stats->ID) return;

  // 3. Target latitude
  _stats->targetLat = strtod(nmea->nextField(), NULL);

  // 4. Target longitude
  _stats->targetLon = strtod(nmea->nextField(), NULL);

  // Set state
  _stats->state = 2;
}
```

K. srb/srb_imu.h

```
/*
   srb_imu.h
   IMU receiver for Sparkfun SEN-10736 using Razor AHRS
       ↪ firmware
   Written by Jarod Lam
*/

#ifndef srb_imu_h
#define srb_imu_h

#define IMU_BUFFER_SIZE 1024
#define IMU_BUFFER_TIMEOUT 1000

#include <Arduino.h>
#include "srb_serial.h"

class SrbImu : public SrbSerial{

  public:

    /*
     * Initialise SrbImu with the IMU serial port.
     */
    SrbImu(SrbStats *stats, HardwareSerial *port);

    /*
     * Update the values from serial.
     */
    void update();

  private:

    /*
     * Parse the sentence in the buffer.
     */
    void _parseBuffer();

};

#endif
```

L. srb/srb_imu.cpp

```
/*
   srb_imu.cpp
   IMU receiver for Sparkfun SEN-10736 using Razor AHRS
       ↪ firmware
   Written by Jarod Lam
*/

#include "srb_imu.h"

SrbImu::SrbImu(SrbStats *stats, HardwareSerial *port) :
    ↪ SrbSerial(stats, port, 57600) {

}

void SrbImu::update() {
```

```
14
15    _updateSerial();
16
17  }
18
19  void SrbImu::_parseBuffer() {
20
21    // Check if first five letters are "#YPR="
22    char s[10];
23    strncpy(s, _buffer, 5);
24    if (strcmp(s, "#YPR=") != 0) {
25      /*Serial.print("Invalid message from IMU: ");
26      Serial.println(_buffer);*/
27      return;
28    }
29
30    // Find the first comma position
31    int endInd = 0;
32    for (int i = 5; i < strlen(_buffer); i++) {
33      if (_buffer[i] == ',') {
34        endInd = i;
35        break;
36      }
37    }
38
39    // Make sure string length is within reasonable bounds
40    int numLen = endInd - 5;
41    if (numLen >= 10 || numLen <= 0) {
42      Serial.print("Unable to parse number from IMU: ");
43      Serial.println(_buffer);
44      return;
45    }
46
47    // Extract the number from the string
48    memset(s, 0, sizeof(s));
49    strncpy(s, &_buffer[5], numLen);
50
51    // Decode the number
52    float heading = strtod(s, NULL);
53
54    // Wrap around negative values
55    if (heading < 0) {
56      heading += 360;
57    }
58    heading = constrain(heading, 0, 360);
59
60    // Add magnetic variation (declination)
61    //heading += _stats->magVar;
62
63    // Update stats value
64    _stats->heading = heading;
65
66  }
```

M. srb/srb_motor.h

```
1   /*
2      srb_motor.h
3      Motor functions for SRB
4      Written by Jarod Lam
5   */
6
7   #ifndef srb_motor_h
8   #define srb_motor_h
9
10  #include <Arduino.h>
11  #include <Servo.h>
12  #include <stdlib.h>
13  #include "srb_stats.h"
14
15  // Pulse width min and max for motor signals
16  #define SRB_SERVO_MIN 1000
17  #define SRB_SERVO_MAX 2000
18
19  // Motor acceleration in power percentage points per
        ↪ second
20  #define MOTOR_ACCEL 100
21
22  class SrbMotor {
23
24    public:
25
26      /*
27       * Initialise the servo objects, call in setup()
28       */
29      void begin(int pins[], int sides[]);
30
31      /*
32       * Update motors every tick
33       */
34      void update();
35
36      /*
37       * Set power for specified motor
38       */
39      void setPower(int motorNo, float power);
40
41      /*
42       * Set power for specified side of motors
43       */
44      void setSidePower(int side, float power);
45
46      /*
47       * Set power for all left/right motors
48       */
49      void setLeft(int power);
50      void setRight(int power);
51
52      /*
53       * Stop all motors
54       */
55      void stopAll();
56
57      /*
58       * Return number of motors
59       */
60      int count();
61
62      /*
63       * Return the side of the SRB the motor is on (0=
             ↪ left, 1=right)
64       */
65      int side(int motorNo);
66
67    private:
68
69      /*
70       * Servos
71       */
72      Servo _motors[NUM_MOTORS];
73
74      /*
75       * Motor sides
76       */
77      int _motorSides[NUM_MOTORS];
78
79      /*
80       * Motor target powers
81       */
82      float _currentPowers[NUM_MOTORS];
83      int _targetPowers[NUM_MOTORS];
84
85      /*
86       * Milliseconds since last update
87       */
88      unsigned long _lastMillis;
89
90      /*
91       * Directly set speed of motor
92       */
93      void _setMotor(int motorNo, float power);
94
95  };
96
97  #endif
```

N. srb/srb_motor.cpp

```
1   /*
2      srb_motor.cpp
3      Motor functions for SRB
4      Written by Jarod Lam
5   */
6
7   #include "srb_motor.h"
8
9   void SrbMotor::begin(int pins[], int sides[]) {
10
```

```
11    // Attach motors to their pins
12    for (int i = 0; i < NUM_MOTORS; i++) {
13      _motors[i].attach(pins[i], SRB_SERVO_MIN,
            ↪ SRB_SERVO_MAX);
14      setPower(i, 0);
15      _motorSides[i] = sides[i];
16    }
17
18  }
19
20  void SrbMotor::update() {
21
22    // Get time since last update
23    unsigned long dt = millis() - _lastMillis;
24    _lastMillis = millis();
25
26    // Calculate amount to add to power
27    float a = (float)dt / 1000.0 * MOTOR_ACCEL;
28
29    // Check each motor if they need accelerating
30    for (int i = 0; i < NUM_MOTORS; i++) {
31
32      float dp = _targetPowers[i] - _currentPowers[i];
33
34      if (dp == 0) {
35        continue;
36      }
37      else if (abs(dp) < a) {
38        _setMotor(i, _targetPowers[i]);
39      }
40      else {
41        int sign = (dp > 0) - (dp < 0);
42        float newPower = _currentPowers[i] + (sign*a);
43        _setMotor(i, newPower);
44      }
45
46    }
47
48  }
49
50  void SrbMotor::setPower(int motorNo, float power) {
51
52    // Abort if motor number not in range
53    if (motorNo < 0 || motorNo >= NUM_MOTORS) return;
54
55    // Constrain to range
56    power = constrain(power, -100, 100);
57
58    // Update stored power value
59    _targetPowers[motorNo] = power;
60
61  }
62
63  void SrbMotor::setSidePower(int side, float power) {
64
65    for (int i = 0; i < NUM_MOTORS; i++) {
66      if (_motorSides[i] == side) {
67        setPower(i, power);
68      }
69    }
70
71  }
72
73  void SrbMotor::stopAll() {
74
75    for (int i = 0; i < NUM_MOTORS; i++) {
76      setPower(i, 0);
77    }
78
79  }
80
81  int SrbMotor::count() {
82
83    return NUM_MOTORS;
84
85  }
86
87  int SrbMotor::side(int motorNo) {
88
89    return _motorSides[motorNo];
90
91  }
92
93  void SrbMotor::_setMotor(int motorNo, float power) {
94
95    // Abort if motor number not in range
96    if (motorNo < 0 || motorNo >= NUM_MOTORS) return;
97
98    // Map value from power domain to angle domain
99    int val = map(power, -100, 100, 0, 180);
100
101   // Update stored power value and servo value
102   _currentPowers[motorNo] = power;
103   _motors[motorNo].write(val);
104
105 }
```

O. srb/srb_nav.h

```
1   /*
2      srb_nav.h
3      Navigatoin system for SRB
4      Written by Jarod Lam
5   */
6
7   #ifndef srb_nav_h
8   #define srb_nav_h
9
10  #include <Arduino.h>
11  #include <math.h>
12  #include "srb_stats.h"
13  #include "srb_motor.h"
14
15  class SrbNav {
16
17    public:
18
19      /*
20       * Intialise navigation with stats object.
21       */
22      SrbNav(SrbStats *stats, SrbMotor *motors);
23
24      /*
25       * Update function called every loop.
26       */
27      void update();
28
29    private:
30
31      /*
32       * Pointer to the stats object given at creation.
33       */
34      SrbStats *_stats;
35
36      /*
37       * Pointer to the motors object given at creation.
38       */
39      SrbMotor *_motors;
40
41      /*
42       * Navigation functions.
43       */
44      void _navDisabled();
45      void _navAuto();
46      void _navManual();
47
48      /*
49       * Get the difference between target and current
             ↪ heading
50       */
51      int _headingDiff(int goalDirection, int
             ↪ currentHeading);
52
53      /*
54       * Send the SRB in a particular direction with a
             ↪ particular power
55       */
56      void _moveTo(int power, int tHeading);
57
58      /*
59       * Calculate a multiplier used to scale motor power
             ↪ according to heading offset
60       */
61      float _turnMultiplier(int dHead);
62
63  };
64
65
66  #endif
```

P. srb/srb_nav.cpp

```
/*
    srb_nav.cpp
    Navigation system for SRB
    Written by Jarod Lam
*/

#include "srb_nav.h"

SrbNav::SrbNav(SrbStats *stats, SrbMotor *motors) {
  _stats = stats;
  _motors = motors;
}

void SrbNav::update() {

  // Check the SRB status and run corresponding function
  switch(_stats->state) {
    case 1: _navManual(); break;
    case 2: _navAuto();   break;
    default: _navDisabled();
  }

}

void SrbNav::_navDisabled() {

  _motors->stopAll();

}

void SrbNav::_navManual() {

  _moveTo(_stats->forwardPower, _stats->targetHeading);

}

void SrbNav::_navAuto() {

  // Calculate distance from goal
  float dLat = _stats->targetLat - _stats->lat;
  float dLon = _stats->targetLon - _stats->lon;

  // Get target heading of goal (convert to deg)
  int tHead = atan2(dLat, dLon) * 57.29578;

  // Move
  _moveTo(_stats->forwardPower, tHead);

}

int SrbNav::_headingDiff(int goalDirection, int
    ↪ currentHeading) {

  int maxHeading = max(goalDirection, currentHeading);
  int minHeading = min(goalDirection, currentHeading);

  int clockDiff = maxHeading - minHeading;
  int antiClockDiff = minHeading + 360 - maxHeading;
  int smallest = min(clockDiff, antiClockDiff);

  if (((currentHeading + smallest) % 360) ==
      ↪ goalDirection) {
    return smallest * -1; //clockwise
  } else {
    return smallest;
  }

}

void SrbNav::_moveTo(int power, int tHeading) {

  // Calculate distance from heading
  int dHead = _headingDiff(tHeading, _stats->heading);

  // Set motor speeds
  int powerL = power;
  int powerR = power;

  // Scale motors by heading rotation
  if (dHead < 0) {
    powerR *= _turnMultiplier(dHead);
  } else {
    powerL *= _turnMultiplier(dHead);
  }

  // Turn motors
  _motors->setSidePower(LEFT,  powerL);
  _motors->setSidePower(RIGHT, -powerR);

}

float SrbNav::_turnMultiplier(int dHead) {

  if (dHead < 0) dHead *= -1;
  dHead = constrain(dHead, 0, 180);

  // Scale according to the function y = (1/90)x + 1
  float multiplier = -(1.0 / 90.0) * dHead + 1;

  return multiplier;

}
```

Q. srb-base/srb-base.py

```
#!/usr/bin/env python3

import serial, sys, threading
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import BOTH, END, LEFT, RIGHT, DISABLED,
    ↪ NORMAL, N, S, E, W, X, Y
from serial.tools.list_ports import *
from datetime import *
from nmea import *

class Application(ttk.Frame):

    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)

        self.master.protocol("WM_DELETE_WINDOW", self.
            ↪ exitHandler)

        self.logFileOpen()
        self.setupGUI()
        self.setupThread()

    def setupGUI(self):
        self.pack(fill=BOTH, expand=True)
        self.master.title("SRB base")

        self.master.bind('<Return>', self.sendSentence)

        # Log box
        textScrollFrame = ttk.Frame(self)
        textScrollFrame.pack(fill=BOTH, expand=True)
        textScrollFrame.grid_columnconfigure(0,weight=1)
        textScrollFrame.grid_columnconfigure(0,weight=1)
        textScrollFrame.grid_rowconfigure(0,weight=1)

        scrollbar = ttk.Scrollbar(textScrollFrame)
        scrollbar.grid(column=1,row=0,sticky=N+S)
        self.textBox = tk.Text(textScrollFrame,
            ↪ yscrollcommand=scrollbar.set)
        self.textBox.grid(column=0,row=0,sticky=N+S+E+W)
        self.textBox.config(state=DISABLED)
        self.textBox.tag_configure("error", foreground="
            ↪ red")
        scrollbar.config(command=self.textBox.yview)

        # Command entry
        self.textEntry = ttk.Entry(self)
        self.textEntry.pack(fill=X)

        # Bottom controls
        controls = ttk.Frame(self)
        controls.pack(fill=X,ipady=3)

        self.autoscroll = tk.IntVar()
        scrollCheck = ttk.Checkbutton(controls, text="
            ↪ Autoscroll",
                                      variable=self.
                                          ↪ autoscroll
                                          ↪ )
        scrollCheck.pack(side=LEFT)
        self.autoscroll.set(1)

```

```
        serialPorts = [item.device for item in comports()]
        self.selectedPort = tk.StringVar()
        self.connectButton = ttk.Button(controls, text="Connect",
                                        command=self.openPort)
        self.connectButton.pack(side=RIGHT)
        portMenu = ttk.OptionMenu(controls, self.selectedPort, *serialPorts)
        portMenu.pack(side=RIGHT, fill=X, expand=True)

    def setupThread(self):
        self.ser = serial.Serial()
        self.serialThread = threading.Thread(target=self.readSerial)
        self.serialThread.daemon = True

    def log(self, text, error=False):
        # Get current time string
        timeStr = datetime.now().strftime("%H:%M:%S.%f")
        timeStr = timeStr[:-3]  # Truncate microseconds

        # Console
        print("[%s] %s" % (timeStr, text))

        # Log file
        self.logFile.write(timeStr + "," + text + "\n")
        self.logFile.flush()

        # GUI window
        self.textBox.config(state=NORMAL)
        if error:
            self.textBox.insert(END, "[%s] %s\n" % (timeStr, text), "error")
        else:
            self.textBox.insert(END, "[%s] %s\n" % (timeStr, text))
        if self.autoscroll.get():
            self.textBox.yview_moveto(1)
        self.textBox.config(state=DISABLED)

    def openPort(self):
        port = self.selectedPort.get()
        try:
            self.ser = serial.Serial(port, 9600, timeout=1)
            self.serialThread.start()
            self.log("Serial port opened at %s" % self.ser.name)
        except Exception as err:
            self.log("Error opening port %s: %s" % (port, str(err)), error=True)
            self.closePort()

    def closePort(self):
        self.ser.close()

    def readSerial(self):
        while True:
            if self.ser.isOpen():
                self.connectButton.config(state=NORMAL)
                line = self.ser.readline()
                if len(line) > 0:
                    self.parseSentence(line)
            else:
                self.connectButton.config(state=DISABLED)

    def parseSentence(self, sent):
        try:
            # Create a Nmea object and validate
            s = Nmea()
            s.sentence = sent.decode().rstrip('\r\n')
            if s.validate():
                self.log("[RECV] %s" % s.sentence)
            else:
                self.log("[RECV] %s" % s.sentence, error=True)
        except Exception as err:
            self.log("Error parsing sentence: %s" % str(err), error=True)

    def sendSentence(self, *args):
        if not self.ser.isOpen(): return

        # Convert to valid NMEA sentence
        text = self.textEntry.get()
        s = Nmea()
        s.append(text)
        s.appendChecksum()
        s.sentence = s.sentence + "\r\n"

        # Send over serial
        try:
            self.ser.write(s.sentence.encode())
            self.textEntry.delete(0, 'end')
            self.log("[SEND] %s" % s.sentence.rstrip('\r\n'))
        except Exception as err:
            self.log("Error sending sentence: " + str(err), error=True)

    def logFileOpen(self):
        now = datetime.now()
        fileName = now.strftime("srb_%Y-%m-%d_%H-%M-%S.log")
        self.logFile = open(fileName, "w+")

    def logFileClose(self):
        self.logFile.close()

    def exitHandler(self):
        self.closePort()
        self.master.destroy()
        self.logFileClose()

if __name__ == "__main__":
    root = tk.Tk()
    app = Application(root)
    root.mainloop()
```

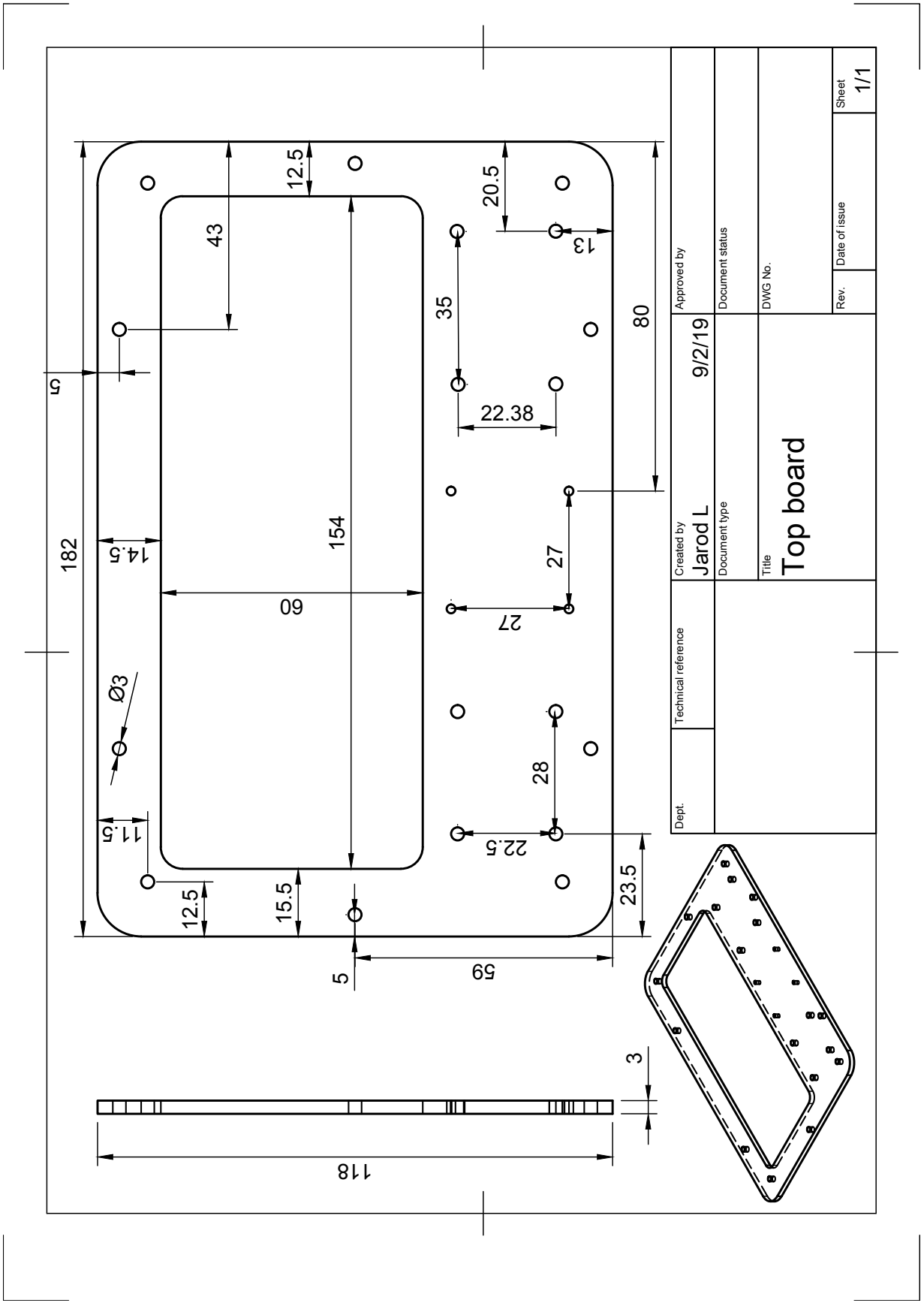
R. srb-base/nmea.py

```
#!/usr/bin/env python3

class Nmea:

    def __init__(self):
        self.sentence = "$"

    # Append string to end of sentence.
    def append(self, s):
        if len(self.sentence) > 0:
            if self.sentence[-1] != '$':
                self.sentence = self.sentence + ','

        self.sentence = self.sentence + s

    # Generate checksum for the sentence.
    def generateChecksum(self):
        checksum = 0

        for c in self.sentence.encode():
            if c == '$'.encode()[0]: continue
            if c == '*'.encode()[0]: break
            checksum ^= c

        checksumString = "%02X" % checksum
        return checksumString

    # Append checksum to the sentence.
    def appendChecksum(self):
        checksum = self.generateChecksum()
        self.sentence = self.sentence + '*' + checksum

    # Check if the sentence is valid.
    def validate(self):
        if self.sentence[0]  != '$': return False
        if self.sentence[-3] != '*': return False

        cs_recv = self.sentence[-2:]
        cs_calc = self.generateChecksum()
        if (cs_recv != cs_calc): return False

        return True

    # Return arguments as a list.
    def listify(self):
```

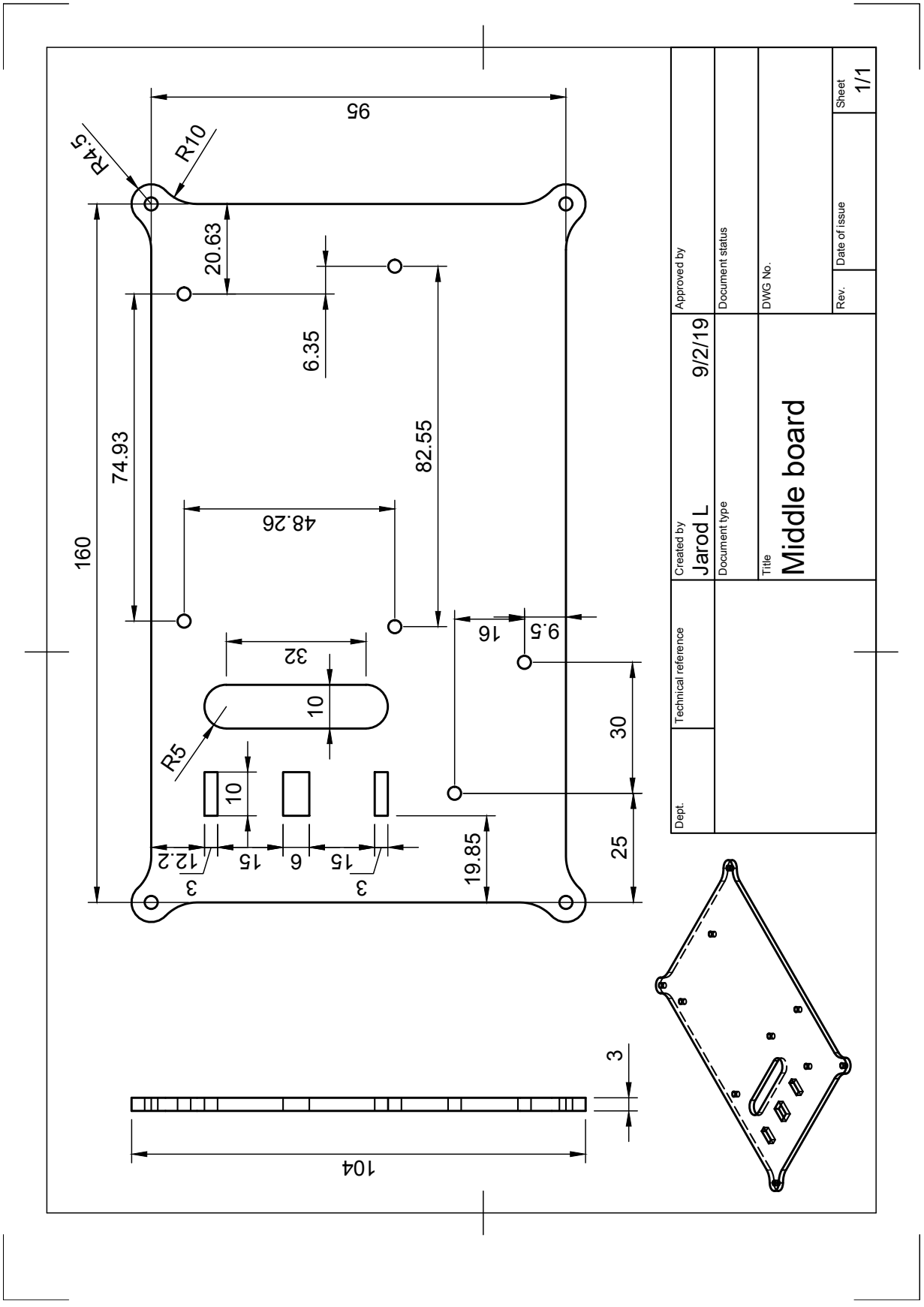
```
46          sentenceStripped = self.sentence[self.sentence.
                ↪ find("$")+1 :
47                                              self.sentence.
                                                    ↪ find("*
                                                    ↪ ")]
48          listified = sentenceStripped.split(',')
49          return listified
```

APPENDIX B
DRAWINGS

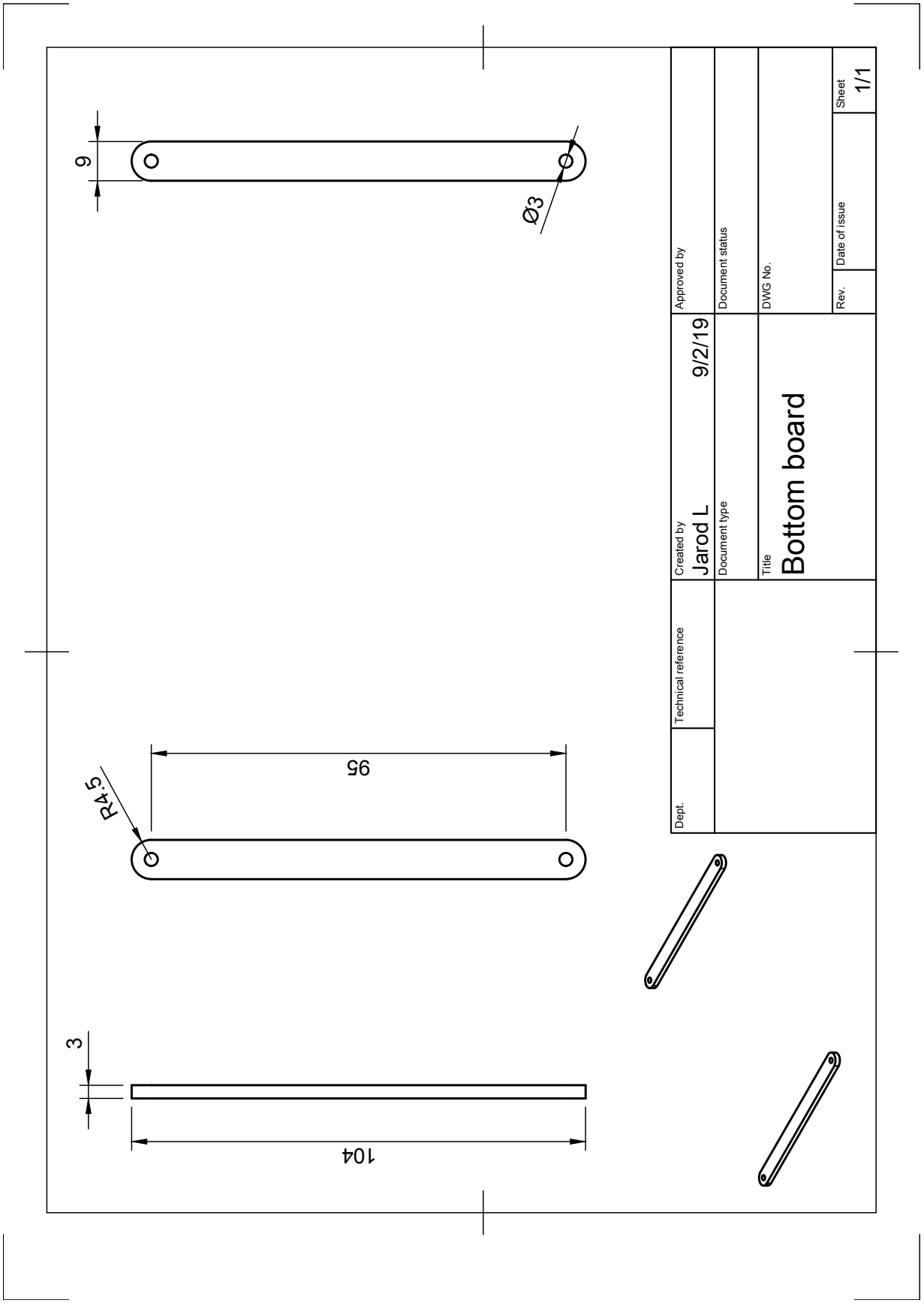
A. Top board



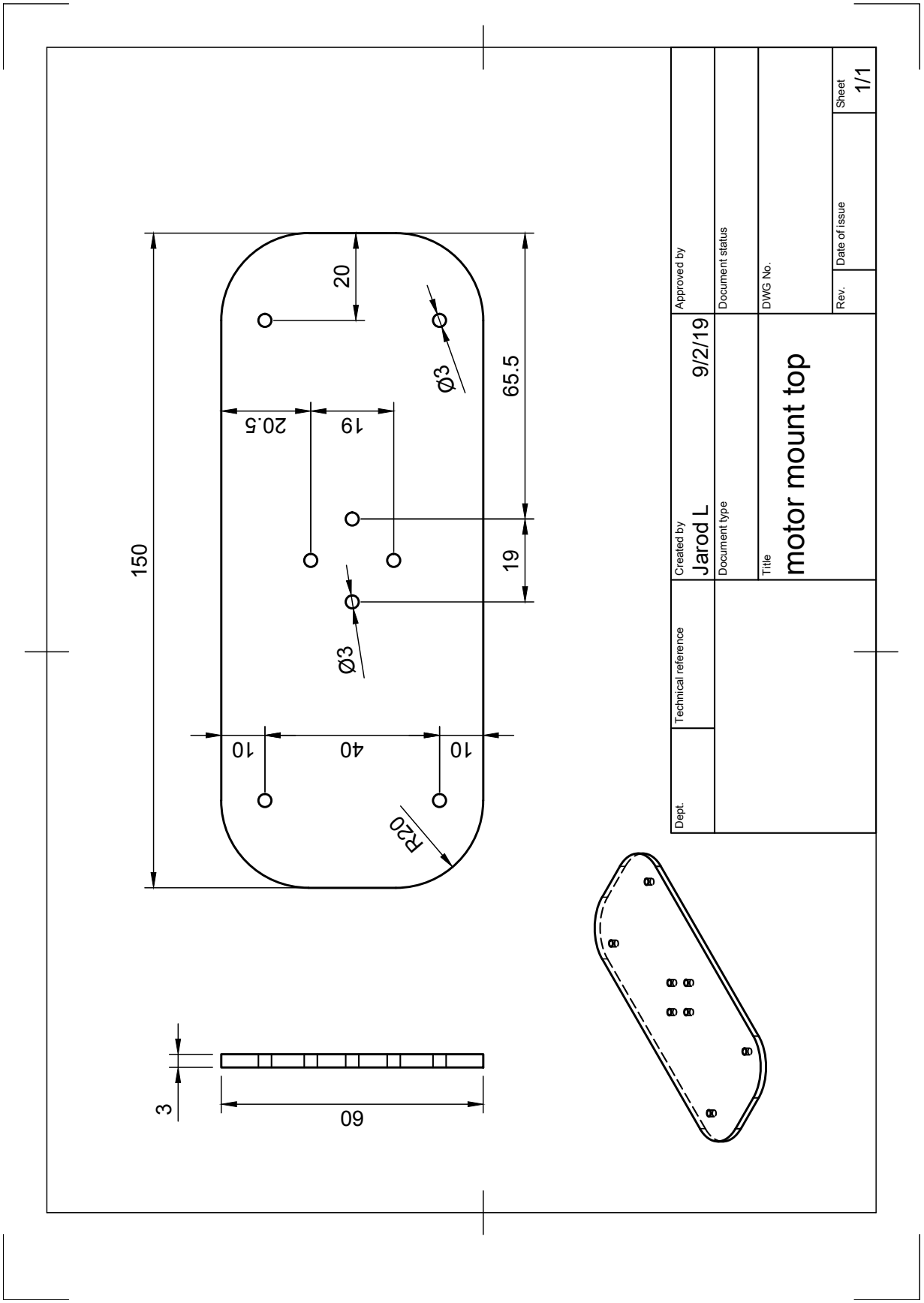
B. Middle board



C. Bottom board



D. Motor mount top



E. Motor mount bottom