“Better Sailing Through Robotics”

Katharine Priu, Nyeli Kratz, Elizabeth Aguirre

Introduction:

The residents at a local retirement community like to sail remote controlled sailboats on a nearby lake. However, algae tends to accumulate on the surface of the water. The purpose of this project is to design a sailboat that can patrol the lake autonomously to keep it free of algae. We programmed our boat with three modes: an idle mode where it is at rest and not moving, an autonomous mode where it drives around the pond in a circle to keep algae from accumulating on the surface of the pond, and a remote control mode, where the user can pilot the boat.

Design Process:

We used an Arduino Uno to control the electronics on the boat. A hobby servo motor controlled the rudder for steering, and a brushless DC motor drove the boat. We controlled the DC motor using an Electronic Speed Control. An HM10 bluetooth module connected to the Arduino allowed the boat to be controlled using the ArduinoBlue app on an iPhone. Our boat had 3 states: idle mode, RC mode, and autonomous mode. In idle mode, the boat sits still and floats on the lake. In RC mode, the user can control the boat using a virtual joystick in the iPhone app interface. In autonomous mode, the boat patrols the lake much like a roomba; it has an infrared sensor at the front that can see when it is approaching the edge of the lake, at which point it turns around and goes in a different predefined direction. The user can easily switch modes using pre-programmed buttons on their ArduinoBlue app.

We initially designed the boat to communicate using an RC transceiver attached to a second Arduino and physical buttons and joysticks rather than using bluetooth to communicate with an iPhone, and we were also using GPS instead of an infrared sensor. However, the SoftwareSerial object needed for the GPS and the transceiver seemed to be interfering with each other, so we decided to switch to bluetooth. This allowed us to eliminate the second Arduino and prevent the issues that we were having with the radio. Multiple teams communicating over the transceivers at the same time could have also led to radio issues, so we felt that eliminating the transceiver was best. Once we switched to bluetooth, however, the bluetooth’s SoftwareSerial object and the GPS’s hardware serial seemed to still be interfering with each other, so we decided to use an infrared sensor instead to eliminate interference. With the GPS off, the bluetooth communication worked well, but as soon as the GPS was reconnected, the bluetooth stopped working, so we realized that the two could not be run simultaneously.

Sensors and Actuators:

Our system uses an Arduino Uno microcontroller to control the following sensors and actuators:

Sensors:

- Sharp IR Sensor: infrared sensor used to calculate the distance between itself and an object in front of it. We used this sensor to detect when the boat is approaching the edge of the lake.
- HM-10 Bluetooth Module: bluetooth module that can communicate with an iPhone using bluetooth low energy (BLE). We connected this module to the ArduinoBlue app, which we used as the controller for the boat. Since the RX and TX pins required 3.3 V, and the Arduino pins provide 5 V, we used a voltage divider to decrease the voltage.
- iPhone: using the ArduinoBlue app, the iPhone communicated which state button had been pressed and what the current throttle position was with the Arduino. We had 3 buttons: 1 for each state (idle, remote control, autonomous).

Actuators:

- Hobby Servo Motor: controlled the direction of the rudders to steer the boat. The servo had a specified range of 0 to 180 degrees, but in reality the actual range of the rudders was approximately 20 to 160 degrees.
- DC Motor: drove and controlled the speed of the boat. We controlled this motor using an Electronic Speed Control (ESC), which allowed us to treat it as a servo motor, except the degree range corresponded to the speed of the propeller instead of the rotation of the motor (i.e. 0 degrees was still, 180 degrees was full speed).

Circuit Diagram and iPhone Interface:

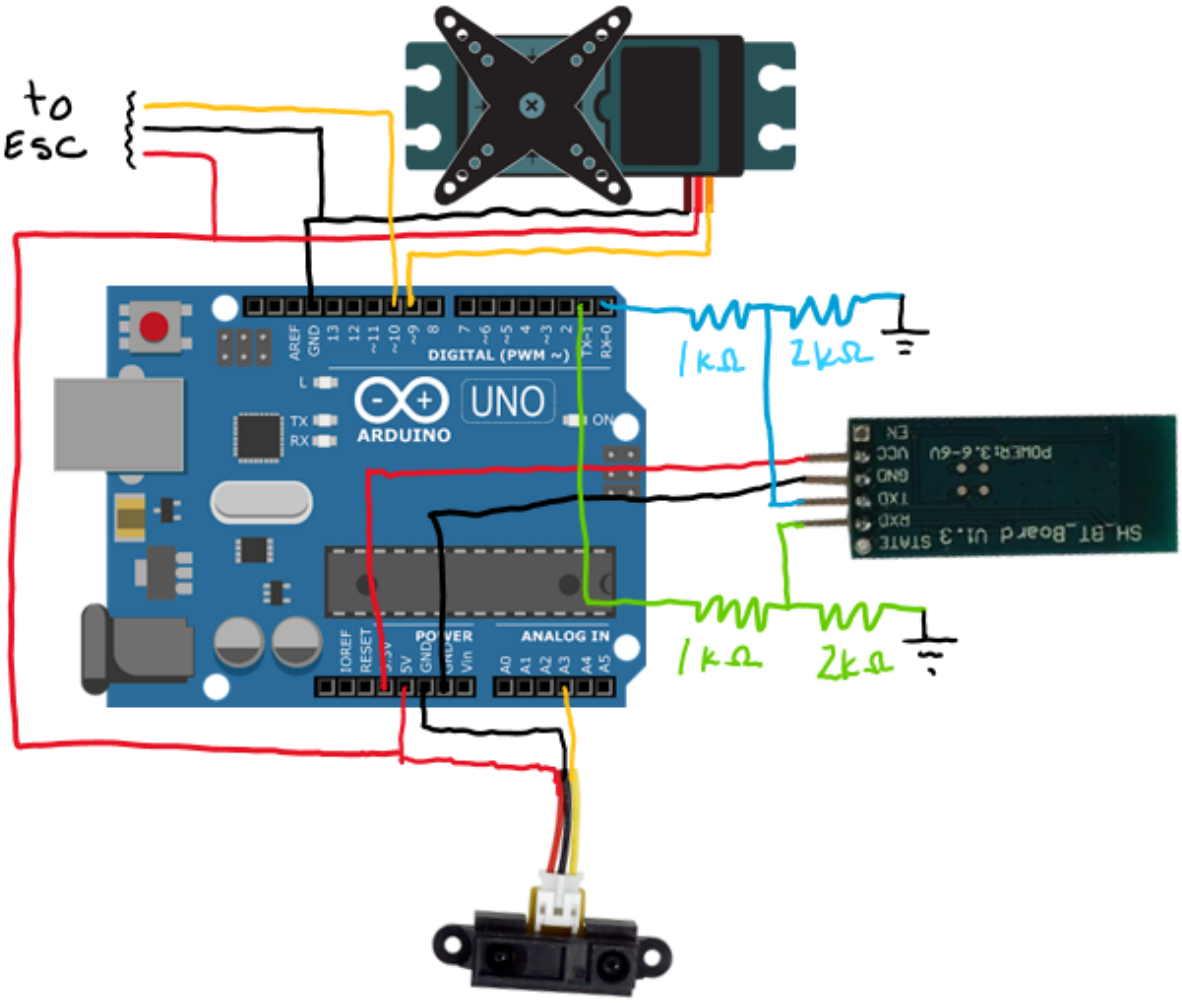

Figure 1. Circuit diagram of the on-board electronics. Note: all grounds are connected to Arduino ground.

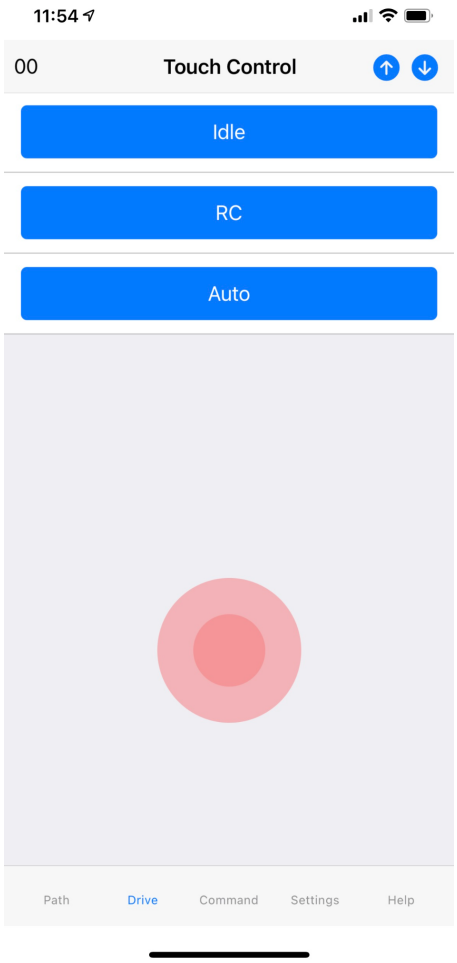

Figure 2. The iPhone's ArduinoBlue interface.

Photos:

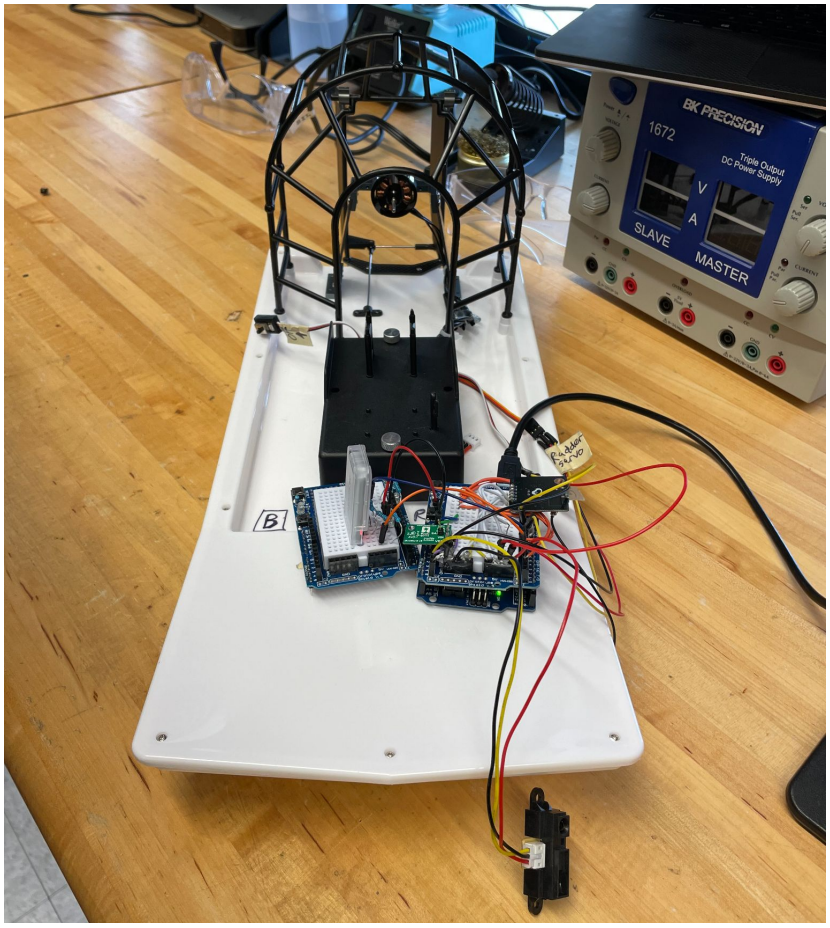

Figure 3. The final boat setup. Note: When used in real life, the infrared sensor would be mounted at the front of the boat, facing outwards.

Budget:

Component	Price
Arduino Uno	$23.00
Sharp IR Sensor	$9.88
Bluetooth Module	$10.99
Hobby servo motor	$2.50
DC motor	$2.00
Arduino Protoboard shield	$7.99
Electronic Speed Control	$17
Total:	$73.36

Arduino Program:

```
#include <SoftwareSerial.h>
#include <ArduinoBlue.h>
#include <Servo.h>
#include <SharpIR.h>


// The bluetooth tx and rx pins must be supported by software serial.
// Visit https://www.arduino.cc/en/Reference/SoftwareSerial for unsupported pins.
// Bluetooth TX -> Arduino pin 4
const int BLUETOOTH_TX = 4;
// Bluetooth RX -> Arduino pin 3
const int BLUETOOTH_RX = 3;

Servo serv; //the rudder servo object
Servo drive; //the propeller servo object

int state = 101; //the current state: idle = 101, RC = 102, autonomous = 103

int throttle, steering; //variables to store the throttle and steering values

SoftwareSerial bluetooth(BLUETOOTH_TX, BLUETOOTH_RX); //software serial object used for bluetooth
ArduinoBlue phone(bluetooth); // pass reference of bluetooth object to ArduinoBlue constructor.

SharpIR ir(SharpIR::GP2Y0A21YK0F, A3); //the infrared sensor object

int numUnder; //store the number of data points outside radius, for error managing

// Setup code runs once after program starts.
void setup() {
  //attach both the drive and rudder servos
  serv.attach(9);
  drive.attach(10);
```

```
  Serial.begin(9600); // Start serial monitor at 9600 bps

  // Start bluetooth serial at 9600 bps
  bluetooth.begin(9600);

  // delay just in case bluetooth module needs time to "get ready"
  delay(100);

  //initialize the drive motor by having it speed up and then slow back down again
  for (int throttle = 0; throttle < 180; throttle++) {
    drive.write(throttle);
    delay(5);
  }
  for (int throttle = 180; throttle > 0; throttle--) {
    drive.write(throttle);
    delay(5);
  }

  Serial.println("setup complete");


}

// Put your main code here, to run repeatedly:
void loop() {
  int newState = phone.getButton(); //get the current button reading
  //if the button reading is valid (button has been pressed), change state
  if (newState != -1) {
    state = newState;
  }
  switch (state) {
    case 101: //idle mode
    {
      drive.write(0); //turn off drive motor, nothing moving on the boat
      break;
    }
    case 102: //RC mode
    {
      //getThrottle() and getSteering() return integers between 0 and 99
      throttle = phone.getThrottle(); //get the joystick reading for throttle
      steering = phone.getSteering(); //get the joystick reading for steering

      serv.write((steering / 99.0) * 180.0); //set the rudder to the user specified value

      /* Since the boat cannot go backwards, have the throttle value be zero unless the
      joystick is pushed up */
      if (throttle > 50) {
        drive.write(((throttle - 50) / (50.0)) * 180.0); //set the drive motor to specified value
      } else {
        drive.write(0); //turn off the drive motor
      }
      break;
    }
    case 103: //autonomous mode
    {
      int x = ir.getDistance(); //distance reading from IR sensor
      Serial.println(x); //print this reading (for debugging)
      drive.write(120); //set the drive motor speed

      /* Since there is some error in the IR sensor, we decided it had to read 5 values under
      the specified limit before adjusting the steering */
      if (x < 20 && numUnder > 5) {
```

```
        serv.write(10); //if this limit has been reached, turn the boat
        delay(500); //delay to give time to turn away from the water's edge
      } else if (x < 20) { //if the threshold has not been reached yet, but the value is under
        numUnder++; //increment the number of times the boat has been under the threshold
        serv.write(90); //continue driving straight
        delay(100);
      } else {
        numUnder = 0; //if the value is over the threshold, reset counter
        serv.write(90); //drive straight
        delay(100);
      }
      break;
    }
  }
}
```

Conclusion

With a few design changes from the sensors that we were originally planning to use, we were successful in creating a boat which can autonomously patrol the lake to keep it free of algae, idle on the lake's surface, and also be driven remotely. If we were to mass produce this boat, we would use an IR sensor with better range, or use an ultrasonic sensor instead of the IR sensor. The IR sensor is only accurate at close distances, and the boat would have less chance of hitting the edge of the lake if it could see the edge approaching from farther away. Our autonomous mode control strategy is superior to that of the GPS because the rectangular shape of the lake prevents the boat from being able to reach the edges of the lake when it is constrained to a simple circle. The circle's location also depends on the point which the user chooses to be the center, and if they chose a point too close to the edge of the lake, then the boat could unknowingly hit the bank and is unlikely to cover very much of the edge of the lake. With our strategy of driving straight until the boat senses the edge and then turning, the boat is likely to cover more of the lake's edge than just driving in a circle of specified radius. However, if we wanted the boat to follow the lake's edge closely in autonomous mode, we could use a PID controller to stay a specified distance from the lake's edge, which would make the boat much more effective at clearing algae from the edge of the lake. Despite these potential improvements, our boat provided a simple solution to the proposed problem, stayed below the budget of $75, and was ultimately successful.

Appendix

GPS code:

```
#include <SoftwareSerial.h>
#include <ArduinoBlue.h>
#include <Servo.h>
#include <TinyGPS.h>

// Bluetooth TX -> Arduino pin 4
const int BLUETOOTH_TX = 4;
// Bluetooth RX -> Arduino pin 3
const int BLUETOOTH_RX = 3;

Servo serv; //steering servo
Servo drive; //drive servo

int state = 101; //the current state: idle = 101, RC = 102, autonomous = 103

int throttle, steering; //variables to store the throttle and steering values

SoftwareSerial bluetooth(BLUETOOTH_TX, BLUETOOTH_RX); //bluetooth's SoftwareSerial object
ArduinoBlue phone(bluetooth); // pass reference of bluetooth object to ArduinoBlue constructor.

TinyGPS tiny = TinyGPS(); //the GPS object

float lat1, lon1; //the starting GPS latitude and longitude
unsigned long age1; // the starting GPS age

int autoEntry = 0; //store whether autonomous has been entered

int radius = 15; //max autonomous radius, in meters

bool useGPS = true; //for testing; true if we are using the GPS

// Setup code runs once after program starts.
void setup() {
  serv.attach(9);
  drive.attach(10);

  // Start bluetooth serial at 9600 bps.
  bluetooth.begin(9600);

  Serial.begin(9600);

  // delay just in case bluetooth module needs time to "get ready".
  delay(100);

  //initialize the drive motor by having it speed up and then slow back down again
  for (int throttle = 0; throttle < 180; throttle++) {
    drive.write(throttle);
    delay(5);
  }
  for (int throttle = 180; throttle > 0; throttle--) {
    drive.write(throttle);
    delay(5);
  }

  if (useGPS) { //if the GPS is being used
    bool statusGPS = false; //initially set the status to false (hasn't gotten a reading)
    while (!statusGPS) { //loop until GPS gets a reading
      if (Serial.available() && tiny.encode(Serial.read())) {
        statusGPS = true; //set the reading status to true
```

```
      }
    }
  }

  Serial.println("setup complete");


}

// Put your main code here, to run repeatedly:
void loop() {
  int newState = phone.getButton(); //get the current button reading
  //if the button reading is valid (button has been pressed), change state
  if (newState != -1) {
    state = newState;
  }
  switch (state) {
    case 101: //idle mode
    {
      drive.write(0); //turn off drive motor, nothing moving on the boat
      break;
    }
    case 102: //RC mode
    {
      //getThrottle() and getSteering() return integers between 0 and 99
      throttle = phone.getThrottle(); //get the joystick reading for throttle
      steering = phone.getSteering(); //get the joystick reading for steering

      serv.write((steering / 99.0) * 180.0); //set the rudder to the user specified value

      /* Since the boat cannot go backwards, have the throttle value be zero unless the
      joystick is pushed up */
      if (throttle > 50) {
        drive.write(((throttle - 50) / (50.0)) * 180.0); //set the drive motor to specified value
      } else {
        drive.write(0); //turn off the drive motor
      }
      break;
    }
    case 103: //autonomous mode
    {
      if (useGPS) { //if the GPS is being used
        if (!autoEntry) { //if autonomous mode is being entered for the first time
          if (Serial.available() && tiny.encode(Serial.read())) {
            //set the current coordinates as the center of the boat's patrol circle
            tiny.f_get_position(&lat1, &lon1, &age1);
            autoEntry = 1; //set to true since autonomous mode has now been entered
          }
        } else {
          if (Serial.available() && tiny.encode(Serial.read())) {
            float lat2, lon2; //the new latitude and longitude
            unsigned long age2; //the new age
            tiny.f_get_position(&lat2, &lon2, &age2); //find the current postion
            //calculate the north/south and east/west distance from the starting point
            float NS = TinyGPS::distance_between(lat1, lon1, lat2, lon1);
            float EW = TinyGPS::distance_between(lat1, lon1, lat1, lon2);
            float radialDist = sqrt(sq(NS) + sq(EW)); //radial distance from start
            //if the boat has left it's patrol circle, it needs to turn around
            if (radialDist > radius) {
              serv.write(10); //turn boat
              delay(500); //delay to give time to turn fully
            } else {
```

```
            serv.write(90); //drive straight
          }
        }
      }
      drive.write(120); //set the drive motor speed
    }
    break;
  }
 }
}
```