Autonomous WIL

Objective

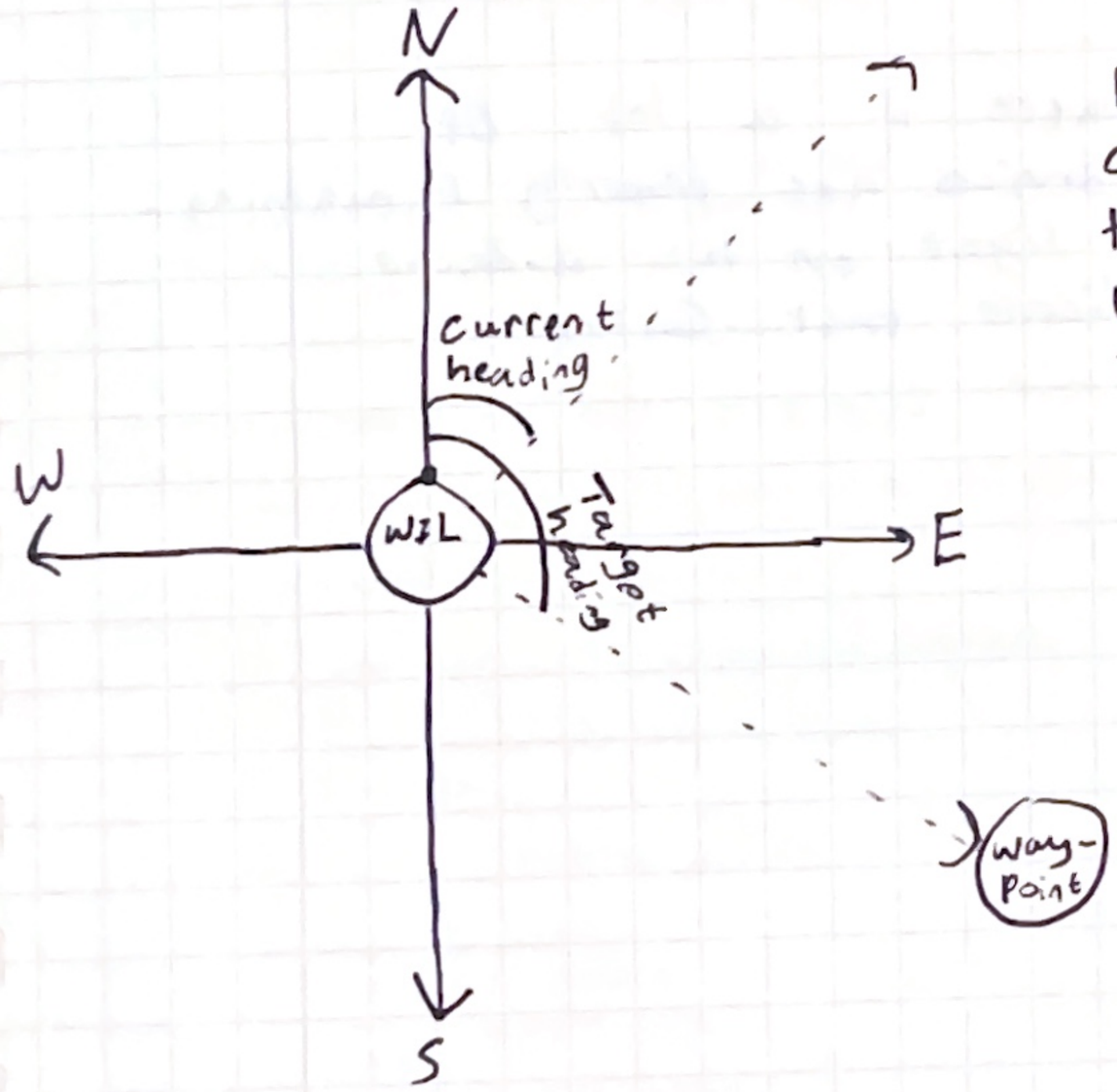
Combine the knowledge of GPS, BNO085, T200 thrusters, and LoRA to develop an autonomous boat affectionately call "WIL". Water is Life.

The gps will provide current location information.
The BNO will provide heading information.
Using google maps, we can select GPS destinations or "waypoints".

Using the current location and the Waypoint information, we can calculate what heading WIL needs to be to at. to travel to the destination.

The BNO will ~~tells~~ tell us. if we get too far off from the target heading.

We have added a MKR 1310 to allow for LoRa communications with a controller.



When we detect that the current heading is off from the target by some threshold, we control the motors in a way that rotates WIL towards the waypoint.

Code

```
#ifndef MainObjects_h_
# define MainObjects_h_

//////////////////
// GNSS Object //
#include <SparkFun_u-blox_GNSS_v3.h>
SFE_UBLOX_GNSS_SERIAL myGNSS;

//////////////////
// BNO Object //
#include <Adafruit_BNO08x_RVC.h>
Adafruit_BNO08x_RVC myIMU = Adafruit_BNO08x_RVC();
const int bnoRST = 13;

//////////////
// Servos // for the motor esc's
#include <Servo.h>
const int leftServoPin = 2;
const int rightServoPin = 7;
Servo leftServo;
Servo rightServo;

//////////////////
// Program Vars //
bool programState = 0; → 0 = Autonomous, 1 = Remote control.
int leftMotorSpeed = 1500;
int rightMotorSpeed = 1500; } 1500 = stopped
int leftMotorSpeedTarget = 1500;
int rightMotorSpeedTarget = 1500; } updated from remote control.

#endif
```

These variables and objects are used throughout the program.

On the following page, there is a lot of code that ensures different devices are properly functioning. If some component fails, a light on the arduino giga will light up to indicate that failure.

Autonomous WIL

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
1 #include "MainObjects.h"
2 #include "AutoMode.h"
3 #include "RemoteMode.h"
4
5 #include <SPI.h>
6 #include <SD.h>
7 #include <Adafruit_BNO08x_RVC.h>
8 #include <SparkFun_u-blox_GNSS_v3.h>
9 #include <Servo.h>
```

} Files I made

} Libraries we are using

```
36 void setupSD() // This method ensures that our SD card
37 {                  is properly functioning.
38     int chipSelect = 10;
39     pinMode(chipSelect, OUTPUT);
40     if(!SD.begin(chipSelect))
41     {
42         Serial.println("SD card failed to init... Freezing.");
43
44         digitalWrite(LED_BUILTIN, HIGH);
45         while(1);
46     }
47     Serial.println("Card Initialized!");
48
49     File outputfile = SD.open("data.txt", FILE_WRITE);
50     outputfile.println("Latitude, Longitude, Name, DistanceBetween, CurrentHeading, Targ
51     outputfile.close();
52 }
54 void setupGNSS() // This method ensures that our GPS is
55 {                  properly functioning.
56     Serial1.begin(38400); // RX0 TX0
57
58     if (!myGNSS.begin(Serial1))
59     {
60         Serial.println("GNSS failed to init... Freezing.");
61
62         digitalWrite(LED_BUILTIN, HIGH);
63         while(1);
64     }
65
66     myGNSS.setUART1Output(COM_TYPE_UBX);
67     myGNSS.setI2COutput(COM_TYPE_UBX);
68     myGNSS.saveConfiguration();
69
70     Serial.println("GNSS initialized!");
71 }
73 void setupBNO() // This method ensures our inertial
74 {                  measurement unit is
75     Serial2.begin(115200); // Baud set by datasheet properly functioning.
76
77     if (!myIMU.begin(&Serial2))
78     {
79         Serial.println("Could not find BNO08x!");
80
81         digitalWrite(LED_BUILTIN, HIGH);
82         while(1);
83     }
84     Serial.println("BNO08x found!");
85
86     // Reset the heading
87     pinMode(bnoRST, OUTPUT);
88     digitalWrite(bnoRST, LOW);
89     delay(10);
90     digitalWrite(bnoRST, HIGH);
91
92     Serial.println("BNO heading zeroed...");
93 }
```

Autonomous WIL

07/30/2025
Wesley
Code

```
95 void setupServos()
96 {
97     leftServo.attach(leftServoPin);
98     rightServo.attach(rightServoPin);
99
100    leftServo.writeMicroseconds(1500);
101    rightServo.writeMicroseconds(1500);
102
103    // Let the ESC's catch up
104    // Listen for a high pitched tone after the first sequence of notes.
105    delay(10000);
106 }
108 void setup()
109 {
110    Serial.begin(115200);
111    Serial.println("Initializing sensors!");
112
113    pinMode(LED_BUILTIN, OUTPUT);
114
115    // setupUSB();
116    setupSD();
117
118    setupGNSS();
119
120    setupBNO();
121
122    setupServos(); // Delays for 10 seconds for the ESC's
123
124    Serial3.begin(115200); // For the LoRa UART.
125 }
127 void loop()
128 {
129    readController(); // Make sure we read the controller
130
131    if (programState == 0) // If the controller is in autonomous mode
132    {
133        autoMode(); // do the autonomous mode.
134    }
135    else if (programState == 1) // If the controller is in remote control
136    {                           mode
137        remoteMode();
138    }                  // do the remote mode.
139 }
```

Here, our code is fairly simple:
~~On the~~ On start, run all the setup methods.

In our loop, we are ~~ret~~ checking to see if the controller has sent some information. If it has changed which program mode we are in, the appropriate function will be called.

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
#ifndef AutoMode_h_
# define AutoMode_h_

#include "GetHeading.h"
#include "TargetHeading.h"
#include "DistanceBetween.h"
#include "UpdateGPS.h"
#include "HeadingError.h"
#include "RotateInPlace.h"
#include "LogToSD.h"
#include "UpdateMotors.h"
#include "LogToSD.h"
// Index that we access the destination arrays by.
int waypointIndex = 0;

// Around the block but more waypoints
uint8_t numWaypoints = 27;

double currentHeading;

// Current Latitude and Longitude
long cLat;
long cLon;
long dLat; // destination Lat + Lon
long dLon;

long gpsHeading;
long gpsGroundSpeed;

double distanceBetween; // distance between current position and destination
double targetHeading;

double headingError;
double acceptableHeadingError = 0.02;

byte SIV;
// Destination Longitudes and Lattitudes
long dLons [] = { -820051916, ...}; // Cut for brevity
long dLats [] = { 334924613, ...};
void printVars()
{
  Serial.print("SIVs: ");
  Serial.println(SIV);

  Serial.print("Current Lat: ");
  Serial.println(cLat);

  Serial.print("Current Lon: ");
  Serial.println(cLon);

  Serial.print("Target Heading: ");
  Serial.println(targetHeading, 6);

  Serial.print("Current Heading: ");
  Serial.println(currentHeading);

  Serial.print("Heading Error: ");
  Serial.println(headingError, 6);

  Serial.print("Distance Between: ");
  Serial.println(distanceBetween, 6);
}
```

These are files that we are using to autonomously navigate.

Autonomous WIL

07/30/2023
Wesley Lacke

```
160 void autoMode()
161 {
162   // Get the destination we want to go to based on the index.
163   dLat = dLats[waypointIndex];
164   dLon = dLons[waypointIndex];
165
166   // Update our GPS variables and their distance
167   updateGPS(&cLat, &cLon, &SIV, &gpsHeading, &gpsGroundSpeed);
168   distanceBetween = getDistanceBetween(cLat, cLon, dLat, dLon);
169
170   // Get the currentHeading, targetheading, and the error between them.
171   currentHeading = getHeading();
172   targetHeading = getTargetHeading(cLat, cLon, dLat, dLon);
173   headingError = getHeadingError(currentHeading, targetHeading);
174
175   // If at least 4 sats are in view
176   if (SIV >= 4)
177   {
178     // Heading is too far off
179     if (abs(headingError) > acceptableHeadingError)
180     {
181       rotateInPlace(&targetHeading, &currentHeading, &headingError, &acceptableHeadingEr
182     }
183
184     // We have entered remote control mode...
185     if (programState == 1) *
186     {
187       return;
188     }
189     // We are close enough to our target waypoint
190     if (distanceBetween <= 3.0)
191     {
192
193       // Turn motors off and go to the next waypoint
194       updateMotors(1500, 1500);
195       waypointIndex++;
196
197       // Next waypoint is out of bounds
198       if (waypointIndex >= numWaypoints)
199       {
200         // Reset
201         programState = 1;
202         waypointIndex = 0;
203         return;
204       }
205
206       // Our heading is not too far off
207       // and we are not close enough to the point
208       else
209       {
210         // Go Forwards
211         updateMotors(1700, 1700);
212       }
213     }
214     // 4 Sats are not in view. Sit still...
215     else
216     {
217       updateMotors(1500, 1500);
218       Serial.println("No satellites yet...");
219     }
220
221     // Log data to the SD card for analysis
222     logToSd(cLat, cLon, distanceBetween, currentHeading, targetHeading, gpsHeading, gpsGro
223     // Log to serial monitor for debugging
224     printVars();
225 }
226
227 #endif
```

① updateGPS
② getDistanceBetween
③ getHeading
④ getTargetHeading
⑤ getHeadingError
⑥ rotateInPlace
⑦ updateMotors
⑧ logToSd

3.0 m radius

This is the main autonomous navigation algorithm.

* Inside the rotate in place method, we are checking the controller for updates.

I have labeled some helper functions on the previous page ① - ⑧. Here is the code for them.

①

This method gets important information from the GPS.

```
#ifndef UpdateGPS_h_
# define UpdateGPS_h_

#include "MainObjects.h"

void updateGPS(long* aLat, long* aLon, byte* aSIV, long* aHeading, long* aGroundSpeed)
{
    *aLat = myGNSS.getLatitude();
    *aLon = myGNSS.getLongitude();
    *aSIV = myGNSS.getSIV();
    *aHeading = myGNSS.getHeading();
    *aGroundSpeed = myGNSS.getGroundSpeed();
}

#endif
```

②

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

This method computes the distance between a pair of Lat & Lons.

```
#ifndef DistanceBetween_h_
# define DistanceBetween_h_

double getDistanceBetween(long lat1_1, long long1_1, long lat2_1, long long2_1)
{
  // returns distance in meters between two positions, both specified
  // as signed decimal-degrees latitude and longitude. Uses great-circle
  // distance computation for hypothetical sphere of radius 6372795 meters.
  // Because Earth is no exact sphere, rounding errors may be up to 0.5%.
  // Courtesy of Maarten Lamers

  double lat1 = (double)lat1_1 / 10000000.0; // Convert lat and long to degrees
  double long1 = (double)long1_1 / 10000000.0;
  double lat2 = (double)lat2_1 / 10000000.0;
  double long2 = (double)long2_1 / 10000000.0;
  double delta = radians(long1-long2);
  double sdlong = sin(delta);
  double cdlong = cos(delta);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  double slat1 = sin(lat1);
  double clat1 = cos(lat1);
  double slat2 = sin(lat2);
  double clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  double denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  return delta * 6372795;
}

#endif
```

Autonomous WIL

07/30/2023
Weekly check

(3)

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```h
#ifndef GetHeading_h_
# define GetHeading_h_

#include "MainObjects.h"

BNO08x_RVC_Data bnoData;
const float declination = 7;

float getHeading()
{
    // Clear the buffer before reading
    while (Serial2.available() > 0)
    {
        Serial2.read();
    }

    // Continue reading until we get a new heading.
    while(!myIMU.read(&bnoData)) {}

    float heading = bnoData.yaw - declination;

    if (heading < 0)
    {
        heading += 360;
    }

    return heading;
}

#endif
```

(4)

```h
#ifndef TargetHeading_h_
# define TargetHeading_h_

double getTargetHeading(long lat1_l, long long1_l, long lat2_l, long long2_l)
{
  // returns course in degrees (North=0, West=270) from position 1 to position 2,
  // both specified as signed decimal-degrees latitude and longitude.
  // Because Earth is no exact sphere, calculated course may be off by a tiny fraction.
  // Courtesy of Maarten Lamers

  double lat1 = (double)lat1_l / 10000000.0; // Convert lat and long to degrees
  double long1 = (double)long1_l / 10000000.0;
  double lat2 = (double)lat2_l / 10000000.0;
  double long2 = (double)long2_l / 10000000.0;
  double dlon = radians(long2-long1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  double a1 = sin(dlon) * cos(lat2);
  double a2 = sin(lat1) * cos(lat2) * cos(dlon);
  a2 = cos(lat1) * sin(lat2) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0)
  {
    a2 += TWO_PI;
  }
  return degrees(a2);
}

#endif
```

(5)

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
#ifndef HeadingError_h_
# define HeadingError_h_

double getHeadingError(double aHeading, double aTargetHeading)
{
    // What is the Difference in our heading
    double error = aTargetHeading - aHeading;

    // Rescale to give shortest turn distance.
    // Will be between -180 and 180.
    if (error >= 180)
    {
        error -= 360;
    }
    else if (error <= -180)
    {
        error += 360;
    }
    return error/180.0;
}

#endif
```

(6)

```
#ifndef RotateInPlace_h_
# define RotateInPlace_h_

#include "HeadingError.h"
#include "GetHeading.h"
#include "MainObjects.h"
#include "ReadController.h"
#include "UpdateMotors.h"
void rotateInPlace(double* theTargetHeading, double* theHeading, double* theError, double* acceptableError)
{
    while(abs(*theError) > *acceptableError)
    {
        Serial.println("Fixing the heading...");
        // Pivot right
        if (*theError > 0)
        {
            updateMotors(1700, 1280);
        }
        // Pivot Left
        else
        {
            updateMotors(1280, 1700);
        }

        *theHeading = getHeading();

        *theError = getHeadingError(*theHeading, *theTargetHeading);

        Serial.print("Fixing Error: ");
        Serial.println(*theError);

        readController();

        if (programState == 1)
        {
            return;
        }
    }
}

#endif
```

Autonomous WIL

07/30/2023,
Wednesday
Osaka

(7)

```
#ifndef UpdateMotors_h_
# define UpdateMotors_h_

#include "MainObjects.h"

void updateMotors(int targetLeft, int targetRight)
{
    while (leftMotorSpeed != targetLeft OR rightMotorSpeed != targetRight)
    {
        if (leftMotorSpeed != targetLeft)
        {
            if(leftMotorSpeed < targetLeft)
            {
                leftMotorSpeed++;
            }
            else
            {
                leftMotorSpeed--;
            }
        }

        if (rightMotorSpeed != targetRight)
        {
            if(rightMotorSpeed < targetRight)
            {
                rightMotorSpeed++;
            }
            else
            {
                rightMotorSpeed--;
            }
        }

        leftServo.writeMicroseconds(leftMotorSpeed);
        rightServo.writeMicroseconds(rightMotorSpeed);
    }
}

#endif
```

only change the motor speed by 1 at a time.

This helps eliminate abrupt flutters.

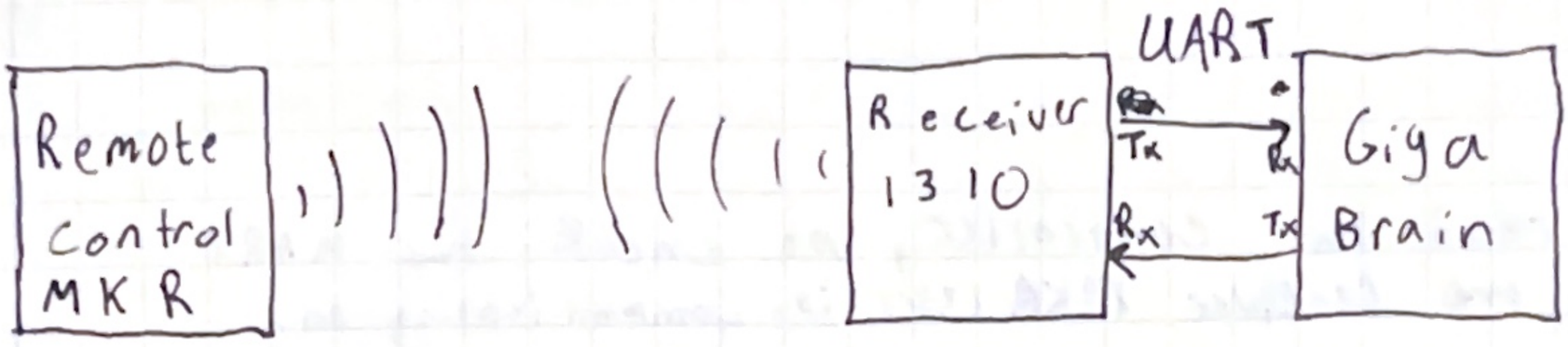
(8)

a_FinalBoat | AutoMode.h | DistanceBetween.h | GetHeading.h | HeadingError.h | LogToSD.h | M

```
#ifndef LogToSD_h_
# define LogToSD_h_

#include <SD.h>

int sampleNum = 1;

void logToSd(long lat, long lon, double distance, double cHead, double tHead, long gpsHead, long groundSpeed, byte SIVs)
{
    File outputfile = SD.open("data.txt", FILE_WRITE);

    if (outputfile)
    {
        outputfile.print(lat);          // Latitude
        outputfile.print(", ");
        outputfile.print(lon);          // Longitude
        outputfile.print(", ");
        outputfile.print(sampleNum);          // Name
        outputfile.print(", ");
        outputfile.print(distance);          // Distance
        outputfile.print(", ");
        outputfile.print(cHead);
        outputfile.print(", ");
        outputfile.print(tHead);
        outputfile.print(", ");
        outputfile.print(gpsHead);       // SIV
        outputfile.print(", ");
        outputfile.print(groundSpeed); // GroundSpeed from GPS
        outputfile.print(", ");
        outputfile.print(SIVs);     // Heading from GPS
    }

    outputfile.close();
    sampleNum++;
}

#endif
```

Next, we should explain how the remote control works. To achieve this, we are using point to point LoRa. However, the giga board does not have LoRa capabilities. To fix this, we are using a 1310 MKR board to recieve signals and then using a uart to send that data to the giga.

Remote control Block Diagram



Code on the Giga for Remote control

```
#ifndef RemoteMode_h_
# define RemoteMode_h_

#include "ReadController.h"
#include "UpdateMotors.h"
#include "MainObjects.h"

void remoteMode()
{
    readController();

    Serial.print(leftMotorSpeed);
    Serial.print(", ");
    Serial.print(rightMotorSpeed);
    Serial.print(", ");
    Serial.print(leftMotorSpeedTarget);
    Serial.print(", ");
    Serial.println(rightMotorSpeedTarget);

    updateMotors(leftMotorSpeedTarget, rightMotorSpeedTarget);
}

#endif
```

This method is ran when the controller reports the Remote control mode. This mode will continue to check the controller for moter speeds and update them accordingly.

Autonomous WIL

07/30/2023
Wednesday

⑨

```
61 void readController()
62 {
63   while (Serial3.available())
64   {
65       char start = Serial3.read();
66
67       if (start == 'S')
68       {
69           _read_speed();
70       }
71       else if (start == 'M')
72       {
73           _read_state();
74       }
75   }
76 }
77
78 #endif
```

To read the controller, we check the UART that the receiver MKR 1310 is communicating on.

Here, we are looking for two types of LoRa packets. 'S' stands for "Speed" and 'M' stands for "Mode".

```
1  #ifndef ReadController_h_
2  # define ReadController_h_
3
4  #include "MainObjects.h"
5  #include "UpdateMotors.h"
6
7  char leftSpeedChars[5];
8  char rightSpeedChars[5];
9
10 void _read_state()
11 {
12     char data = Serial3.read();
13
14     if (Serial3.read() == 'E')
15     {
16         if (data == '0')
17         {
18             programState = 0;
19
20             Serial.println("Switched to autonomous mode.");
21         }
22         else if (data == '1')
23         {
24             programState = 1;
25
26             Serial.println("Switched to remote control mode.");
27             updateMotors(1500, 1500);
28         }
29     }
30 }
```

If we see an 'M' we are looking for a 0 or 1.

```
32 void _read_speed()
33 {
34     uint8_t numPackets = Serial3.available();
35
36     while(numPackets < 8)
37     {
38         numPackets = Serial3.available();
39         delay(10);
40     }
41
42     for (int i = 0; i < 4; i++)
43     {
44         leftSpeedChars[i] = Serial3.read();
45     }
46     leftSpeedChars[4] = '\0';
47
48     for (int i = 0; i < 4; i++)
49     {
50         rightSpeedChars[i] = Serial3.read();
51     }
52     rightSpeedChars[4] = '\0';
53
54     if (Serial3.read() == 'E')
55     {
56         leftMotorSpeedTarget = atoi(leftSpeedChars);
57         rightMotorSpeedTarget = atoi(rightSpeedChars);
58     }
59 }
```

If we see an 's', we know we are looking
4 bytes for the left speed
4 bytes for the right speed
an 'E' for the end of the packet.

We are storing each byte into an array and then using the "atoi" method to transform ASCII to an integer value.

Autonomous WIL

07/30/2023
2nd week
coded

Code on the Receiver MKR 1310.

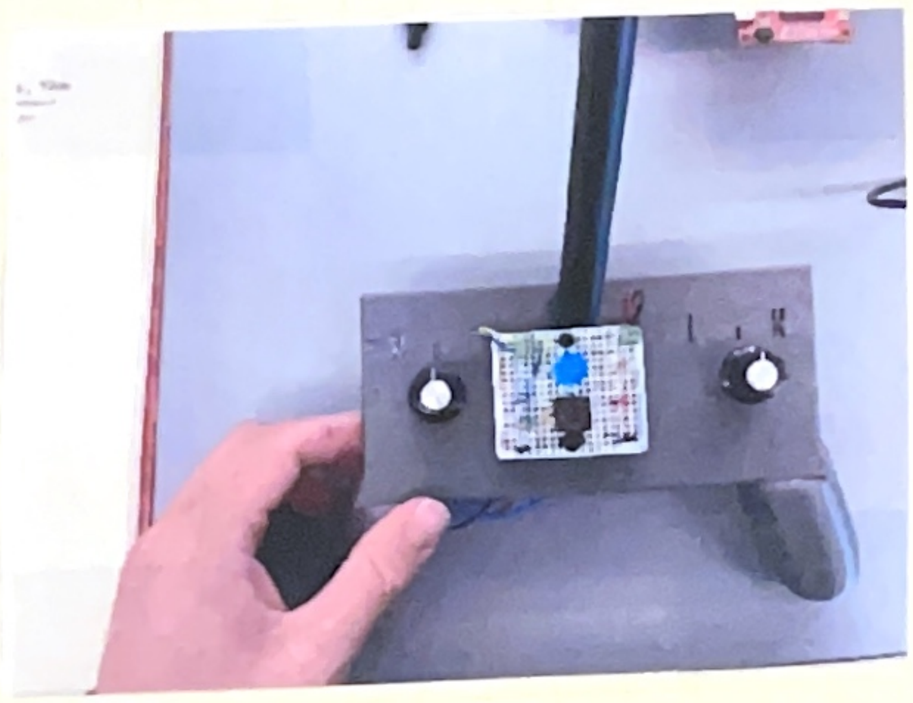
controller_receiver

```
#include <SPI.h>
#include <LoRa.h>

int leftSpeed;
int rightSpeed;
bool ledState;
char leftSpeedChar[5];
char rightSpeedChar[5];
bool validData;

void setup()
{
//  Serial.begin(9600);
  Serial1.begin(115200);// Hard ware Serial to Giga.

  if (!LoRa.begin(915E6))
  {
//    Serial.println("Starting LoRa failed!");
    digitalWrite(LED_BUILTIN, HIGH);
    while (1);
  }
  digitalWrite(LED_BUILTIN, LOW);
}
void loop()
{
  // Grab a packet
  int packetSize = LoRa.parsePacket();

  // Assuming there is a packet
  if (packetSize)
  {
    validData = 0;//Assume the data is invalid.

    // Read the first bit.
    char start = LoRa.read();

    if (start == 'S') // 'S' means we are receiving speed data
    {

      // Parse each char
      for (int i = 0; i < 4; i++)
      {
        leftSpeedChar[i] = LoRa.read();
      }
      leftSpeedChar[4] = '\0';
      for (int i = 0; i < 4; i++)
      {
        rightSpeedChar[i] = LoRa.read();
      }
      rightSpeedChar[4] = '\0';

      // Convert ASCII to INT
      leftSpeed = atoi(leftSpeedChar);
      rightSpeed = atoi(rightSpeedChar);
```

```
57      // Assuming we get the proper ending bit
58      if (LoRa.read() == 'E')
59      {
60
61        // Check to see if the data is valid
62        validData = 1;
63        if (leftSpeed > 1900)
64          validData = 0;
65        if (leftSpeed < 1100)
66          validData = 0;
67
68        if (rightSpeed > 1900)
69          validData = 0;
70        if (rightSpeed < 1100)
71          validData = 0;
72      }
73
74      if (validData)
75      {
76        // Send it out over the hardware serial
77        Serial1.print('S');
78        Serial1.print(leftSpeed);
79        Serial1.print(rightSpeed);
80        Serial1.print('E');
81      }
82    }
83
84    if (start == 'M') // 'M' means we are getting a program mode
85    {
86      char data = LoRa.read(); // Read the mode
87      if (LoRa.read() == 'E')  // Check the ending bit
88      {
89        Serial1.print('M');    // Send out over hardware serial
90        Serial1.write(data);
91        Serial1.print('E');
92      }
93    }
94
95  }
96
97 }
```

This code will receive data from the remote control and check if the values make sense.

If not, we will discard the packet.



Autonomous WIL

07/30/2023
Wesley Cooke

Code on the Remote Control.

lora_controller

```
#include <SPI.h>
#include <LoRa.h>

// leftSpeed and rightSpeed will be transmitted
int leftSpeed;
int rightSpeed;
int leftSpeedPre = 0;
int rightSpeedPre = 0;

// Readings from the two pots
float leftPot;
float rightPot;

int motorValue;
int turnBias;

// flag to update program from interrupt
bool updateProgram;

bool ledState = 1;
bool programState = 1;

// Timing Variables
long lastStateSent;
long currentTime;
```

```
void setup()
{
  Serial.begin(9600);

  analogReadResolution(10); // 10 bits on analogRead.. max of 1023
  pinMode(4,  INPUT); // Push button
  pinMode(A1, INPUT); // Potientrometers
  pinMode(A2, INPUT);
  pinMode(6, OUTPUT); // LCD

  // Interrupt for the button
  attachInterrupt(digitalPinToInterrupt(4), changeProgramMode, RISING);

  // LED.. start in "remote control mode"
  digitalWrite(6, HIGH);

  // Start LoRa comms...
  if (!LoRa.begin(915E6))
  {
    Serial.println("Starting LoRa failed!");
    digitalWrite(LED_BUILTIN, HIGH);
    while (1);
  }
}
```

```
void loop()
{
  // If it's been more than 1.5 seconds since the last state sent
  if (millis() - lastStateSent >= 1500)
  {
    // send the state again
    updateProgram = 1;
  }
```

07/30/2023
Wesley Cerda

Autonomous WIL


```
// If in remote control mode
if (ledState == 1)
{
  // Read the pots
  leftPot = analogRead(A1) / 1023.0;
  rightPot = analogRead(A2)/ 1023.0;

  // Rescale reading. Motor Range is 1100 - 1900.
  motorValue = (leftPot) * 800 + 1100;
  // Rescale reading. Bias is between -400 and 400.
  turnBias   = (rightPot) * 800 - 400;

  leftSpeed = motorValue - turnBias;
  rightSpeed = motorValue + turnBias;

  // Bounds checking...
  if (leftSpeed > 1900)
    leftSpeed = 1900;
  if (leftSpeed < 1100)
    leftSpeed = 1100;

  if (rightSpeed > 1900)
    rightSpeed = 1900;
  if (rightSpeed < 1100)
    rightSpeed = 1100;

  // Don't send if we already sent this value
  if (leftSpeed != leftSpeedPre or rightSpeed != rightSpeedPre)
  {
    LoRa.beginPacket();
    LoRa.print('S');
    LoRa.print(leftSpeed);
    LoRa.print(rightSpeed);
    LoRa.print('E');
    LoRa.endPacket();
  }

  // Keep track of the previous values
  leftSpeedPre = leftSpeed;
  rightSpeedPre = rightSpeed;
}

// We need to change the mode of the program
// and send it to the receiver.
if (updateProgram)
{
  // Update the LED with the current state.
  digitalWrite(6, ledState);

  // Send the state to the receiver
  LoRa.beginPacket();
  LoRa.print('M');
  LoRa.print(ledState);
  LoRa.print('E');
  LoRa.endPacket();

  // Update the last time sent and clear the flag.
  lastStateSent = millis();
  updateProgram = 0;
 }
}

void changeProgramMode()
{
  // When the button is pressed, this changes the state of the led
  // and sets the updateProgram flag.
  ledState = !ledState;
  updateProgram = 1;
}
```

Autonomous WIL

07/30/2023
Wesley Cocke

Results from Lake Olmstead

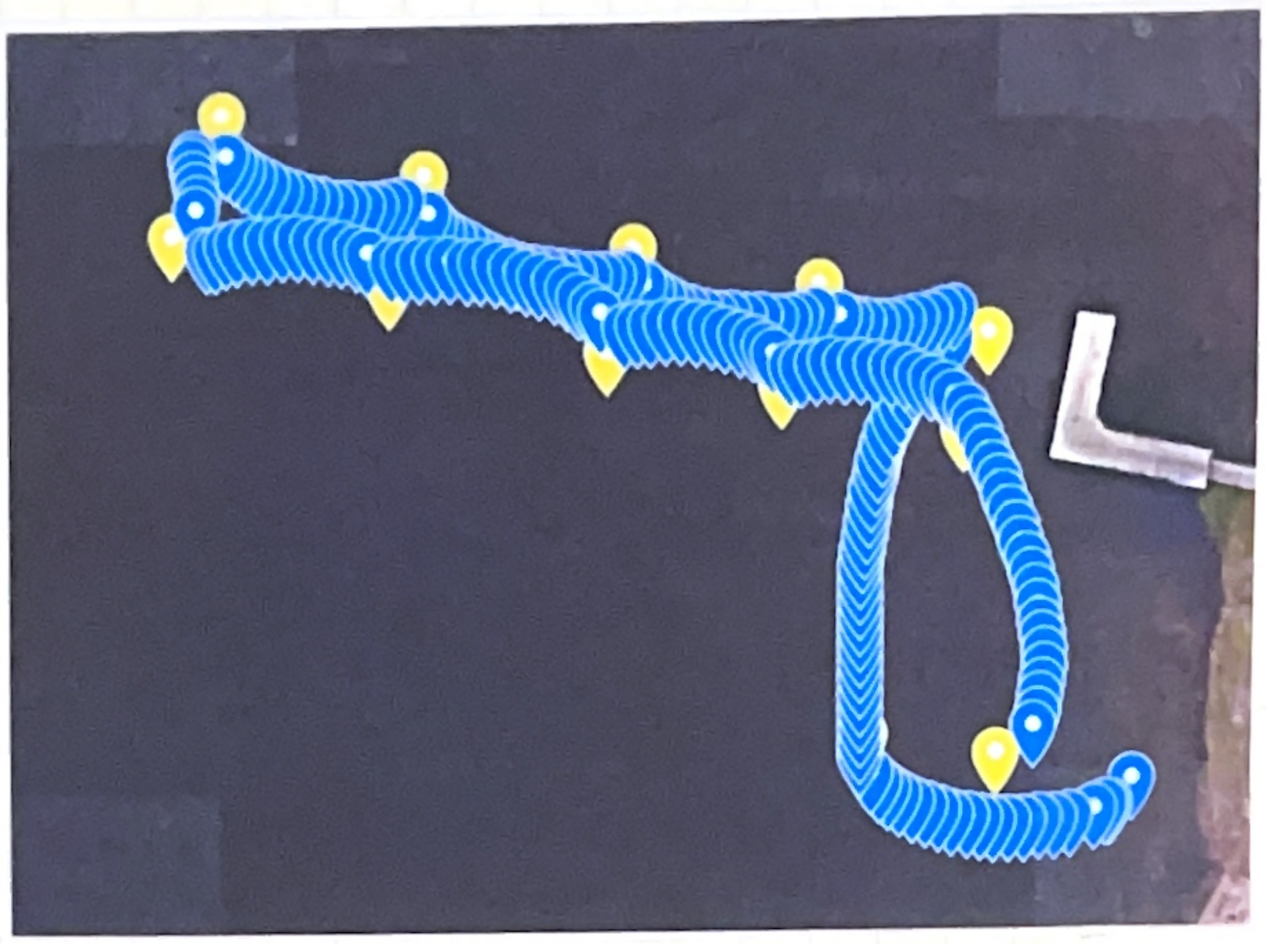


Figure 1

When the heading of WIL is not properly aligned with north, WIL will arc between waypoints.

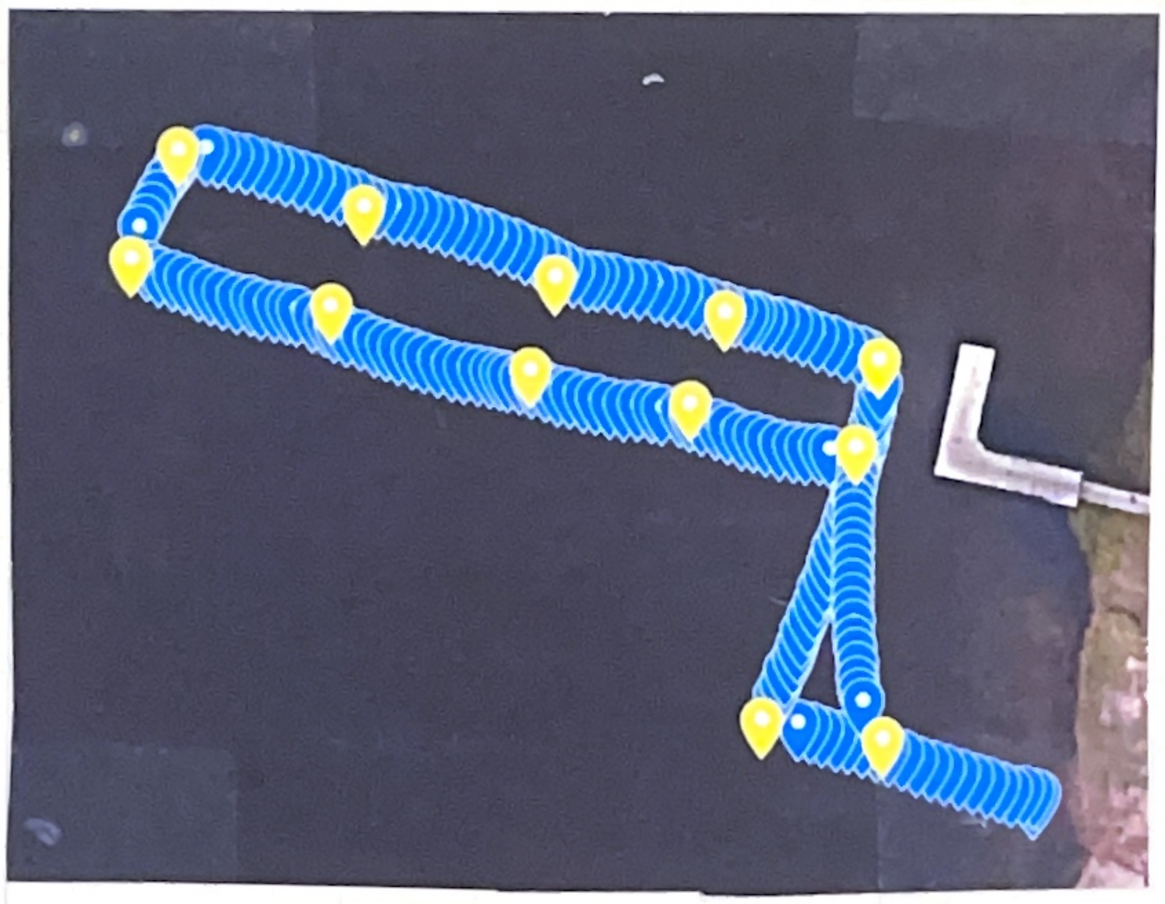


Figure 2

When the heading of WIL is better aligned with north, WIL will take a more straight path between waypoints.

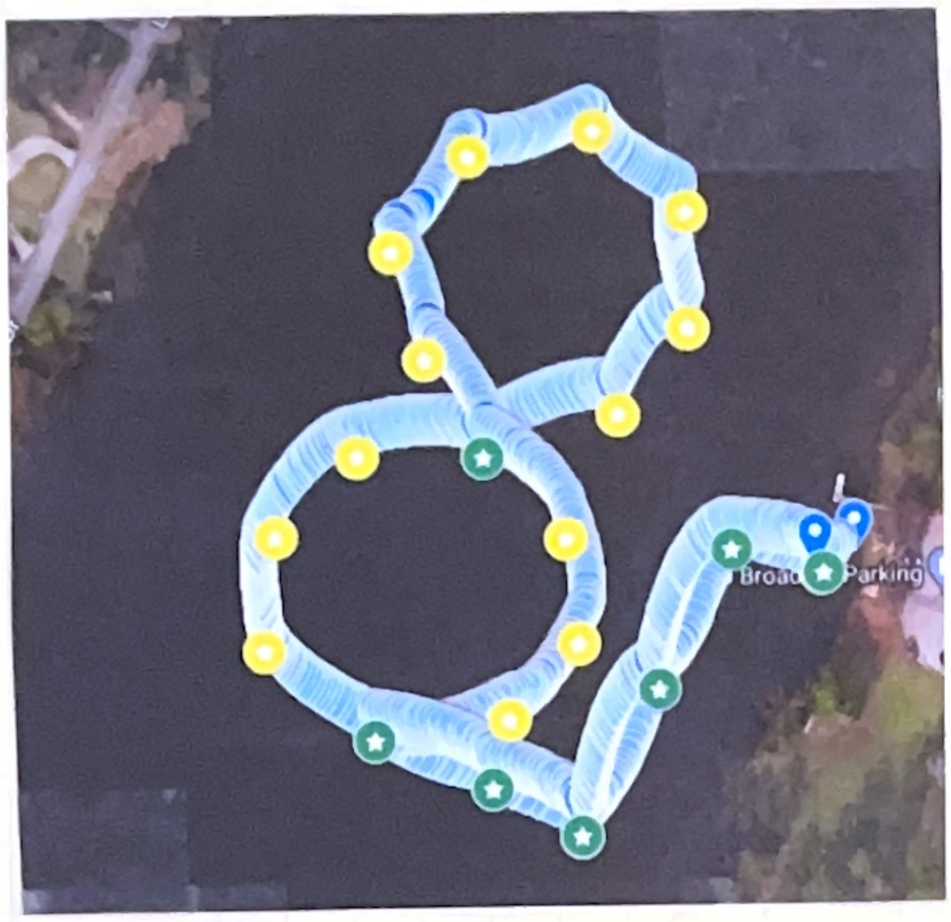


Figure 3

Testing WIL on a Larger figure 8 Path.

Figure 4

Painting the outriggers

Figure 5

Painting the hull

Pictures by Erin Schmidt

Left to Right: Wesley Cooke, Brycen Havens, Dr. Andy Hauger