Mechatronics 3 Assignment 1 - Assistive Technology Design

Georgia Tovich, 500488214 Taj Wedutenko, 500457324 Finn Wilson, 500456925

1 Concept of Operation

1.1 Scope of Project

Quadriparesis can be caused by a multitude of diseases ranging from an infection and damage to the nervous system to neuromuscular diseases such as muscular dystrophy [3]. Although wheelchairs controlled by hand movements are widely available, not all people with quadriparesis can use them and this project will aim to address this gap. Although it will not be usable by all people with quadriparesis, this project will aim to create a proof of concept for a head controlled wheelchair for people who are able to complete head movements but not move the rest of their body.

The project will consist of a headset that will be able to detect head movements using the six-degrees of freedom inertial movement unit (IMU) and from these head movements control a wheelchair.

1.2 User Manual

The project has been designed to maximise ease of use. A design concept drawing can be seen in Figure 1 to the right. For the wheelchair component the user needs to sit in the wheelchair, which should have two powered motors to allow for speed control and direction control. The headset component will require the user to learn commands to control. Both modules will be battery powered, with sufficient capacity to allow the user to travel for an extended period of time. There will be a visual display integrated into the wheelchair such that the user can monitor the connection and battery status, with a warning showing if either module reaches low power. So that the user can move their head to look around as well, they will have to bite on a mouthpiece containing a pressure sensor to give commands to the wheelchair. If there is no biting head movements will not be detected as user commands.

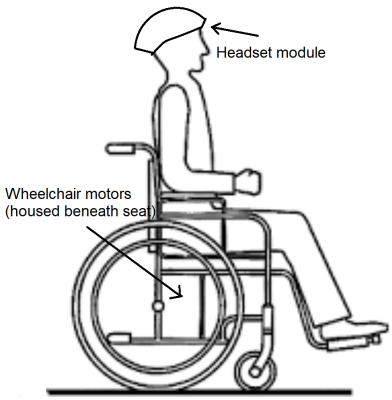


Figure 1: Design concept for project, adapted from the United Nations [6]

For the wheelchair control, a forward head tilt (afterwards returning to a neutral position within 1.5 seconds otherwise this will detect a backwards head tilt) will increase the speed of the wheelchair in a forwards direction. A backwards head tilt will decrease the velocity of the wheelchair and if velocity is negative or zero the wheelchair will reverse. There are three reversal speeds and five forward speeds. Just like the forward tilt, the user should return to a neutral head position within 1.5 seconds. The turning instructions for the wheelchair are commanded by a sideways head tilt, tilting to the left while reversing or going forwards will turn the wheelchair to the left for the user and vice versa for a head tilt to the right. The turning command is continuous so turning will continue as long as the head is tilted to the side.

There is also the option for the user to stop regardless of their speed and to execute this command the user needs to turn their head to either side once. Auto-stopping is also triggered if the user gets within a certain distance of another object, the user does not need to do anything to trigger this, this is just a safety precaution.

2 System Architecture

2.1 High Level Code Flowchart

The project consists of two main modules, the wheelchair module and the headset module. A combination of the overall high level code design is seen in Figure 2.

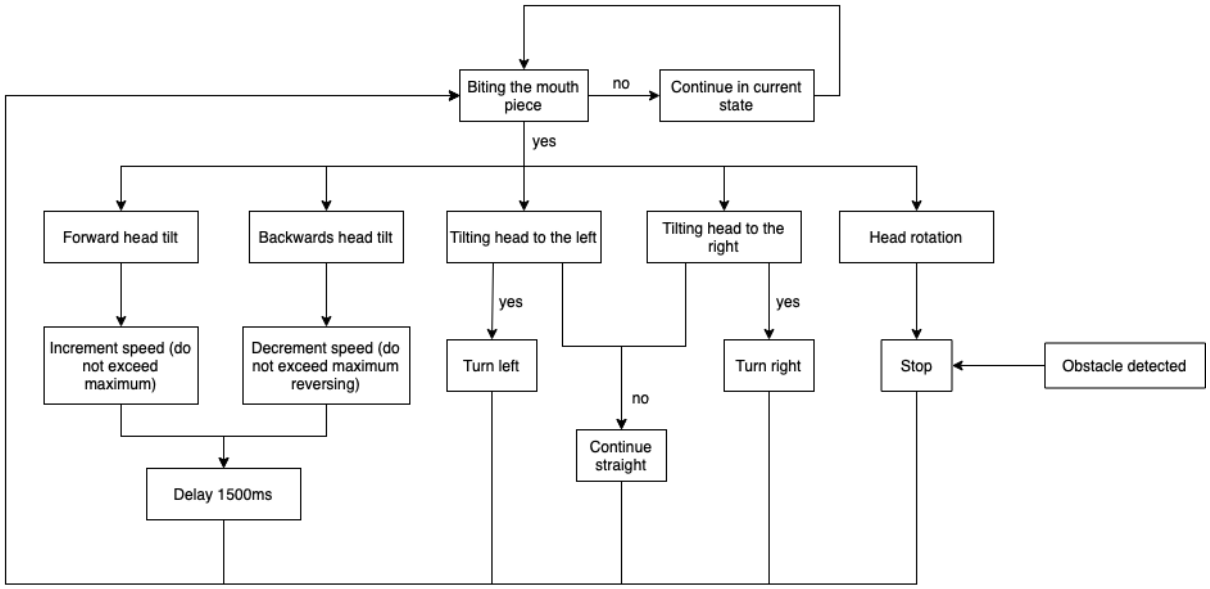


Figure 2: High level code flowchart

2.2 List of parts

Table 1: List of parts

Part	Number
Arduino Nano 33 IoT	1
Arduino Uno	1
HM-10 - bluetooth module	2
KR3160 - 2 wheel drive motor chassis robotics kit	1
L293D - motor driver shield	1
XC3735 - IR distance sensor	1
XC3738 - thin-film pressure sensor	1
Small portable battery pack	2
Male USB header	1
Breadboard	3
Connection wires	$\sim$

2.3 Setup Guide

1. Equip the headset, ensuring that it is securely in place.

2. Power on both modules.
3. Press the reset button on both modules to ensure that their states are reset.
4. Ensure that both lights on the HM-10 are solid, indicating that a connection has been established.
5. Begin giving commands to the wheelchair via the headset movements.

2.4 Wheelchair Module

The wheelchair module is made up of a model wheelchair, an Arduino Uno, a motor driver shield, an HM-10 Bluetooth Module, and an IR distance sensor. The motor driver shield is attached to the Arduino Uno, which provides dedicated motor controllers. These controllers directly connect to the DC motors that are used to move the model wheelchair. The HM-10 and IR distance sensor are connected to analogue input pins on the Arduino Uno as seen in Figure 3 - in the case of the HM-10, these are configured as digital serial pins using the Software Serial package in the Arduino library.

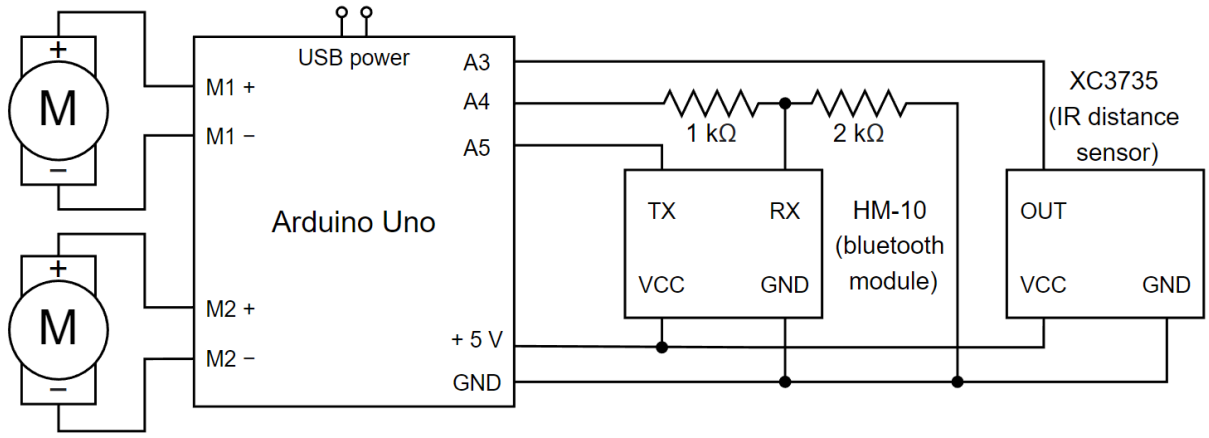


Figure 3: Wheelchair module wiring diagram

2.5 Headset Module

The headset module consists of an Arduino Nano, HM-10 Bluetooth module and a force sensitive resistor (FSR). The Arduino Nano is used to collect the head movements from the user and these are relayed to the model wheelchair through the HM-10 Bluetooth module. The HM-10 bluetooth module is connected to the Arduino Nano via the dedicated serial transmit (TX) and receive (RX) pins and the FSR is connected to the board through an analog pin set to input data, as seen below in Figure 4.

3 Data Acquisition

Various forms of data were acquired by the Arduino Nano and the Arduino Uno. The Arduino Nano collected the gyroscope and accelerometer data with the six degrees of freedom IMU as well as a voltage from the pressure sensor to detect whether or not the mouth piece is being bitten. The Arduino Uno detects a distance up to 1.5m and returns an analog signal to the board to detect whether the wheelchair is about to collide with something and stop suddenly.

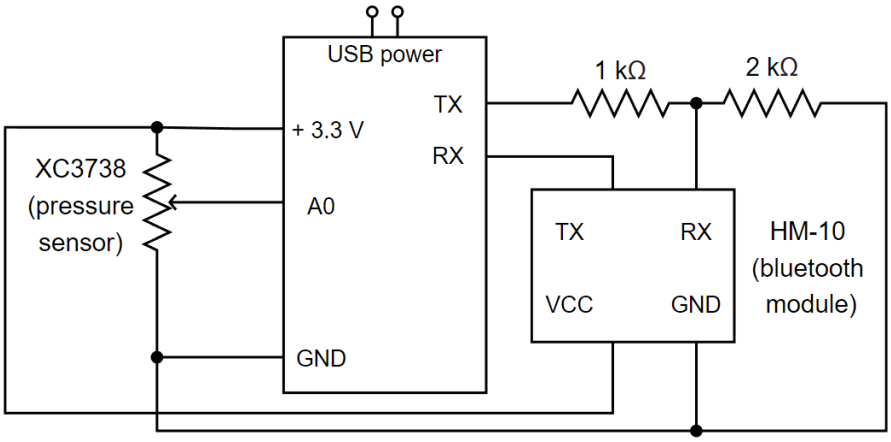


Figure 4: Headset module wiring diagram

To detect forward and backwards movement the gyroscope on the Arduino Nano is used by detecting its' pitch, and this collected raw data can be seen in Figure 5a. To detect continuous turning state of the wheelchair the acceleration of the Arduino Nano in the y-direction was measured and this raw data can be seen in Figure 6a. This raw data was later processed as discussed in Section 4 before being sent as a command to the wheelchair.

The biting pressure sensor outputs a voltage to an analog pin on the Arduino Nano. This can be read in as raw data and very little signal processing was required for this, a threshold was simply set, and above a certain voltage (while biting) commands can be sent to the wheelchair.

3.1 Testing

The first step in designing and building each project modules, was collecting the raw data from the IMU and seeing if it was being plotted as expected. This was done printing to serial monitor and the serial plotter in the Arduino IDE and seeing if the expected data was being received. After that, the raw data was mapped to convert the raw data to an angle using the gyroscope and accelerometer. Our IMU was very accurate and sufficient in detecting head movements of the user to give commands to the wheelchair, further testing and calibration also had to be completed for the serial communication between the wheelchair and headset modules and on the hardware.

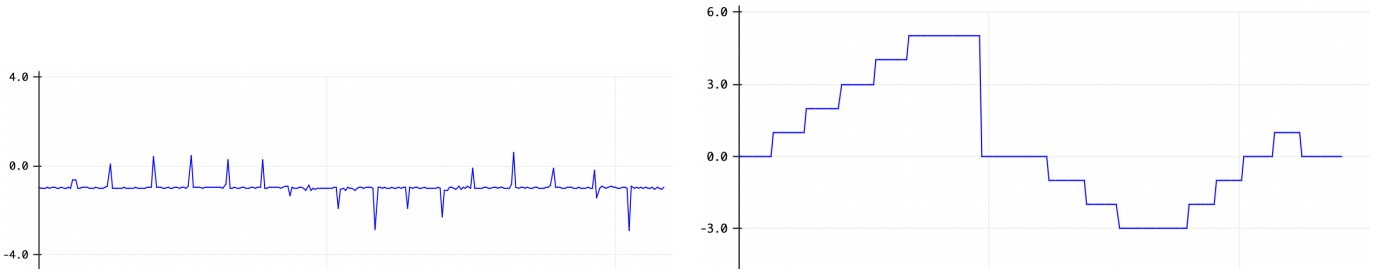
To test the serial communication between modules first the HM-10s had to be configured and it was checked that they had connected by looking at their LED and ensuring that it went solid. Once this connection was established different simple serial messages such as "hello" were sent from the controller (Arduino Nano) to the peripheral (Arduino Uno) and it was checked that these messages were getting received and printed to the serial monitor for the Arduino Uno. Once this was completed, further testing was then completed but instead of sending hard-coded messages, a struct was created and the processed IMU data was sent to the Arduino Uno and it was checked that these messages were getting received and again being printed to the serial monitor. It was quickly discovered that the messages were sending incorrectly, and after some research it was found that this was due to an int being 4 bytes on the Arduino Nano, whilst only 2 bytes on the Arduino Uno. Hence, the values were stored as short (2 bytes) on the Arduino Nano, which fixed the issue. Through additional testing it was also decided that a 60ms delay should be added between each sent signal to ensure that the Arduino Uno could finish running code to again reduce misalignment of the sent struct.

During testing, hard-coded values were sent to the wheelchair module to calibrate and ensure that each motor was lined up to run at the same speed. As expected there was slight variation between the motors

and by sending different values to each motor they could be calibrated to ensure that the wheelchair could drive in a straight line when intended and turn as commanded. Future development could see additional IR sensors placed on the sides of the wheelchair and added motor encoders to close the system and ensure that the wheelchair is responding as expected at all times.

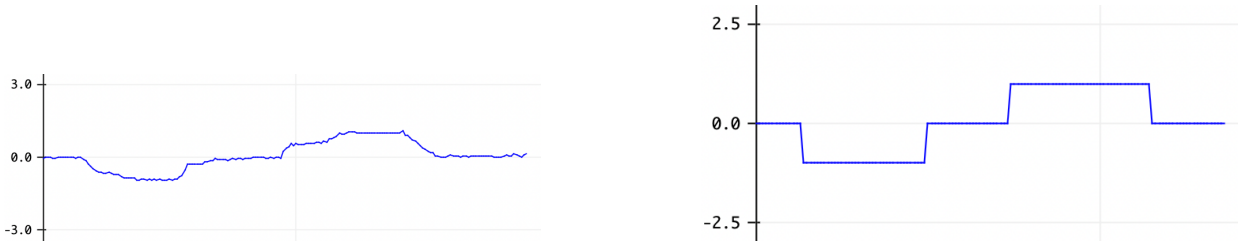
4 Signal Processing

In Section 3 the data acquisition of the raw user inputs were described in detail and this data can be seen below. As this data is not very clear and is not easy to discern, this raw data was processed to convert it into clear commands for the wheelchair module. For the turning, speed and auto-stop functions the data was first mapped to convert it to an angular velocity. To do this, the map() function as part of the Arduino "Math" library was used. This data was still did not offer much in the way of user commands, so the raw angular velocities were then quantised, so that between specific bounds certain commands would be triggered. This processed data can be seen in Figures 5b, 6b and 7b.



(a) Raw Speed Data (y-axis of gyroscope) **(b)** Processed Speed Data (output to wheelchair)

Figure 5: Raw and processed speed data



(a) Raw turn data (y-axis of accelerometer) **(b)** Processed turn data (output to wheelchair)

Figure 6: Raw and processed turning data

(a) Raw stop data (y-axis of accelerometer) **(b)** Processed stop data (output to wheelchair)

Figure 7: Raw and processed auto-stop data

5 Communication

5.1 Bluetooth Communication Between HM-10 Modules

The HM-10 modules communicate using serial UART over Bluetooth 4.0. This communication protocol was selected as it is able to provide sufficient bandwidth to send the movement instructions from the headset to the wheelchair. Furthermore, the Arduino Uno did not have sufficient pins to attach a WiFi module to ruling this out as a communication method. Using Bluetooth also eliminates the need for a router, improving portability, whilst still providing more than enough range (up to 30 m) for communication between the headset and wheelchair.

A connection was established between the two HM-10 modules via the use of AT (Hayes) commands, which directly connect the devices through their UUIDs. The connection was configured as full duplex, with the headset (Nano) configured as the controller and the wheelchair (Uno) as the peripheral, although simplex was only ever used. The Arduino Nano 33 IoT features onboard hardware serial pins, which allows for dedicated serial processing. The Arduino Uno does not; however, the digital pins can be configured to work as serial pins with the use of a software package that handles the processing through the ATmega328P MCU.

Following the initial setup, the modules automatically connected each time they were turned on. When the user bites down on the pressure sensor to indicate that commands are to be sent, then a struct containing the wheelchair speed, wheelchair direction, and auto stop Boolean is converted into binary, sent over serial at 9600 baud rate, and then converted back into the original format. Due to the difference in bytes allocated to integer values between the Nano's SAMD21 32bit MCU and the Uno's ATmega328P, values on the Nano had to be stored as shorts (2 bytes), and ints (2 bytes) on the Uno.

5.2 Bandwidth Estimation

The bandwidth can be estimated based on the total amount of data to be replicated (TD) in gigabytes, the length of the replication window time in hours (RWT) and the data de-duplication ratio (DR). For this estimation a data de-duplication ratio value of 100 will be used. Once this data is acquired the following equation can be used to estimate the network bandwidth in Mbps/second.

$$(TD \cdot (\frac{100}{DR} \cdot 8192)/(RWT \cdot 3600) = Bandwidth$$

BLE 4.0 has a bandwidth of 1 Mbit/sec, which would be the maximum bandwidth we could access for communication between the headband (motion sensor) and wheelchair (movement) modules.

Bibliography

[1] Arduino. Accessing accelerometer data for the nano 33 iot. Available at `https://docs.arduino.cc/tutorials/nano-33-iot/imu-accelerometer` (2022/08/28), .

[2] Arduino. Accessing gyroscope data for the nano 33 iot. Available at `https://docs.arduino.cc/tutorials/nano-33-iot/imu-gyroscope` (2022/08/28), .

[3] Healthline. Quadriparesis. Available at `https://www.healthline.com/health/quadriparesis` (2022/08/28).

[4] Jaycar. Wall dodging robot. Available at `https://github.com/Jaycar-Electronics/Wall-Dodging-Robot` (2022/08/28).

[5] C. Keef. Bluetooth hm10. Available at `https://www.mrswirlyeyes.com/tutorials/bluetooth_hm_10` (2022/08/28).

[6] U. Nations. Accessibility for the disabled - a design manual for a barrier free environment. Available at `https://www.un.org/esa/socdev/enable/designm/AD5-02.htm` (2022/08/28).

Code Appendix

Nano Bluetooth Setup

```
#define BAUDRATE 9600

char c = ' ';
boolean new_line = true;

void setup() {
  // Start Serial Monitor for feedback
  Serial.begin(BAUDRATE);

  // HM-10 default speed in AT command mode
  Serial1.begin(BAUDRATE);

  Serial.println("Enter AT commands:");

}

void loop() {
  // Keep reading from HM-10 and send to Arduino Serial Monitor
  if (Serial1.available())
    Serial.write(Serial1.read());

  // Keep reading from Arduino Serial Monitor and send to HM-10
  if (Serial.available()) {

    // Read from the Serial buffer (from the user input)
    c = Serial.read();

    // Do not send newline ('\n') nor carriage return ('\r') characters
    if(c != 10 && c != 13)
      Serial1.write(c);

    // If a newline ('\n') is true; print newline + prompt symbol; toggle
    if (new_line) {
      Serial.print("\r\n>");
      new_line = false;
    }

    // Write to the Serial Monitor the bluetooth's response
    Serial.write(c);

    // If a newline ('\n') is read, toggle
    if (c == 10)
      new_line = true;
  }
}
```

Uno Bluetooth Setup

```
// Serial communication with Bluetooth HM-10
// Uses serial monitor for communication with Bluetooth HM-10
//
//  Arduino to HM-10 connections
//  Arduino pin 2 (TX) to voltage divider then to HM-10 RX
//  Arduino pin 3 to HM-10 TX
//  Connect GND from the Arduiono to GND on the HM-10
//
// When a command is entered in to the serial monitor on the computer
// the Arduino will relay the command to the HM-10


// Library to make a Software UART
#include <SoftwareSerial.h>

#define RX A4
#define TX A5

#define BAUDRATE 9600

char c = ' ';
boolean new_line = true;

// Instantiation of a Software UART
SoftwareSerial BTSerial(RX, TX); // (RX, TX)

void setup() {

  // Start Serial Monitor for feedback
  Serial.begin(BAUDRATE);

  // HM-10 default speed in AT command mode
  BTSerial.begin(BAUDRATE);

  Serial.println("Enter AT commands:");
}

void loop() {

  // Keep reading from HM-10 and send to Arduino Serial Monitor
  if (BTSerial.available())
    Serial.write(BTSerial.read());

  // Keep reading from Arduino Serial Monitor and send to HM-10
  if (Serial.available()) {

    // Read from the Serial buffer (from the user input)
```

```
    c = Serial.read();

    // Do not send newline ('\n') nor carriage return ('\r') characters
    if (c != 10 && c != 13)
      BTSerial.write(c);

    // If a newline ('\n') is true; print newline + prompt symbol; toggle
    if (new_line) {
      Serial.print("\r\n>");
      new_line = false;
    }

    // Write to the Serial Monitor the bluetooth's response
    Serial.write(c);

    // If a newline ('\n') is read, toggle
    if (c == 10)
      new_line = true;
  }
}
```

Nano Code (Signal Acquisition, Processing and Transmitting)

```
 /*-----------------------------------------------------
 Definitions
 -----------------------------------------------------*/

#include <Arduino_LSM6DS3.h>

// Define baudrate
#define BAUDRATE 9600

// Length of serial message
#define SERIAL_LENGTH 6

// Pin that pressure sensor is connected to
#define PRESSURE_PIN A0

/*-----------------------------------------------------
 IMU setup
 -----------------------------------------------------*/

// Commands to be outputted to wheelchair
// Command for speed [-3 -> 5]
short wheelchairSpeed = 0;

// Command for turning [0 for no turn, 1 for left turn, 2 for right turn]
short wheelchairTurn = 0;
```

```
// Command for autostop [0 for default, 1 for autostop]
short autostop = 0;

/*-------------------------------------------------------
 Serial setup
 -------------------------------------------------------*/
// Data needed to control the motors
struct moveVals {
  short velMagnitude; // 2 bytes
  short velDirection; // 2
  short stopTrue;     // 2
};

// Struct is overlaid onto an array for conversion from
// the binary serial message
union serialOutput {
  moveVals moveData;
  byte serialMessage[SERIAL_LENGTH];
};

// Working instance of union
serialOutput outputData;

/*-------------------------------------------------------*/

int speedChange = 0;

void setup() {

  // Start serial monitor for feedback
  Serial.begin(BAUDRATE);

  // Start HM-10
  Serial1.begin(BAUDRATE);

  // Start the IMU
  IMU.begin();

  // set DAC resolution to 10-bit:
  analogWriteResolution(10);
  pinMode(PRESSURE_PIN, INPUT_PULLUP);


}

void loop() {
```

```
  //if (takeCommandsTrue()) {

    findIfTurning();
    findSpeed();
    findStop();
    //Serial.println(wheelchairSpeed);
    //Serial.println(wheelchairTurn);
    Serial.println(autostop);
    sendData(wheelchairSpeed, wheelchairTurn, autostop);

    if (speedChange == 1){
      delay(1000);
      speedChange = 0;
    }
    else {
      delay(100);
    }
  //}

}

/*-----------------------------------------------------
 If the user wishes to give commands, they bite on a
 pressure sensor. This function detects if they are biting
 -----------------------------------------------------*/
int takeCommandsTrue()
{
  // Default to not biting
  int isBiting = 0;

  // Read in the voltage from the pin
  int level = analogRead(PRESSURE_PIN);
  float pressure = (level * 100) / 1024.0;
  //Serial.println(pressure);

  // If high, pressure sensor is bitten
  if (pressure > 80.0)
  {
    isBiting = 1;
  }

  return isBiting;
}

/*-----------------------------------------------------
 Function to send movement commands to the wheelchair via
 bluetooth serial.
 -----------------------------------------------------*/
void sendData(short velMagnitude, short velDirection, short stopTrue)
```

```
{
  // Assign data to struct
  outputData.moveData.velMagnitude = velMagnitude;
  outputData.moveData.velDirection = velDirection;
  outputData.moveData.stopTrue     = stopTrue;

  // Send data to the bluetooth module;
  Serial1.write(outputData.serialMessage, SERIAL_LENGTH);
}

/*-----------------------------------------------------------
 Function to determine if the user is indicating for the
 wheelchair to turn left or right
 -----------------------------------------------------------*/
void findIfTurning()
{
  // Set up floats for accelerometer data used for direction
  float x, y, z;
  int degreesY;
  IMU.readAcceleration(x, y, z);

  //Serial.println(y);
  //Serial.println(x);

  // Always have default of wheelchair not turning
  wheelchairTurn = 0;

  // Convert to degrees
  if (y > 0.1) {
    y = 100 * y;
    degreesY = map(y, 0, 97, 0, 90);

    // If head is tilted to the right set turn to the right
    if (degreesY > 15) {
      wheelchairTurn = 1;
    }
  }
  if (y < -0.1) {
    y = 100 * y;
    degreesY = map(y, 0, -100, 0, 90);

    if (degreesY > 15) {

      wheelchairTurn = -1;
    }
  }

  if (wheelchairTurn < -1 || wheelchairTurn > 1){
```

```
    wheelchairTurn = 0;
  }
}

/*-----------------------------------------------------------
 Function to determine if the user is needing to stop quickly
 -----------------------------------------------------------*/

void findStop() {

  float xStop, yStop, zStop;
  IMU.readGyroscope(xStop, yStop, zStop);

  if (xStop > 100.0 || xStop < -100.0) {
    wheelchairTurn = 0;
    wheelchairSpeed = 0;
    autostop = 1;
  }
  else {

    autostop = 0;
  }


}

/*-----------------------------------------------------------
 Find if the user is indicating to increment or decrement
 the speed.
 -----------------------------------------------------------*/
void findSpeed()
{
  // Set up floats for the gyroscope data which is used for speed
  float xGyro, yGyro, zGyro;
  IMU.readGyroscope(xGyro, yGyro, zGyro);

  // Forward speeds
  if (yGyro > 75.0) {

    // Increase the wheelchair speed or slow down reverse
    //if (wheelchairSpeed < 5) {
    wheelchairSpeed = min(wheelchairSpeed+1,5);
    speedChange = 1;
    //delay(2000);
    //}
  }

  // Situation for reverse speed
  if (yGyro < -75) {
```

```
    // Decrease wheelchair speed or reverse faster
    //if (wheelchairSpeed > -3) {

    wheelchairSpeed = max(wheelchairSpeed-1,-3);
    speedChange = 1;
    //delay(2000);
    //}
  }

  if (wheelchairSpeed < -3 || wheelchairSpeed > 5){

    wheelchairSpeed = 0;
  }

}
```

Uno Code (Receiving, Motor Control, Obstacle Detection)

```
// Library to make a Software UART
#include <SoftwareSerial.h>

// Pin locations
#define RX A4
#define TX A5

#define BAUDRATE 9600

// Length of serial message
#define SERIAL_LENGTH 6

// Data needed to control the motors
struct moveVals {
  int velMagnitude; // 2 bytes
  int velDirection; // 2
  int stopTrue;     // 2
};

// Struct is overlaid onto an array for conversion from
// the binary serial message
union serialInput {
  moveVals moveData;
  byte serialMessage[SERIAL_LENGTH];
};

// Working instance of union
serialInput inputData;

byte serialData[SERIAL_LENGTH];
```

```
boolean newData = false;
boolean askForData = true;

// Instantiation of a Software UART
SoftwareSerial BTSerial(RX, TX); // (RX, TX)

void setup() {

  // Start Serial Monitor for feedback
  Serial.begin(BAUDRATE);
  while(!Serial);

  // Start HM-10
  BTSerial.begin(BAUDRATE);
  while(!BTSerial);

  Serial.println("Receiving data:");
}

void loop() {
  requestData();
  receiveData();
  displayData();

  delay(1000);
}

void requestData()
{
  if (askForData)
  {
    Serial.print("<M>");
    askForData = false;
  }
}

void receiveData()
{
  if (BTSerial.available() < SERIAL_LENGTH)
  {
    Serial.println("Error");
  }

  for (byte n = 0; n < SERIAL_LENGTH; n++)
  {
    serialData[n] = BTSerial.read();
  }

  for (byte n = 0; n < SERIAL_LENGTH; n++)
```

```
  {
    inputData.serialMessage[n] = serialData[n];
  }

  newData = true;
}

void displayData()
{
  if (newData == false)
  {
    return;
  }
  Serial.print("<Data: ");
  Serial.print(inputData.moveData.velMagnitude);
  Serial.print(" ");
  Serial.print(inputData.moveData.velDirection);
  Serial.print(" ");
  Serial.print(inputData.moveData.stopTrue);
  Serial.println(">");
  newData = false;
  askForData = true;
}
```