Imported Libraries

```
#include <Wire.h>
#include "Sensor.h"
#include <VL53L1X.h>

VL53L1X lox1 = VL53L1X();
VL53L1X lox2 = VL53L1X();
VL53L1X lox3 = VL53L1X();

Sensor Sensor1;
Sensor Sensor2;
Sensor Sensor3;

int Threshold = 10;
```

The only two libraries that are imported that were not created by me is the VL53L1X library and the wire library. The VL53L1X library allows easier interfacing with the sensors, and the wire library allows communication with I^2C devices.

Sensors

In the program, each Sensor is a class. The reason classes are used for sensors is because the methods that each sensor runs are the same, so it is more efficient code to have each sensor be different instances of the same class. Since these variables would only have to be typed once, that means that the code is significantly easier to debug because there will be no typing error that would be hard to spot, for example, if a pin is accidentally set for the wrong sensor. This is adequate for the task because every sensor works independently, so doing this just simplifies the code.

The code of the sensor class is below:

```
#include <VL53L1X.h>
#include "moving_average.h"
class Sensor
{

  public:
  int avg;
    void Initialize(int SENSOR_PIN, VL53L1X ID, uint8_t ADDRESS, String SensorName){
        Sensor = ID;
        pin = SENSOR_PIN;
        SensorAddress = ADDRESS;
        Name = SensorName;
        Data.StartAvg();
        pinMode(pin, OUTPUT);
        digitalWrite(pin,LOW);
        delay(50);
      }
      void Startup() {
        digitalWrite(pin, HIGH);
        delay(150);
        Sensor.init();
        delay(100);
        Sensor.setAddress(SensorAddress);
      }
      void DataStart() {
        Sensor.setDistanceMode(VL53L1X::Long);
        Sensor.setMeasurementTimingBudget(50000);
        Sensor.startContinuous(50);
        Sensor.setTimeout(100);
      }
      void Update() {
        Serial.print(Name + ": ");
        avg = Data.add(Sensor.read());
        Serial.print(avg);
        if (Sensor.timeoutOccurred()) { Serial.print(" TIMEOUT"); }
        Serial.println();
        delay(50);
      }
     private:
      int pin;
      MovingAverage<5> Data;
      VL53L1X Sensor;
      uint8_t SensorAddress;
      String Name;
};
```

The sensors work by using an I^2C protocol, meaning each sensor, which is a slave in the system, has a clock signal and a data signal. The Arduino is the master in the system. The way the master differentiates between the slaves is by address, and in this case, all of the sensors

come with the same default address. That is why I have a separate function for immediate initialization and for starting to collect data. In order to assign an address to a sensor, I need the other sensors to either have been already assigned or have their pin set to low (turned off).

Moving Average

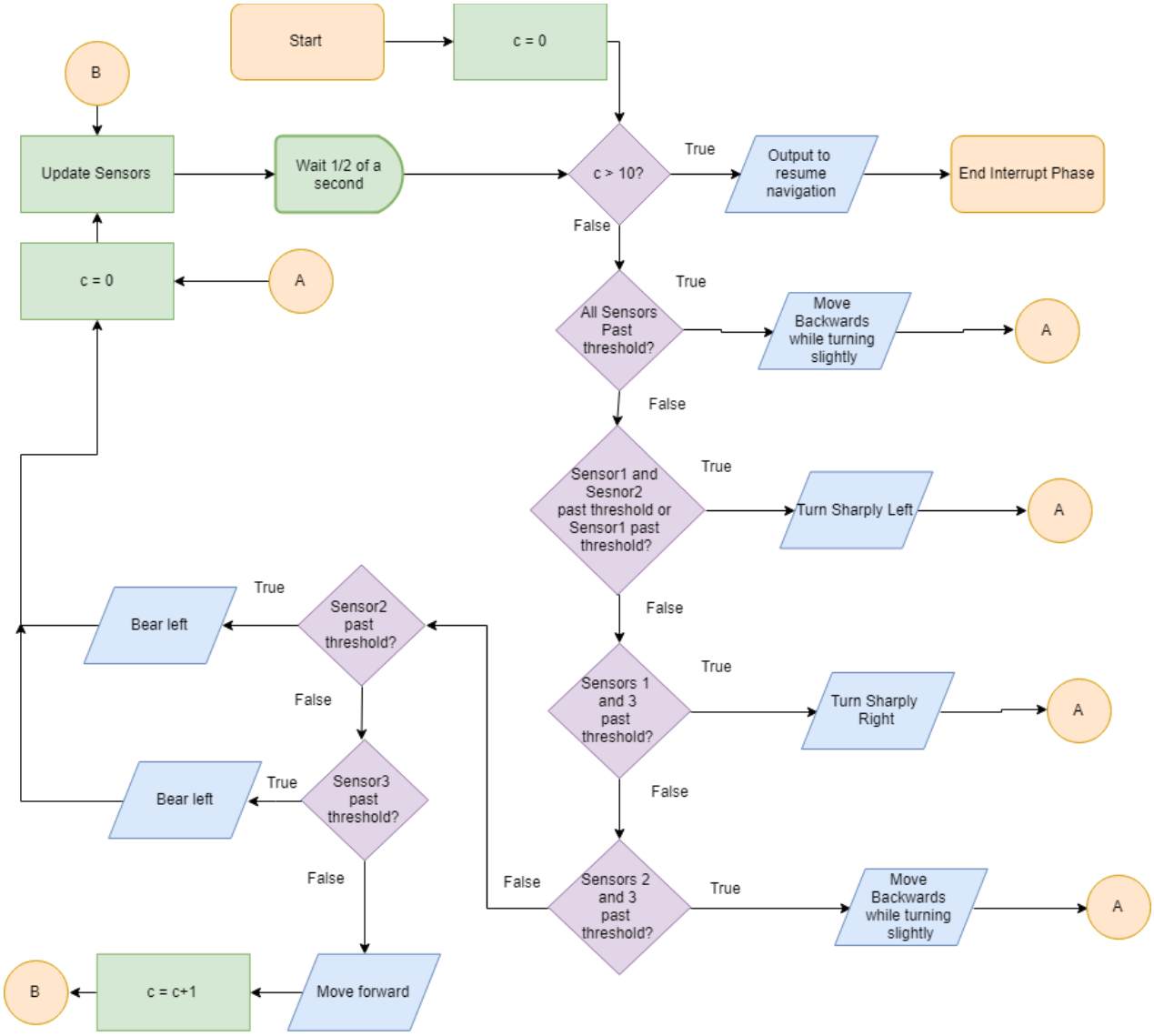
Another class is used for the moving average of each sensor. Again, this is simply to keep the code cleaner, as all that is important for the sensor level is that the newest reading is stored and an average is calculated.

The way that sensor values are stored is by using an array. The array is constantly updated with every single sensor reading, meaning that the oldest value that was recorded is kept track of with a variable, and then the oldest variable is replaced with the newest. An artificial pointer variable is used to keep track of the oldest variable in the array and is reset so that no values are assigned outside of the array. The code for that is below, in two separate images:

```
template <int N> class MovingAverage
{
public:

    void StartAvg(){
      sum = 0;
      p = 0;
        for (int i = 0; i < N; i++) {
            data[i] = 0;
            sum += data[i];
        }
    }


    int add(int new_sample)
    {
        sum = (sum - data[p]) + new_sample;
        data[p] = new_sample;
        if (p >= N){
            p = 0;
        }
        else {
        p += 1;
        }
        return sum / N;
    }

private:
    int data[N];
    unsigned int sum;
    int p;
};
```

Core Algorithm

The core algorithm was done by checking if the average point of any sensor is below a certain threshold. If any sensor measures an average distance below the threshold, a built-in function communicates the linear and angular velocity to travel, as explained in the next section. The entire algorithm for determining what to communicate to the UGV is a single function. The function loops through to figure out which message to write each time based on which sensors have passed the threshold. If no obstacles are detected for long enough, then the program will exit the for loop and utilize a boolean in the message struct so that the UGV knows that it can return control to its regular path following. The core algorithm is described below with both a flowchart and the code:


Start
c = 0
B
Update Sensors
Wait 1/2 of a second
c > 10?
True
Output to resume navigation
End Interrupt Phase
False
c = 0
A
All Sensors Past threshold?
True
Move Backwards while turning slightly
A
False
Sensor1 and Sesnor2 past threshold or Sensor1 past threshold?
True
Turn Sharply Left
A
False
Sensors 1 and 3 past threshold?
True
Turn Sharply Right
A
False
Sensors 2 and 3 past threshold?
True
Move Backwards while turning slightly
A
False
Sensor2 past threshold?
True
Bear left
False
Sensor3 past threshold?
True
Bear left
False
Move forward
c = c+1
B

```
void interrupt(){
  // Incrementer for loop
  for (int c = 0; c < 10; c++){
    if((Sensor1.avg < Threshold)&&(Sensor2.avg < Threshold)&&(Sensor3.avg < Threshold)){
      Serial.print("all");
      setspeedsteer(-20, 30);
      c = 0;
    }
    else if(( (Sensor1.avg < Threshold) && (Sensor2.avg < Threshold))||(Sensor1.avg <Threshold)){
      Serial.print("Front and Right, or just Front");
      setspeedsteer(15,20);
      c = 0;
    }
    else if((Sensor1.avg < Threshold)&&(Sensor3.avg < Threshold)){
      Serial.print("Front and Left");
      setspeedsteer(15, -20);
      c = 0;
    }
    else if((Sensor2.avg < Threshold)&&(Sensor3.avg < Threshold)){
      Serial.print("Sides")
      setspeedsteer(-15, 20);
      c = 0;
    }
    else if(Sensor2.avg <Threshold){
      Serial.print("Right");
      setspeedsteer(15, 10);
      c = 0;
    }
    else if (Sensor3.avg < Threshold){
      Serial.print("left");
      setspeedsteer(15, -20);
      c = 0;
    }
    else {
      setspeedsteer(20, 0);
    }
      Serial.print("in Interrupt Mode");
      Serial.println();
      Sensor1.Update();
      Sensor2.Update();
      Sensor3.Update();
      delay(125);
  }
}
```

Communication with UGV

```
struct message {
  int speed;
  int steer;
  bool resumeNavigation;
}
ugv_msg;
```

Structure for messages to UGV

What is written to the UGV is a structure that communicates the linear velocity and the angular velocity to move, and whether to return to default navigation or continue to allow the Arduino Microcontroller to control the UGV.

```
  void setspeedsteer(char speed, char steer, bool resumeNavigation)
{
  ugv_msg.speed = speed;
  ugv_msg.steer = steer;
  ugv_msg.resumeNavigation = resumeNavigation;
  Serial.write((byte*) &ugv_msg, sizeof(ugv_msg) );
}
```

Function called to communicate with UGV

Sources

"Pololu Arduino library for VL53L1X time-of-flight distance sensor". *Github.Com*, 2019, https://github.com/pololu/vl53l1x-arduino. Accessed 7 Jan 2019.

"I2C - Learn.Sparkfun.Com". *Learn.Sparkfun.Com*, 2019, https://learn.sparkfun.com/tutorials/i2c/all. Accessed 7 Jan 2019.

Jesudason Nathan, consultation with Client, One of the owners of C-Tonomy, LLC, 1/12/2019.

Word Count: 636