MAE 106 Final Report

Group #812

July 2019

Mingqing Yuan
Qitai Meng
Ruby Gu

1 System Descriptions

1.1 Design Concepts

1.1.1 Mechanical

Our mechanical design goals were to build a robot that was stable, efficient and has a low fraction. To achieve stability, we wanted to make sure the error is small, so we employed the CNC router to cut our wood plates. For wood selection, we chose plywood instead of the one given by school because it is lighter, smoother, and has a better structure for CNC cutting. Besides, we cut two polycarbonate plates for the installation of power transmission. There are two reasons we decided to use polycarbonate over wood; firstly, it is more compact and has smaller error than wood. Since we planned to drill many holes on the plate, we need a robust and inflexible material so it can withstand the force when cutting. Secondly, polycarbonate has higher artistic beauty because it is transparent. Transparency makes our observation easier if the power system had mechanical issues when operating. Moreover, we expected collision with other robots, so we tried to lower the center of mass of our robot by reducing the length of vertical plates.

To reach our efficiency goal, we chose to build a rear- wheel drive robot instead of front-wheel drive to prevent slip of wheels. We designed a rack and pinion with a one-way bearing power system. In order to lower the mechanical fraction, we installed numerous (18) bearings on the wheels and wood plates. Initially, we thought about installing the one-way bearing into the gear; however, we realized this idea was not plausible since we cannot find gear with the correct size for installation of bearing. The only way was designed gear and used 3D printing, but it is still risky. Thus, we decided to put one-way bearing into the wheels for safety. Since we wished to build a rear-wheel drive robot, we exploited two gears when designing our CAD model; one plastic gear (16mm pitch diameter) and one steel (30mm pitch diameter). Nevertheless, we encountered a problem when building the robot; the plastic gear was too big to be placed between the rack and steel gear, so we had to replace it with a 15mm pitch diameter gear. To ensure the quality of our robot, we tried to purchase everything we could find online: Rack, a key shaft, gear, 5mm screw, zinc-plated steel L-shaped bracket, and so on. Meanwhile, we used a 3D printer to print our steering, steering arm, servo case, and rack-piston connector.

Overall, it was a very complicated and detailed robot. The building process was not easy. Even though we got every dimension right from the CAD model, we still encountered a couple of issues because of tolerance. The key shaft which connected two rear wheels stuck too tight with the L-shaped bracket, so we had to calibrate its position; we initially planned to use screws to fix vertical plates; however, we found that

our plates were too thin and screws would rupture them heavily. Therefore, we choose to use L bracket over the screw. One thing that made us most proud was we figured out different means when encountering unexpected troubles and successfully built the robot out. The mechanical structure was even better than we expected because nothing broke or went wrong. After the competition, we all believed that we successfully achieved our design goal of pursuing stability and efficiency.

1.1.2 Software

For software design, we first want an accurate heading reading. We had heard that the magnetometer would be interfered by many factors, since the magnetic field of the earth is quite weak compared to the magnets, solenoids, and other electronics, so we decided to also add a gyroscope to eliminate sudden, and incorrect directions change. We bought a LSM9DS1 chip — a 9DOF IMU with a magnetometer, an accelerometer and a gyro. To code this chip, we modified the existing library for this Adafruit LSM9DS1, so the heading is calculated within the library functions. Calibrations were added into the sensor reading functions, along with a simple low pass filter. The magnetometer read just fine, but the gyro, after integration, always have a random shift, the accelerometer after two integrations only got worse. We then tried comparing heading readings both from the magnetometer and the gyro using a Madgwick filter, the shift is kind of eliminated. Initially, it still has a low frequency noise that puts a plus or minus 10 degrees, but after adjusting the low pass filter of the magnetometer and gyro readings, the noise is much smaller.

Another thing worth mentioning is the acceleration function. While watching videos of previous MAE106 robots on Youtube, we discover that all robot are shooting the piston at a constant speed, so these robots either quickly reach a constant low speed, or slowly and inefficiently reach a high speed. The ones that shoots fast can't use all the pushing period on initial accelerating, in other words, they shoot halfway before retract, since the robot has not reach a higher speed. For example, the first on period of our robot's piston has to be approximately 2 seconds before the piston is completely out, the second one is about 1 second, the following ones is less than 0.5 seconds. We could specify the first two piston on time to be different to achieve a higher acceleration at the beginning. However, we were afraid that after collision our robot would not start properly, thus we need the piston to be 'smart' that the efficiency of the power cycle is maximized. We use a magnet on the gear rack, and write a program to detect when the piston is completely out, then piston can shoot all the way out and retract without stopping, even under various loads. Thanks to this variable piston on duration program, our robot can start quick and stay in front even though it weighs 7.8kg, which puts our robot on the heavier side.

1.2 Photograph and CAD Model

Figure 1. Front view

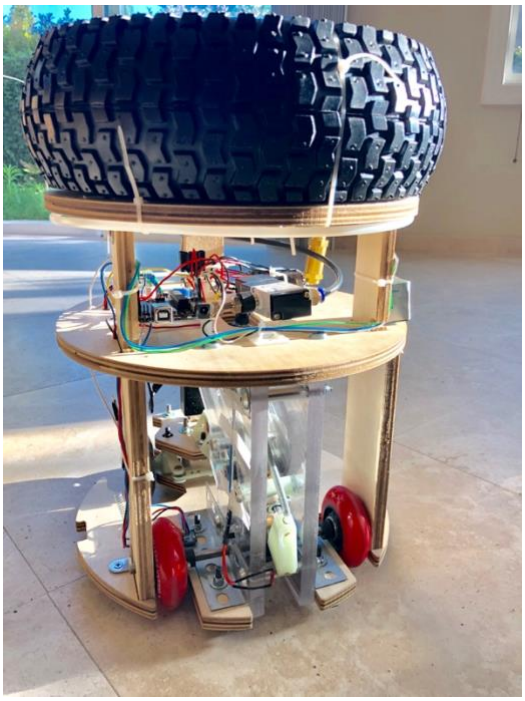

Figure 2. Rear view

Figure 3. Propulsion Mechanism

Figure 4. Steering Mechanism

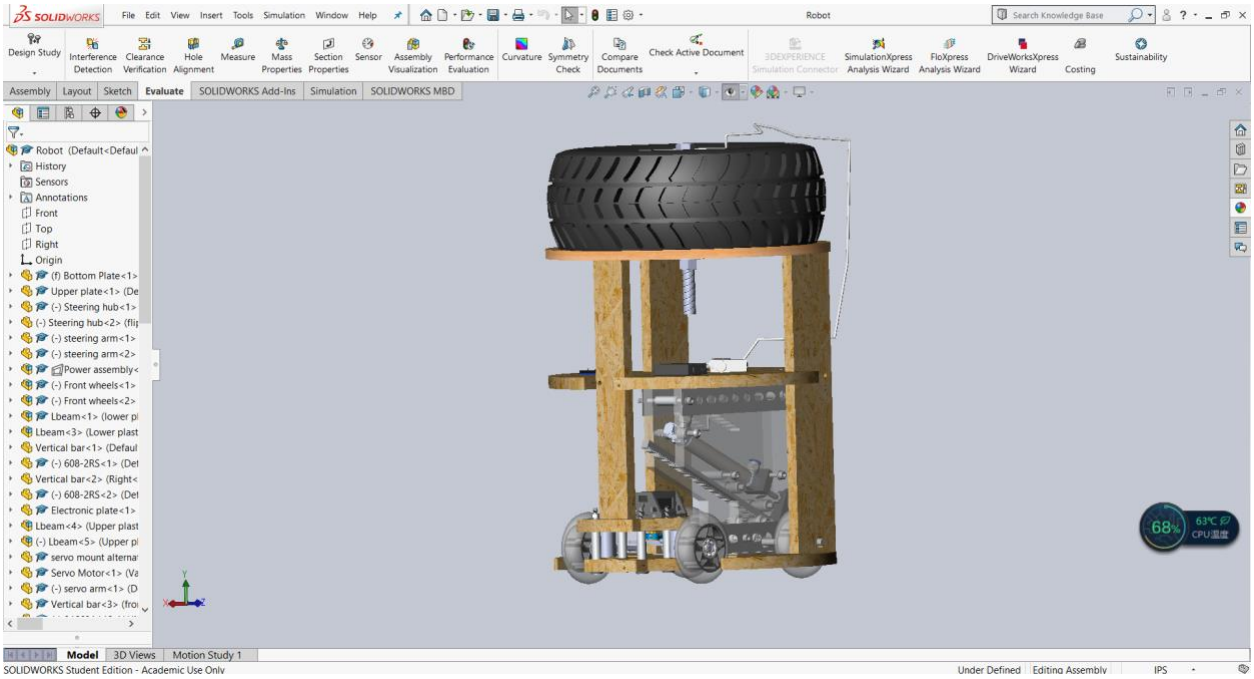

Figure 5. CAD model

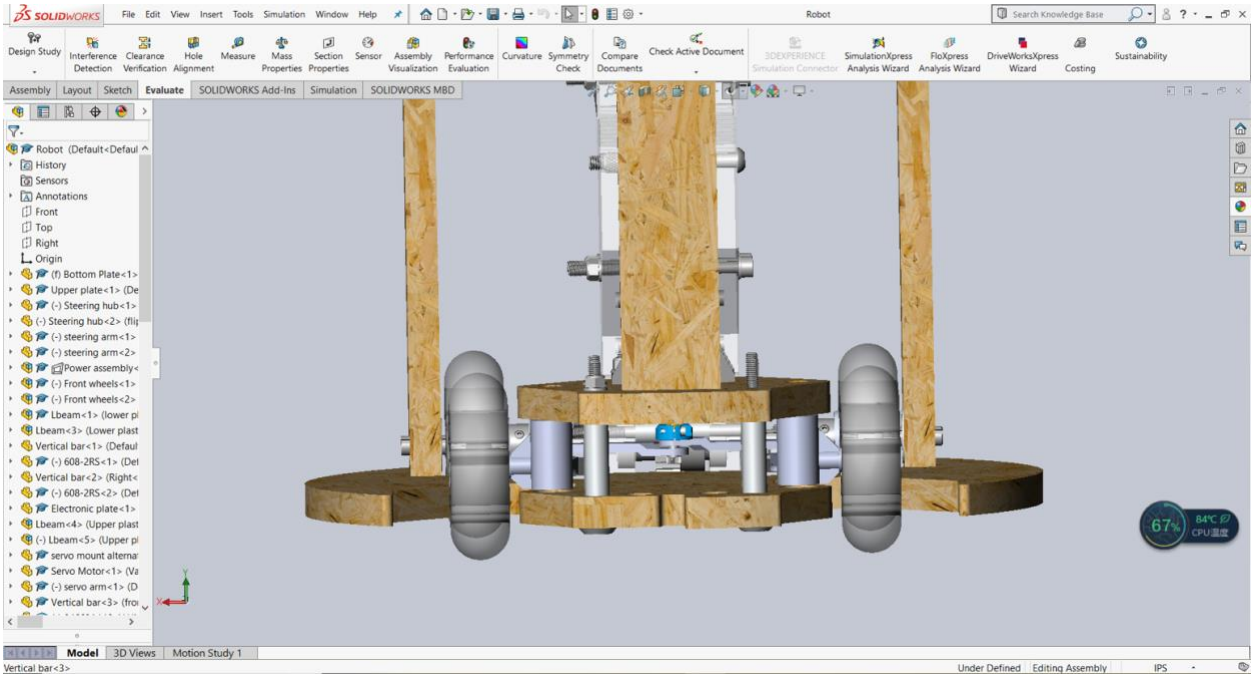

Figure 6. Steering Mechanism CAD

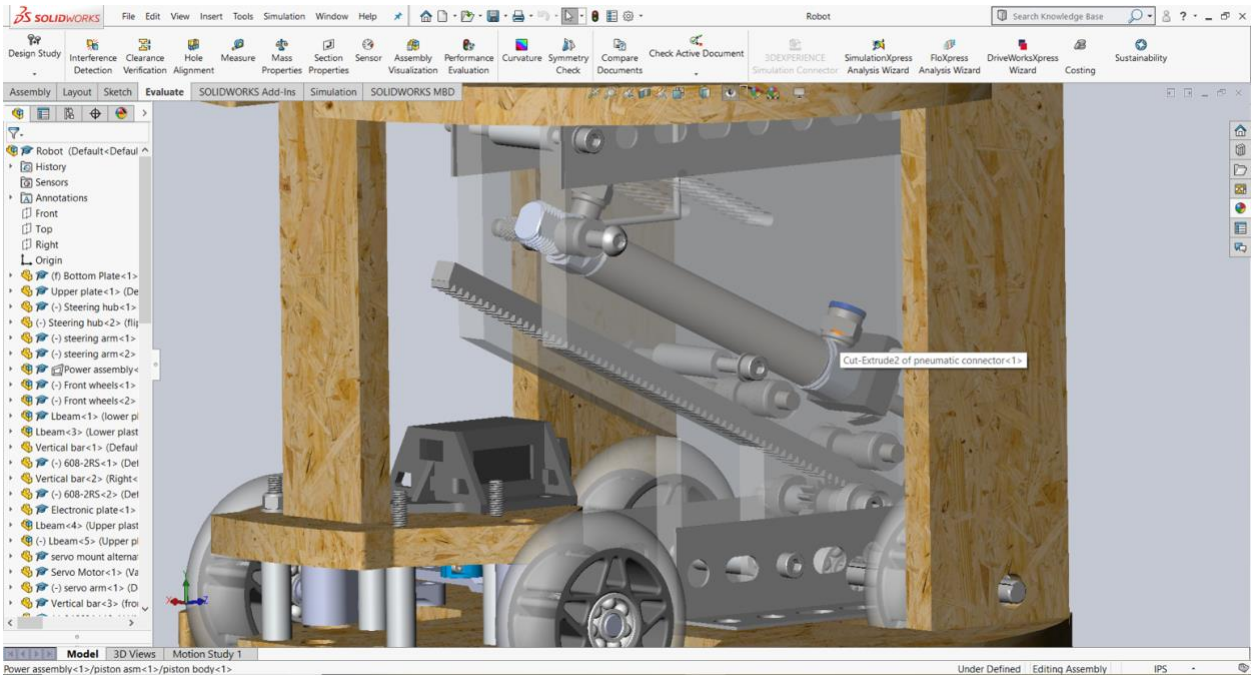

Figure 7. Propulsion Mechanism CAD front

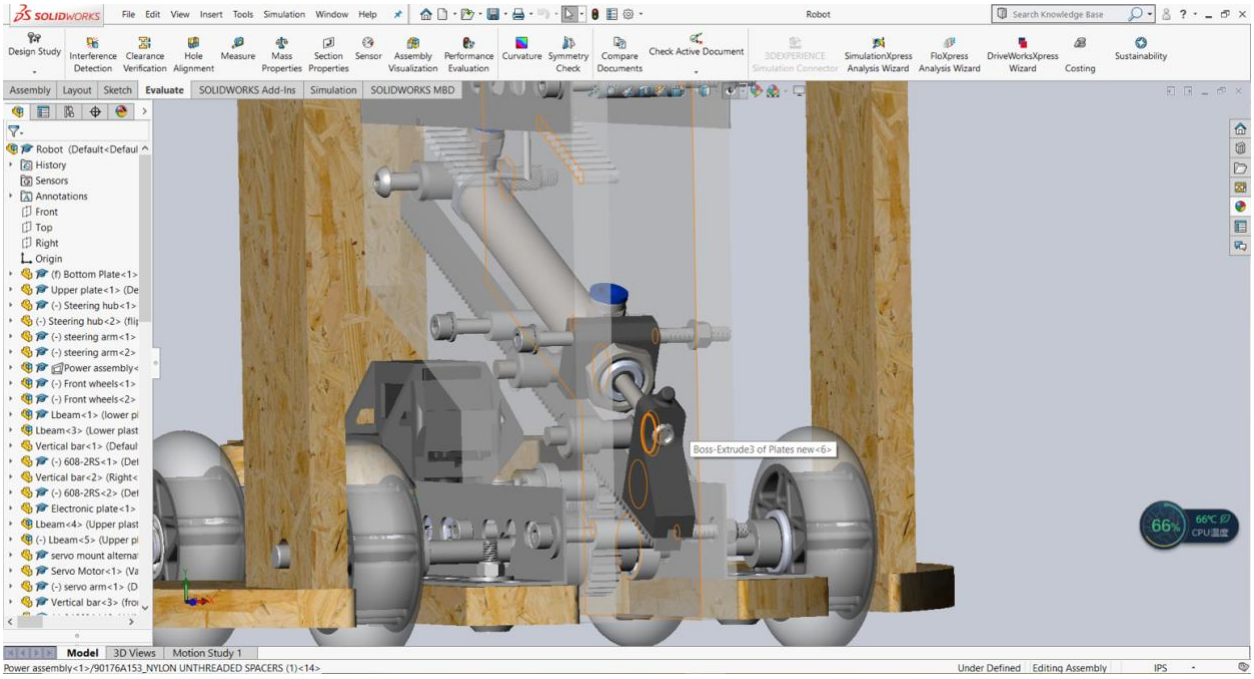

Figure 8. Propulsion Mechanism CAD rear

1.3 Electrical and Pneumatic circuit diagrams

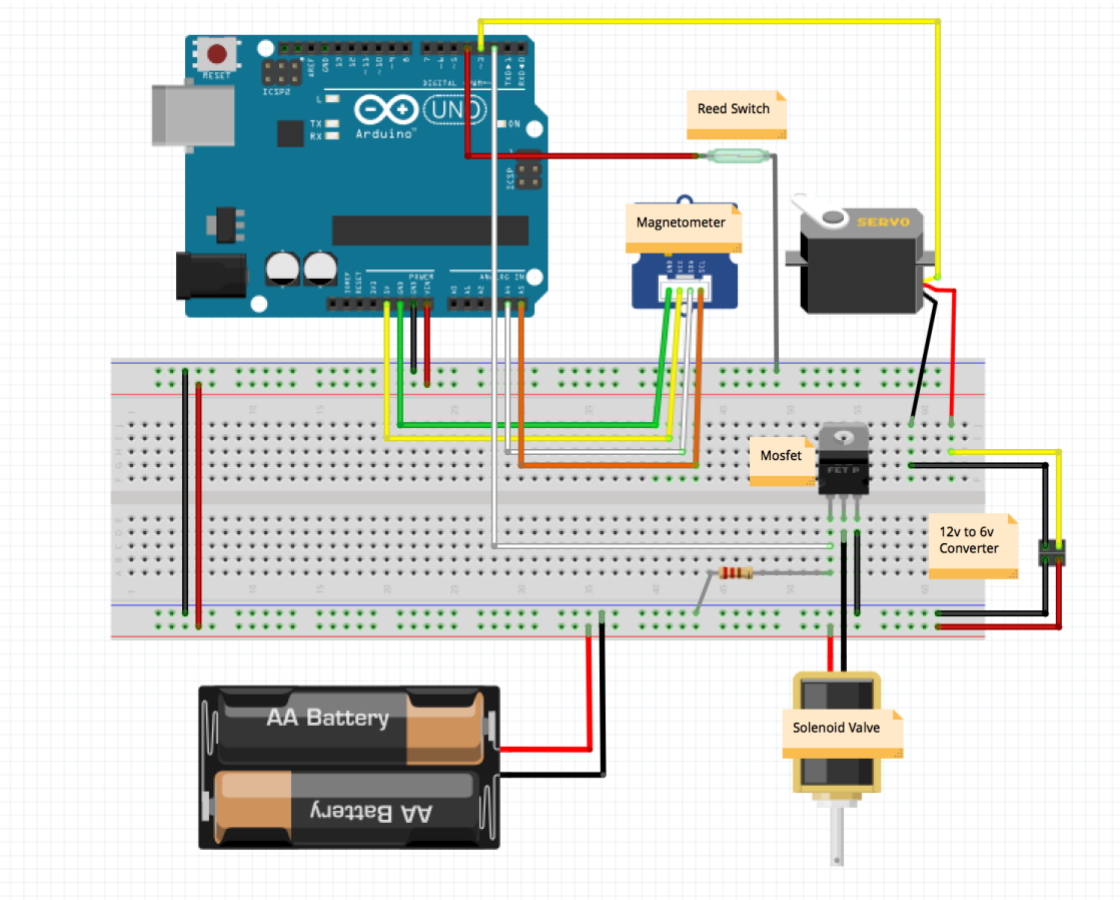

Figure 9. Electrical circuit diagram

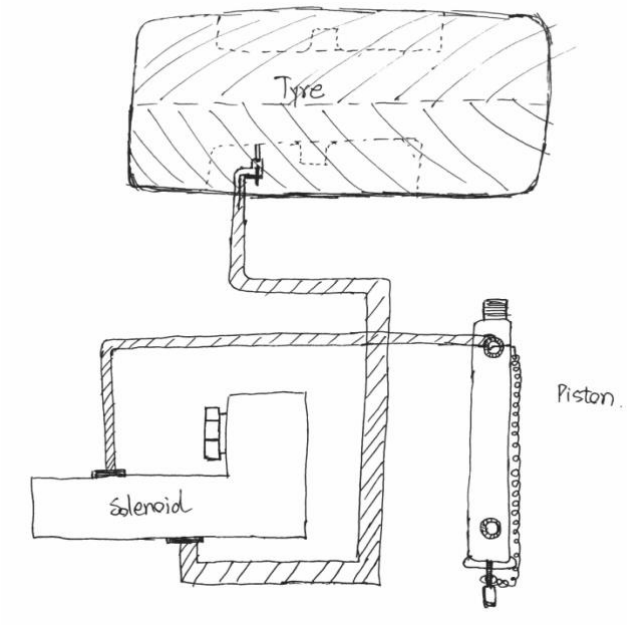

Figure 10. Pneumatic circuit diagram

1.4 Software Code

Three code files were used together in controlling our robot.

- General code
- Library header
- Library cpp

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <MyLSM9DS5.h>

// pin route
#define pinpiston           2
#define pinservo            3
#define pinreedpiston       4
#define pinhall             6
#define pinled              13

// data: store the constants into the program storage rather than take up the
 memory space

// program configuration
#define SERIAL_COM_ON       true        // open or close Serial communication
#define TEST_MODE_ON        false       // decrese the initiation delay
#define FAST_MODE_ON        true        // set true to use mode 3, false to
 force mode 3 to mode 2

// start configuration
#define Dturn1              2300
#define Dturn2              14300
#define DstopPoint          1000

// route configurations
#define DdirOffset          120         // read the starting direction
#define Dfinalmin           200         // stop piston brfore this distance
#define Dslowdowndis        4500        // change mode to 1 before this
 distance
#define Dnextstepdir        5           // if direction is within +-
 Dnextstepdir Sstep goes to 2
#define Dvelmin             200         // minimum speed
#define Dvelnorm            800         // normal speed safe for turning(but
 no fire)
#define Dvelmax             1000        // maximum speed but still efficient
 enough

// PID control parameters
#define Ka                  (2.0F)      // constant for P control of steering

// robot configurations
#define DTstart             16000       // time delay before any action
#define DTrun               58000       // time interval that robot can take
 actions
#define DTpistonfirelen     1200        // duration of piston firing, safe
 exit if reed swtich is not responsive
#define DTpistonretractlen  300         // duration of piston retraction
#define DTreeddelay         50          // reed switch delay time
#define DencoderTic         (42.0F)     // the distance period of every hall
 effect sensor reading, wheel perimeter devided by number of magnets
```

```
#define DrobotCenter        (80.7F)     // half of track width
#define Dsteermax           (22.5F)     // absolute max steering, after which
 is considered dangerous when traveling in normal speed
#define Dturnradius         850         // turn radius, after which robot
 begin turning
#define Dsteersmall         5           // steering is small so speed returns
 to normal and I control is on
#define SERVO_MID           (136.6)     // steering middle value which robot
 would go in a straight line
#define SERVO_MAX           60          // the maximum steering from steering
 middle



// global variables
// time related
unsigned long Tstart =      0;          // record the time before any
 execution of program
unsigned long Tcurrent =    0;          // current time, read from this
 variable instead of millis() to save space
unsigned long Tpiston =     0;          // the time piston started fire and
 started retraction
unsigned long Treed =       0;
unsigned long Tshutdown =   0;
unsigned long TreedSpeed =  0;
unsigned long TcurrentMicro=0;
unsigned long TlastMicro =  0;
unsigned long Tprint =      0;

// commands
bool fire =                 false;      // command for of the piston
bool pistonCom =            false;      // actuation of piston
bool reedcorrected =        false;      // corrected reed switch reading
bool hall =                 false;      // hall effect sensor reading
float steer =               0;          // steering command

// robot datas
float vel =                 0;          // current velocity of robot
float distance =            0;          // current distance traveled of robot
float dir =                 0;          // current heading


// for IMU uses
Adafruit_LSM9DS1            imu;

// functions
void movement               (float Fdestination, float dirLocal, float
 dirBefLocal, int Fmode, float steerInt);

// setup
void setup() {
```

```
    // initiations
    Wire.begin();
    imu.begin();
    Tstart = millis();
    pinMode(pinpiston,OUTPUT);
    pinMode(pinservo,OUTPUT);
    pinMode(pinreedpiston,INPUT_PULLUP);
    pinMode(pinhall,INPUT_PULLUP);
    if (SERIAL_COM_ON)
    {
        Serial.begin(115200);
    }
}
// arduino main loop
void loop() {
    byte steps = 0;
    int mode = 0;                           // speed mode command
    float dirLocal;                         // direciton command
    float destination = 0;
    float dirBef = 0;                       // direction to follow before
     turning point
    // loop act as delay
    while(millis()<(Tstart+DTstart))
    {
        imu.read();
    }
    // second initiation
    imu.dirOffset = imu.direction + 90;
    float steerInt = 0;
    Tstart = millis();
    Tcurrent = millis();
    while(Tcurrent<(Tstart+DTrun))
    {
        // main action loop
        // get values for each cycle
        Tcurrent = millis();
        imu.read();
        dir = imu.direction;
        //main steps
        switch(steps)
        {
            case 0:
            {
                steps ++;
                break;                       // empty utility case
            }
            case 1:
            {
                //approach center line
                destination = Dturn1;
                dirLocal = 5;
```

```
        dirBef = -90;
        mode = 2;
        if (abs(dir - 5) < Dnextstepdir) steps ++;
        break;
    }
    case 2:
    {
        // go the lone way
        destination = Dturn2 + Dturn1 - (Dturnradius * (4 - M_PI) / 2);
        dirLocal = 0;
        dirBef = 0;

        if ((destination - distance) > Dslowdowndis) mode = 400;
        else
        {
            mode = 1;
            steps ++;
            Tshutdown = Tcurrent;
        }
        break;
    }
    case 3:
    {
        // approach the final horizontal line
        destination = Dturn2 + Dturn1 - (Dturnradius * (4 - M_PI) / 2);
        dirLocal = -90;
        dirBef = 0;
        steerInt = 5;
        if (abs(90+dir) < Dnextstepdir) steps ++;
        break;
    }
    case 4:
    {
        // stop at the final block
        destination = DstopPoint + Dturn2 + Dturn1 - (Dturnradius * (4
         - M_PI));
        dirLocal = -90;
        dirBef = -90;
        steerInt = 5;
        mode = 1;
        if (destination - distance < Dfinalmin) mode = 0;
        break;
    }
    default:{

        break;
    }
}
movement(destination,dirLocal,dirBef,mode,steerInt);    // giving the
 command
```

```
        if (steps > 4) break;                                   // indicating
         the program is finished and jump out of this loop
        if (SERIAL_COM_ON)
        {
            Serial.print(steps);
            Serial.print("    ");
            Serial.print(dir);
            Serial.print("    ");
            Serial.print(distance);
            Serial.print("    ");
            Serial.println(steer);
        }
    }
    Serial.print("complete");
    //make sure the robot is stoped
    digitalWrite(pinpiston,false);
    while(true){}                                               // ensure
     there is no further action
}

// functions

void movement(float Fdestination, float dirLocal, float dirBefLocal, int Fmode
 = 2, float steerInt = 0){
    float distanceLocal = Fdestination - distance;
    float steerMaxLocal = Dsteermax;
    if (steerInt != 0) steerMaxLocal = steerInt;                // special
     steering maximum command
    // calculations
    speedCal();                                                 // calculate
     the current speed and distance moved
    if (distanceLocal > Dturnradius) dirLocal = dirBefLocal;    // maintaining
     one direction before turning
    steer = (dirLocal - dir) * Ka;                              // P control
     of steer
    // limit filter for steer command to avoid tight turning
    if (steer > steerMaxLocal) steer = steerMaxLocal;
    else if (steer < -steerMaxLocal) steer = -steerMaxLocal;
    // speed limit when turning
    if (abs(steer) > Dsteersmall){      // decrese speed when turning
        if (Fmode > 1) Fmode = 1;
    }
    //if (abs(steer) < 3) steer = 0;                              // avoid
     small errors to accumulate
    speedMode(Fmode);                                           // speed
     command
    steering();                                                 // steering
     command
    return;

}
```

```
void steering(){
    // filter out the command over the mechanical limit
    if (steer < -SERVO_MAX) steer = -SERVO_MAX;
    else if (steer > SERVO_MAX) steer = SERVO_MAX;
    // shift the servo command
    int steerOut = steer + SERVO_MID;
    analogWrite(pinservo,steerOut);
}


void speedMode(int Fmode = 0)                                   // control
 actuation and the speed of the robot
{
    static bool Ifire =                 true;
    static bool Iretract =              true;
    static bool Ireed =                 false;
    static bool firstfire =             true;
    static unsigned long TfireDelay =   0;
    // mode selection = 0: slide; 1: turning; 2: normal; 3: fast;
    if (!FAST_MODE_ON) if (Fmode > 2) Fmode = 2;                // fastmode
     switch
    // initiation for custom firing frequency
    if (Fmode>3)
    {
        if (firstfire)
        {
            TfireDelay = Tcurrent;
            fire = true;
            firstfire = false;
        }
    }
    else firstfire = true;
    // mode selector
    switch(Fmode){
        case 0:                                                 // stop, no
         action
            fire = false;
            break;
        case 1:                                                 // slow
            if (vel < Dvelmin) fire = true;
            break;
        case 2:                                                 // normal
            if (vel < Dvelnorm) fire = true;
            break;
        case 3:                                                 // fast
            if (vel < Dvelmax) fire = true;
            break;
        default:                                                // custom
         firing frequency
        {
```

```
        if (fire) TfireDelay = Tcurrent;
        if (Tcurrent - TfireDelay > Fmode) fire = true;
    }
}
// fire command
if (fire){
    if (Ifire)
    {
        // initiation for shooting our
        pistonCom = true;
        Tpiston = Tcurrent;
        Ifire = false;
    }
    if (Iretract)
    {
        if (reedcorrected||(Tcurrent - Tpiston > DTpistonfirelen))
        {
            // retraction initiation
            pistonCom = false;
            Tpiston = Tcurrent;
            Iretract = false;
        }
    }
    else if (Tcurrent - Tpiston > DTpistonretractlen)
    {
        // clear locks for consecutive firing
        Ifire = true;
        Iretract = true;
        fire = false;
    }
}
else
{
    // rest state, clear locks
    Ifire = true;
    Iretract = true;
    pistonCom = false;
}
digitalWrite(pinpiston,pistonCom);
// reedswitch correction, give a adjustable delay to the reedswitch, to
 conterract the program delay and eliminate the need to adjust
 reedswitch's position
if (Tcurrent - Treed > DTreeddelay){
    if (reedcorrected)
        reedcorrected = false;
    if (Ireed){
        reedcorrected = true;
        Ireed = false;
    }
    else if (!digitalRead(pinreedpiston))
        Treed = Tcurrent;
```

```
    }
    else
        Ireed = true;
}


void speedCal()
{
    static float speedLocal = 0;
    static bool Ihall = false;
    TcurrentMicro = micros();
    // count the events of magnet near
    hall = digitalRead(pinhall);
    // lowest speed or compensation for sudden stops
    if (TcurrentMicro - TlastMicro > DencoderTic / Dvelmin * 1000000) vel = 0;
    // hall effect sensor counting
    if (hall)
    {
        if (Ihall)
        {
            float timeFrac = ((float)TcurrentMicro - (float)TlastMicro) /
             1000000.0;    // time fraction
            speedLocal = DencoderTic / timeFrac;
                      // speed calculation
            distance += DencoderTic;
                      // velocity calculation
            // with rotation compenstion
            vel = speedLocal - imu.gyro.z * DrobotCenter;
            // ending
            TlastMicro = TcurrentMicro;
            Ihall = false;
        }
    }
    else Ihall = true;
}
```

```
/**************************************************************************
 This is a library for the LSM9DS1 Accelerometer and magnentometer/compass

 Designed specifically to work with the Adafruit LSM9DS1 Breakouts

 These sensors use I2C to communicate, 2 pins are required to interface.

 Adafruit invests time and resources providing this open source code,
 please support Adafruit andopen-source hardware by purchasing products
 from Adafruit!

 Written by Kevin Townsend for Adafruit Industries.
 BSD license, all text above must be included in any redistribution
 **************************************************************************/
#ifndef __LSM9DS1_H__
#define __LSM9DS1_H__

#if (ARDUINO >= 100)
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include "Wire.h"
#include <SPI.h>
#include <Adafruit_Sensor.h>

#define LSM9DS1_ADDRESS_ACCELGYRO           (0x6B)
#define LSM9DS1_ADDRESS_MAG                 (0x1E)
#define LSM9DS1_XG_ID                       (0b01101000)
#define LSM9DS1_MAG_ID                      (0b00111101)

// Linear Acceleration: mg per LSB
#define LSM9DS1_ACCEL_MG_LSB_2G             (0.061F)
#define LSM9DS1_ACCEL_MG_LSB_4G             (0.122F)
#define LSM9DS1_ACCEL_MG_LSB_8G             (0.244F)
#define LSM9DS1_ACCEL_MG_LSB_16G            (0.732F)

// Magnetic Field Strength: gauss range
#define LSM9DS1_MAG_MGAUSS_4GAUSS           (0.14F)
#define LSM9DS1_MAG_MGAUSS_8GAUSS           (0.29F)
#define LSM9DS1_MAG_MGAUSS_12GAUSS          (0.43F)
#define LSM9DS1_MAG_MGAUSS_16GAUSS          (0.58F)

// Angular Rate: dps per LSB
#define LSM9DS1_GYRO_DPS_DIGIT_245DPS       (0.00875F)
#define LSM9DS1_GYRO_DPS_DIGIT_500DPS       (0.01750F)
#define LSM9DS1_GYRO_DPS_DIGIT_2000DPS      (0.07000F)

// Temperature: LSB per degree celsius
#define LSM9DS1_TEMP_LSB_DEGREE_CELSIUS     (8)  // 1°C = 8, 25° = 200, etc.
```

```
#define MAGTYPE                           (true)
#define XGTYPE                            (false)

/* Forward reference required for function pointers below. */
class Adafruit_LSM9DS1;

/* Pointer to member functions for read, get event, and get sensor.  These are
 used */
/* by the Adafruit_LSM9DS1::Sensor class to read and retrieve individual
 sensors. */
typedef void (Adafruit_LSM9DS1::*lsm9ds1_read_func)(void);
typedef void (Adafruit_LSM9DS1::*lsm9ds1_get_event_func)(sensors_event_t*,
 uint32_t);
typedef void (Adafruit_LSM9DS1::*lsm9ds1_get_sensor_func)(sensor_t*);

typedef struct vector_s
{
    float x=0;
    float y=0;
    float z=0;
}vector;
static vector emptyVector;


class Adafruit_LSM9DS1
{
public:
    Adafruit_LSM9DS1 ( int32_t sensorID = 0 );
    Adafruit_LSM9DS1 ( TwoWire* wireBus, int32_t sensorID = 0 );
    Adafruit_LSM9DS1 ( int8_t xmcs, int8_t gcs, int32_t sensorID = 0 );
    Adafruit_LSM9DS1 ( int8_t clk, int8_t miso, int8_t mosi, int8_t xmcs,
     int8_t gcs, int32_t sensorID = 0 );

    void initI2C( TwoWire* wireBus, int32_t sensorID );


    typedef enum
    {
        LSM9DS1_REGISTER_WHO_AM_I_XG         = 0x0F,
        LSM9DS1_REGISTER_CTRL_REG1_G         = 0x10,
        LSM9DS1_REGISTER_CTRL_REG2_G         = 0x11,
        LSM9DS1_REGISTER_CTRL_REG3_G         = 0x12,
        LSM9DS1_REGISTER_TEMP_OUT_L          = 0x15,
        LSM9DS1_REGISTER_TEMP_OUT_H          = 0x16,
        LSM9DS1_REGISTER_STATUS_REG          = 0x17,
        LSM9DS1_REGISTER_OUT_X_L_G           = 0x18,
        LSM9DS1_REGISTER_OUT_X_H_G           = 0x19,
        LSM9DS1_REGISTER_OUT_Y_L_G           = 0x1A,
        LSM9DS1_REGISTER_OUT_Y_H_G           = 0x1B,
        LSM9DS1_REGISTER_OUT_Z_L_G           = 0x1C,
        LSM9DS1_REGISTER_OUT_Z_H_G           = 0x1D,
```

```
    LSM9DS1_REGISTER_CTRL_REG4          = 0x1E,
    LSM9DS1_REGISTER_CTRL_REG5_XL       = 0x1F,
    LSM9DS1_REGISTER_CTRL_REG6_XL       = 0x20,
    LSM9DS1_REGISTER_CTRL_REG7_XL       = 0x21,
    LSM9DS1_REGISTER_CTRL_REG8          = 0x22,
    LSM9DS1_REGISTER_CTRL_REG9          = 0x23,
    LSM9DS1_REGISTER_CTRL_REG10         = 0x24,

    LSM9DS1_REGISTER_OUT_X_L_XL         = 0x28,
    LSM9DS1_REGISTER_OUT_X_H_XL         = 0x29,
    LSM9DS1_REGISTER_OUT_Y_L_XL         = 0x2A,
    LSM9DS1_REGISTER_OUT_Y_H_XL         = 0x2B,
    LSM9DS1_REGISTER_OUT_Z_L_XL         = 0x2C,
    LSM9DS1_REGISTER_OUT_Z_H_XL         = 0x2D,

} lsm9ds1AccGyroRegisters_t;

typedef enum
{

    LSM9DS1_REGISTER_WHO_AM_I_M           = 0x0F,
    LSM9DS1_REGISTER_CTRL_REG1_M          = 0x20,
    LSM9DS1_REGISTER_CTRL_REG2_M          = 0x21,
    LSM9DS1_REGISTER_CTRL_REG3_M          = 0x22,
    LSM9DS1_REGISTER_CTRL_REG4_M          = 0x23,
    LSM9DS1_REGISTER_CTRL_REG5_M          = 0x24,
    LSM9DS1_REGISTER_STATUS_REG_M         = 0x27,
    LSM9DS1_REGISTER_OUT_X_L_M            = 0x28,
    LSM9DS1_REGISTER_OUT_X_H_M            = 0x29,
    LSM9DS1_REGISTER_OUT_Y_L_M            = 0x2A,
    LSM9DS1_REGISTER_OUT_Y_H_M            = 0x2B,
    LSM9DS1_REGISTER_OUT_Z_L_M            = 0x2C,
    LSM9DS1_REGISTER_OUT_Z_H_M            = 0x2D,
    LSM9DS1_REGISTER_CFG_M                = 0x30,
    LSM9DS1_REGISTER_INT_SRC_M            = 0x31,
} lsm9ds1MagRegisters_t;

typedef enum
{
    LSM9DS1_ACCELRANGE_2G                 = (0b00 << 3),
    LSM9DS1_ACCELRANGE_16G                = (0b01 << 3),
    LSM9DS1_ACCELRANGE_4G                 = (0b10 << 3),
    LSM9DS1_ACCELRANGE_8G                 = (0b11 << 3),
} lsm9ds1AccelRange_t;

typedef enum
{
    LSM9DS1_ACCELDATARATE_POWERDOWN       = (0b0000 << 4),
    LSM9DS1_ACCELDATARATE_3_125HZ         = (0b0001 << 4),
    LSM9DS1_ACCELDATARATE_6_25HZ          = (0b0010 << 4),
    LSM9DS1_ACCELDATARATE_12_5HZ          = (0b0011 << 4),
```

```
    LSM9DS1_ACCELDATARATE_25HZ              = (0b0100 << 4),
    LSM9DS1_ACCELDATARATE_50HZ              = (0b0101 << 4),
    LSM9DS1_ACCELDATARATE_100HZ             = (0b0110 << 4),
    LSM9DS1_ACCELDATARATE_200HZ             = (0b0111 << 4),
    LSM9DS1_ACCELDATARATE_400HZ             = (0b1000 << 4),
    LSM9DS1_ACCELDATARATE_800HZ             = (0b1001 << 4),
    LSM9DS1_ACCELDATARATE_1600HZ            = (0b1010 << 4)
} lm9ds1AccelDataRate_t;

typedef enum
{
    LSM9DS1_MAGGAIN_4GAUSS                = (0b00 << 5),  // +/- 4 gauss
    LSM9DS1_MAGGAIN_8GAUSS                = (0b01 << 5),  // +/- 8 gauss
    LSM9DS1_MAGGAIN_12GAUSS               = (0b10 << 5),  // +/- 12 gauss
    LSM9DS1_MAGGAIN_16GAUSS               = (0b11 << 5)   // +/- 16 gauss
} lsm9ds1MagGain_t;

typedef enum
{
    LSM9DS1_MAGDATARATE_3_125HZ           = (0b000 << 2),
    LSM9DS1_MAGDATARATE_6_25HZ            = (0b001 << 2),
    LSM9DS1_MAGDATARATE_12_5HZ            = (0b010 << 2),
    LSM9DS1_MAGDATARATE_25HZ              = (0b011 << 2),
    LSM9DS1_MAGDATARATE_50HZ              = (0b100 << 2),
    LSM9DS1_MAGDATARATE_100HZ             = (0b101 << 2)
} lsm9ds1MagDataRate_t;

typedef enum
{
    LSM9DS1_GYROSCALE_245DPS              = (0b00 << 3),  // +/- 245
     degrees per second rotation
    LSM9DS1_GYROSCALE_500DPS              = (0b01 << 3),  // +/- 500
     degrees per second rotation
    LSM9DS1_GYROSCALE_2000DPS             = (0b11 << 3)   // +/- 2000
     degrees per second rotation
} lsm9ds1GyroScale_t;

vector  accel;
vector  mag;
vector  gyro;
float   temperature;

float   roll;
float   pitch;
float   heading;
float   yaw;
float   direction;
float   q[4] = {1.0f, 0.0f, 0.0f, 0.0f};
bool    readMagHold =   false;

// setup parameters
```

```
    bool    filterEnable =  true;
    float   dirOffset =     0;

#define GyroMeasError PI * (30.0f / 180.0f)
#define beta sqrt(3.0f / 4.0f) * GyroMeasError

#define     LPF_ACCEL                           (0.1F)
#define     LPF_GYRO                            (0.15F)
#define     LPF_MAG                             (0.1F)

    vector  plus            (vector front,vector rear);
    vector  minus           (vector front,vector rear);
    vector  devide          (vector up,float down);
    vector  multi           (vector front,float rear);
    void    lowPassFilter   (vector& target, vector filterInput, float
     filterAlpha);

    bool    begin           ( void );
    void    read            ( void );
    void    readAccel       ( void );
    void    readGyro        ( void );
    void    readMag         ( void );
    void    readTemp        ( void );
    void    setupAccel      ( lsm9ds1AccelRange_t range );
    void    setupMag        ( lsm9ds1MagGain_t gain );
    void    setupGyro       ( lsm9ds1GyroScale_t scale );
    void    MadgwickQuaternionUpdate( void );
    void    write8          ( boolean type, byte reg, byte value );
    byte    read8           ( boolean type, byte reg);
    byte    readBuffer      ( boolean type, byte reg, byte len, uint8_t
     *buffer);


private:
    boolean _i2c;
    TwoWire* _wire;
    int8_t  _csm, _csxg, _mosi, _miso, _clk;
    float   _accel_mg_lsb;
    float   _mag_mgauss_lsb;
    float   _gyro_dps_digit;
    int32_t _lsm9dso_sensorid_accel;
    int32_t _lsm9dso_sensorid_mag;
    int32_t _lsm9dso_sensorid_gyro;
    int32_t _lsm9dso_sensorid_temp;
    float timeSplit = 0;
    unsigned long sensorTimeLast,sensorTime;
};
#endif
```

```
/**************************************************************************
 This is a library for the LSM9DS1 Accelerometer and magnentometer/compass

 Designed specifically to work with the Adafruit LSM9DS1 Breakouts

 These sensors use I2C to communicate, 2 pins are required to interface.

 Adafruit invests time and resources providing this open source code,
 please support Adafruit andopen-source hardware by purchasing products
 from Adafruit!

 Written by Kevin Townsend for Adafruit Industries.
 BSD license, all text above must be included in any redisplacetribution
 **************************************************************************/
#include <MyLSM9DS5.h>

/**************************************************************************
 CONSTRUCTOR
 **************************************************************************/

void Adafruit_LSM9DS1::initI2C( TwoWire* wireBus, int32_t sensorID ) {
    _i2c = true;
    _wire = wireBus;
    _lsm9dso_sensorid_accel = sensorID + 1;
    _lsm9dso_sensorid_mag = sensorID + 2;
    _lsm9dso_sensorid_gyro = sensorID + 3;
    _lsm9dso_sensorid_temp = sensorID + 4;
}
// default
Adafruit_LSM9DS1::Adafruit_LSM9DS1( int32_t sensorID )
{
    initI2C(&Wire, sensorID);
}

Adafruit_LSM9DS1::Adafruit_LSM9DS1( TwoWire* wireBus, int32_t sensorID )
{
    initI2C(wireBus, sensorID);
}

vector Adafruit_LSM9DS1::plus(vector front,vector rear)
{
    vector result;
    result.x = front.x + rear.x;
    result.y = front.y + rear.y;
    result.z = front.z + rear.z;
    return result;
}
vector Adafruit_LSM9DS1::minus(vector front,vector rear)
{
    vector result;
    result.x = front.x - rear.x;
```

```
    result.y = front.y - rear.y;
    result.z = front.z - rear.z;
    return result;
}
vector Adafruit_LSM9DS1::devide(vector up,float down)
{
    vector result;
    result.x = up.x/down;
    result.y = up.y/down;
    result.z = up.z/down;
    return result;
}
vector Adafruit_LSM9DS1::multi(vector front,float rear)
{
    vector result;
    result.x = front.x * rear;
    result.y = front.y * rear;
    result.z = front.z * rear;
    return result;
}
void Adafruit_LSM9DS1::lowPassFilter(vector& target, vector filterInput, float
 filterAlpha)
{
    if (filterEnable)
    {
        target.x = filterInput.x * filterAlpha + (target.x * (1.0 -
         filterAlpha));
        target.y = filterInput.y * filterAlpha + (target.y * (1.0 -
         filterAlpha));
        target.z = filterInput.z * filterAlpha + (target.z * (1.0 -
         filterAlpha));
    }
    else target = filterInput;
    return;
}

bool Adafruit_LSM9DS1::begin(){
    _wire->begin();
    // soft reset & reboot accel/gyro
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG8, 0x05);
    // soft reset & reboot magnetometer
    write8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG2_M, 0x0C);
    delay(10);
    uint8_t id = read8(XGTYPE, LSM9DS1_REGISTER_WHO_AM_I_XG);
    //Serial.print ("XG whoami: 0x"); Serial.println(id, HEX);
    if (id != LSM9DS1_XG_ID) return false;
    id = read8(MAGTYPE, LSM9DS1_REGISTER_WHO_AM_I_M);
    //Serial.print ("MAG whoami: 0x"); Serial.println(id, HEX);
    if (id != LSM9DS1_MAG_ID) return false;
    // enable gyro continuous
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG1_G, 0xC0); // on XYZ
```

```
    // Enable the accelerometer continous
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG5_XL, 0x38); // enable X Y and Z
     axis
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG6_XL, 0xC0); // 1 KHz out data
     rate, BW set by ODR, 408Hz anti-aliasing
    //write8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG1_M, 0xFC); // high perf XY, 80
     Hz ODR
    write8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG3_M, 0x00); // continuous mode
    //write8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG4_M, 0x0C); // high perf Z mode
    // Set default ranges for the various sensors
    setupAccel(LSM9DS1_ACCELRANGE_2G);
    setupMag(LSM9DS1_MAGGAIN_4GAUSS);
    setupGyro(LSM9DS1_GYROSCALE_245DPS);
    return true;
}

/**************************************************************************
 PUBLIC FUNCTIONS
 **************************************************************************/
void Adafruit_LSM9DS1::read()
{
    readAccel();
    readGyro();
    readMag();
    if ( sensorTimeLast == 0 )
    {
        // sub initialization
        sensorTimeLast = micros();
        return;
    }
    sensorTime = micros();
    timeSplit = ((sensorTime - sensorTimeLast)/1000000.0f);
    sensorTimeLast = sensorTime;
    MadgwickQuaternionUpdate();
    yaw   = atan2(2.0f * (q[1] * q[2] + q[0] * q[3]), q[0] * q[0] + q[1] *
     q[1] - q[2] * q[2] - q[3] * q[3]);
    pitch = -asin(2.0f * (q[1] * q[3] - q[0] * q[2]));
    roll  = atan2(2.0f * (q[0] * q[1] + q[2] * q[3]), q[0] * q[0] - q[1] *
     q[1] - q[2] * q[2] + q[3] * q[3]);
    pitch *= 180.0f / PI;
    yaw   *= 180.0f / PI;
    roll  *= 180.0f / PI;
    direction = yaw - dirOffset;
    if (direction > 180) direction -= 360;
    else if (direction < -180) direction += 360;
    return;
}

// read raw data with calibratoin and filter
void Adafruit_LSM9DS1::readAccel() {
    // Read the accelerometer
```

```
    byte buffer[6];
    readBuffer(XGTYPE,
               0x80 | LSM9DS1_REGISTER_OUT_X_L_XL,
               6, buffer);

    uint8_t xlo = buffer[0];
    int16_t xhi = buffer[1];
    uint8_t ylo = buffer[2];
    int16_t yhi = buffer[3];
    uint8_t zlo = buffer[4];
    int16_t zhi = buffer[5];

    // Shift values to create properly formed integer (low byte first)
    xhi <<= 8; xhi |= xlo;
    yhi <<= 8; yhi |= ylo;
    zhi <<= 8; zhi |= zlo;
    vector temperal,accelLocal,accelCal;
    temperal.x = xhi; temperal.y = yhi; temperal.z = zhi;
    accelLocal = multi(temperal,  _accel_mg_lsb);
    accelLocal = devide(accelLocal,1000);
    accelLocal = multi(accelLocal, SENSORS_GRAVITY_STANDARD);
    // Accelerometer calibration
    accelLocal.x += 0.126756; //X-axis combined bias (Non calibrated data -
     bias)
    accelLocal.y += 0.176095; //Y-axis combined bias (Default: substracting
     bias)
    accelLocal.z += 0.176943; //Z-axis combined bias
    // Correction for combined scale factors
    accelCal.x =  1.001643 * accelLocal.x - 0.005084 * accelLocal.y + 0.000969
     * accelLocal.z;
    accelCal.y = -0.005084 * accelLocal.x + 1.004374 * accelLocal.y - 0.003597
     * accelLocal.z;
    accelCal.z =  0.000969 * accelLocal.x - 0.003597 * accelLocal.y + 0.994253
     * accelLocal.z;
    // low pass filter
    lowPassFilter(accel,accelCal,LPF_ACCEL);
    return;

}

void Adafruit_LSM9DS1::readGyro() {
    // Read gyro
    byte buffer[6];
    readBuffer(XGTYPE, 0x80 | LSM9DS1_REGISTER_OUT_X_L_G, 6, buffer);
    uint8_t xlo = buffer[0];
    int16_t xhi = buffer[1];
    uint8_t ylo = buffer[2];
    int16_t yhi = buffer[3];
    uint8_t zlo = buffer[4];
    int16_t zhi = buffer[5];
    // Shift values to create properly formed integer (low byte first)
```

```
    xhi <<= 8; xhi |= xlo;
    yhi <<= 8; yhi |= ylo;
    zhi <<= 8; zhi |= zlo;
    // displacet processing
    vector temperal,gyroLocal;
    temperal.x = xhi; temperal.y = yhi; temperal.z = zhi;
    gyroLocal = multi(temperal,  _gyro_dps_digit);
    // Calibration
    gyroLocal.x += 0.35;
    gyroLocal.y -= 1.03;
    gyroLocal.z -= 0.30;
    // Low pass filter
    lowPassFilter(gyro,gyroLocal,LPF_GYRO);
    return;
}

void Adafruit_LSM9DS1::readMag() {
    // Read the magnetometer
    byte buffer[6];
    readBuffer(MAGTYPE, 0x80 | LSM9DS1_REGISTER_OUT_X_L_M, 6, buffer);
    uint8_t xlo = buffer[0];
    int16_t xhi = buffer[1];
    uint8_t ylo = buffer[2];
    int16_t yhi = buffer[3];
    uint8_t zlo = buffer[4];
    int16_t zhi = buffer[5];
    // Shift values to create properly formed integer (low byte first)
    xhi <<= 8; xhi |= xlo;
    yhi <<= 8; yhi |= ylo;
    zhi <<= 8; zhi |= zlo;
    // displacet processing
    vector temperal,magLocal,magCal;
    temperal.x = xhi; temperal.y = yhi; temperal.z = zhi;
    magLocal = multi(temperal,  _mag_mgauss_lsb);
    magLocal = devide(magLocal,1000);
    // Magnetometer calibration
    magLocal.x -= 0.061188; //X-axis combined bias (Non calibrated data - bias)
    magLocal.y += 0.013368; //Y-axis combined bias (Default: substracting bias)
    magLocal.z += 1.041893; //Z-axis combined bias
    // Correction for combined scale factors (Default: displaceitive factors)
    if (!readMagHold)
    {
        magCal.x =  1.030888 * magLocal.x + 0.015885 * magLocal.y - 0.017615 *
         magLocal.z;
        magCal.y =  0.015885 * magLocal.x + 1.194124 * magLocal.y - 0.012146 *
         magLocal.z;
        magCal.z = -0.017615 * magLocal.x - 0.012146 * magLocal.y + 1.115388 *
         magLocal.z;
        lowPassFilter(mag,magCal,LPF_MAG);
    }
    return;
```

```
}

void Adafruit_LSM9DS1::readTemp() {
    // Read temp sensor
    byte buffer[2];
    readBuffer(XGTYPE, 0x80 | LSM9DS1_REGISTER_TEMP_OUT_L, 2, buffer);
    uint8_t xlo = buffer[0];
    int16_t xhi = buffer[1];
    xhi <<= 8; xhi |= xlo;
    // Shift values to create properly formed integer (low byte first)
    temperature = xhi;
}

void Adafruit_LSM9DS1::setupAccel ( lsm9ds1AccelRange_t range )
{
    uint8_t reg = read8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG6_XL);
    reg &= ~(0b00011000);
    reg |= range;
    //Serial.println("set range: ");
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG6_XL, reg );
    switch (range)
    {
        case LSM9DS1_ACCELRANGE_2G:
            _accel_mg_lsb = LSM9DS1_ACCEL_MG_LSB_2G;
            break;
        case LSM9DS1_ACCELRANGE_4G:
            _accel_mg_lsb = LSM9DS1_ACCEL_MG_LSB_4G;
            break;
        case LSM9DS1_ACCELRANGE_8G:
            _accel_mg_lsb = LSM9DS1_ACCEL_MG_LSB_8G;
            break;
        case LSM9DS1_ACCELRANGE_16G:
            _accel_mg_lsb =LSM9DS1_ACCEL_MG_LSB_16G;
            break;
    }
}

void Adafruit_LSM9DS1::setupMag ( lsm9ds1MagGain_t gain )
{
    uint8_t reg = read8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG2_M);
    reg &= ~(0b01100000);
    reg |= gain;
    write8(MAGTYPE, LSM9DS1_REGISTER_CTRL_REG2_M, reg );
    switch(gain)
    {
        case LSM9DS1_MAGGAIN_4GAUSS:
            _mag_mgauss_lsb = LSM9DS1_MAG_MGAUSS_4GAUSS;
            break;
        case LSM9DS1_MAGGAIN_8GAUSS:
            _mag_mgauss_lsb = LSM9DS1_MAG_MGAUSS_8GAUSS;
            break;
```

```
        case LSM9DS1_MAGGAIN_12GAUSS:
            _mag_mgauss_lsb = LSM9DS1_MAG_MGAUSS_12GAUSS;
            break;
        case LSM9DS1_MAGGAIN_16GAUSS:
            _mag_mgauss_lsb = LSM9DS1_MAG_MGAUSS_16GAUSS;
            break;
    }
}

void Adafruit_LSM9DS1::setupGyro ( lsm9ds1GyroScale_t scale )
{
    uint8_t reg = read8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG1_G);
    reg &= ~(0b00011000);
    reg |= scale;
    write8(XGTYPE, LSM9DS1_REGISTER_CTRL_REG1_G, reg );
    switch(scale)
    {
        case LSM9DS1_GYROSCALE_245DPS:
            _gyro_dps_digit = LSM9DS1_GYRO_DPS_DIGIT_245DPS;
            break;
        case LSM9DS1_GYROSCALE_500DPS:
            _gyro_dps_digit = LSM9DS1_GYRO_DPS_DIGIT_500DPS;
            break;
        case LSM9DS1_GYROSCALE_2000DPS:
            _gyro_dps_digit = LSM9DS1_GYRO_DPS_DIGIT_2000DPS;
            break;
    }
}

void Adafruit_LSM9DS1::MadgwickQuaternionUpdate()
{
    float ax = accel.x, ay = accel.y,  az = accel.z;
    float gx = gyro.x * M_PI / 180, gy = gyro.y * M_PI / 180, gz = gyro.z *
     M_PI / 180;
    float mx = mag.x, my = mag.y, mz = mag.z;

    float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local
     variable for readability
    float norm;
    float hx, hy, _2bx, _2bz;
    float s1, s2, s3, s4;
    float qDot1, qDot2, qDot3, qDot4;

    // Auxiliary variables to avoid repeated arithmetic
    float _2q1mx, _2q1my, _2q1mz, _2q2mx;
    float _4bx, _4bz;
    float _2q1 = 2.0f * q1, _2q2 = 2.0f * q2, _2q3 = 2.0f * q3, _2q4 = 2.0f *
     q4;
    float _2q1q3 = 2.0f * q1 * q3, _2q3q4 = 2.0f * q3 * q4;
    float q1q1 = q1 * q1;
    float q1q2 = q1 * q2;
```

```
float q1q3 = q1 * q3;
float q1q4 = q1 * q4;
float q2q2 = q2 * q2;
float q2q3 = q2 * q3;
float q2q4 = q2 * q4;
float q3q3 = q3 * q3;
float q3q4 = q3 * q4;
float q4q4 = q4 * q4;

// Normalise accelerometer measurement
norm = sqrt(ax * ax + ay * ay + az * az);
if (norm == 0.0f) return; // handle NaN
norm = 1.0f/norm;
ax *= norm;
ay *= norm;
az *= norm;

// Normalise magnetometer measurement
norm = sqrt(mx * mx + my * my + mz * mz);
if (norm == 0.0f) return; // handle NaN
norm = 1.0f/norm;
mx *= norm;
my *= norm;
mz *= norm;

// Reference direction of Earth's magnetic field
_2q1mx = 2.0f * q1 * mx;
_2q1my = 2.0f * q1 * my;
_2q1mz = 2.0f * q1 * mz;
_2q2mx = 2.0f * q2 * mx;
hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 +
 _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;
hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my
 * q3q3 + _2q3 * mz * q4 - my * q4q4;
_2bx = sqrt(hx * hx + hy * hy);
_2bz = -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 +
 _2q3 * my * q4 - mz * q3q3 + mz * q4q4;
_4bx = 2.0f * _2bx;
_4bz = 2.0f * _2bz;

// Gradient decent algorithm corrective step
s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 -
 ay) - _2bz * q3 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) -
 mx) + (-_2bx * q4 + _2bz * q2) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 +
 q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 -
 q3q3) - mz);
```

```
    s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 -
     ay) - 4.0f * q2 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 *
     (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q3 +
     _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx *
     q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) -
     mz);
    s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 -
     ay) - 4.0f * q3 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 -
     _2bz * q1) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) +
     (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) -
     my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f -
     q2q2 - q3q3) - mz);
    s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 -
     ay) + (-_4bx * q4 + _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz *
     (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz * q3) * (_2bx * (q2q3 - q1q4) +
     _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 + q2q4) + _2bz *
     (0.5f - q2q2 - q3q3) - mz);
    norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    // normalise step
     magnitude
    norm = 1.0f/norm;
    s1 *= norm;
    s2 *= norm;
    s3 *= norm;
    s4 *= norm;

    // Compute rate of change of quaternion
    qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
    qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
    qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
    qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;

    // Integrate to yield quaternion
    q1 += qDot1 * timeSplit;
    q2 += qDot2 * timeSplit;
    q3 += qDot3 * timeSplit;
    q4 += qDot4 * timeSplit;
    norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise
     quaternion
    norm = 1.0f/norm;
    q[0] = q1 * norm;
    q[1] = q2 * norm;
    q[2] = q3 * norm;
    q[3] = q4 * norm;
}

/**************************************************************************
 PRIVATE FUNCTIONS
 **************************************************************************/
void Adafruit_LSM9DS1::write8(boolean type, byte reg, byte value)
{
    byte address, _cs;
```

```
    if (type == MAGTYPE) {
        address = LSM9DS1_ADDRESS_MAG;
        _cs = _csm;
    } else {
        address = LSM9DS1_ADDRESS_ACCELGYRO;
        _cs = _csxg;
    }
    _wire->beginTransmission(address);
    _wire->write(reg);
    _wire->write(value);
    _wire->endTransmission();

}

byte Adafruit_LSM9DS1::read8(boolean type, byte reg)
{
    uint8_t value;
    readBuffer(type, reg, 1, &value);
    return value;
}

byte Adafruit_LSM9DS1::readBuffer(boolean type, byte reg, byte len, uint8_t
 *buffer)
{
    byte address, _cs;

    if (type == MAGTYPE) {
        address = LSM9DS1_ADDRESS_MAG;
        _cs = _csm;
    } else {
        address = LSM9DS1_ADDRESS_ACCELGYRO;
        _cs = _csxg;
    }
    if (_i2c) {
        _wire->beginTransmission(address);
        _wire->write(reg);
        _wire->endTransmission();
        if (_wire->requestFrom(address, (byte)len) != len) {
            return 0;
        }
        for (uint8_t i=0; i<len; i++) {
            buffer[i] = _wire->read();
            //Serial.print(buffer[i], HEX); Serial.print(", ");
        }
        //Serial.println();
    }
    return len;
}
```

2 Testing and Development

2.1 Experimental Testing

- Performance Criterion

  One of our performance goals was decreasing the time of the robot take to get to the finish line by increasing straight-line velocity, therefore, to avoid collision of other robots. In order to do that we measured the time of our robot take to complete a 5m-straight-track starting form static.

- Experimental Parameter
  - The time of solenoid valve actuating state

- Control Parameters
  - The time of solenoid valve releasing state = 0.3s
  - Tire initial pressure = 30psi
  - Track length = 5m

- Data Collected

Actuating time (s)	1	2	3	4	5	6	7	8	9	10
	Time of completing the 5m-straight-track (s)									
0.4	6.9	6.3	7.1	7	7.2	6.8	6.9	7.1	7.2	7
0.5	6.3	6.5	6.1	6	6.7	6.2	6.1	6.3	6.4	6.2
0.6	6.6	6.3	6.7	6.5	6.5	6.7	6.6	6.2	6	6.9
0.8	7.1	7.5	6.3	6.8	7	7.1	7.2	6.8	6.9	7
1	8.2	7.2	7.8	8.3	8	7.4	8.2	7.6	8.4	8.6

- Result

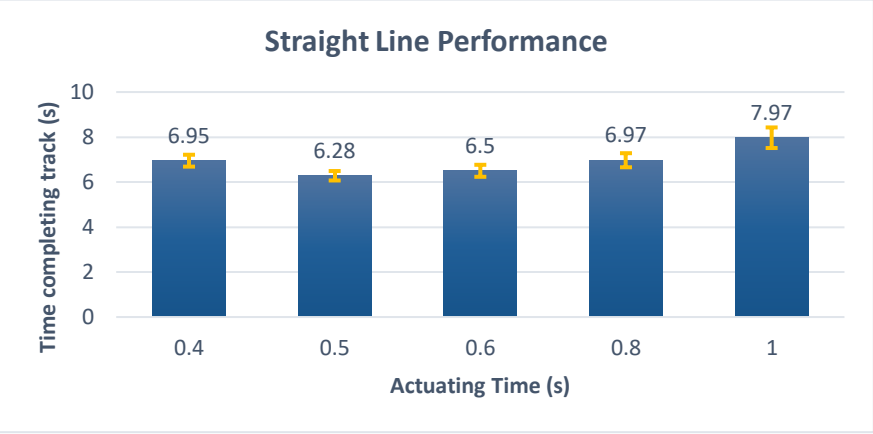

Figure 11. Data collected for straight line performance

Based on the data collected, an actuating time of 0.5 seconds will be used to achieve a shorter straight-line time since it has an average which is the least time usage of completing the short-distance track staring from steady, showing that this parameter value will allow the robot accelerate at a maximum.

2.2 Comparison with Mathematical Mode

- Simulated Impulse Response

  We measured one impulse response of the robot by making the robot fire once, and inputting the times it took to turn ¼ of the way of the wheel. We also measured the physical conditions of the robot including the radius of the wheel and size of the robot. The impulse response of the robot is:

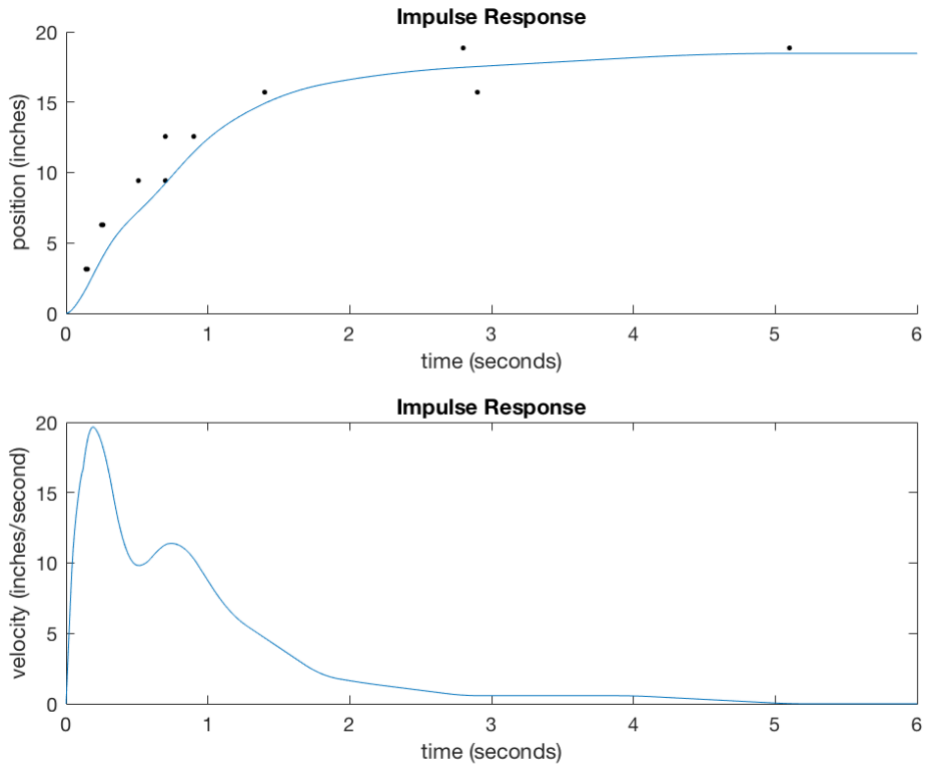

Figure 12. Simulated impulse response

- Actual Performance

Impluse	Displacement (m)	Time (s)	Velocity (m/s)	Simulator displacement (m)
0	0	0	0	0
1	0.2	0.69	0.289	0.22
2	0.37	1.29	0.287	0.38
3	0.52	1.75	0.298	0.52
4	0.78	2.62	0.298	0.79
5	0.92	3.24	0.284	0.92
6	1.12	4	0.28	1.14
7	1.38	4.76	0.29	1.4
8	1.62	5.7	0.284	1.62
9	1.88	6.84	0.275	1.88
10	2.09	7.5	0.28	2.11

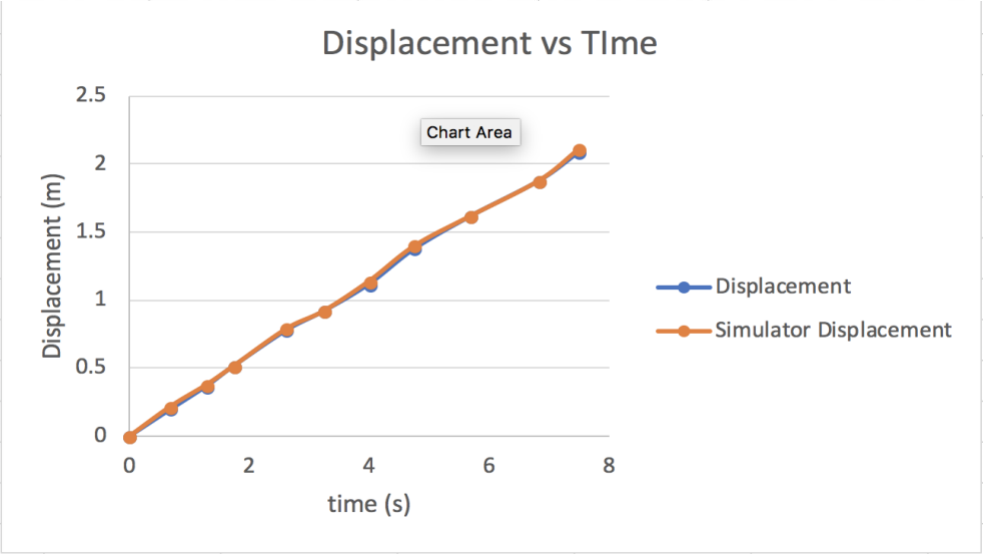

Figure 13. Actual performance vs. simulation

- Causes of Discrepancies

  The environment and the tire pressure caused the error. The wind and different ground surface can cause different friction to the robot, and the different frictions cause different velocities. The tire pressure decides the magnitude of the impulse.

3 Summary of Contributions

3.1 Team Members

- Mingqing Yuan (Mechanical)
  - ✓ Created CAD model
  - ✓ Fabricated and CNCed the wood and polycarbonate structure
  - ✓ 3D-printed of the steering parts
  - ✓ Helped test out the software code
- Ruby Gu (Electrical)
  - ✓ Wired up the circuit
  - ✓ Debugged problems with the broken magnetometer and the Arduino board
  - ✓ Bought the materials used for the wood and polycarbonate structure and tools for clamping-connection of the circuit
  - ✓ Helped test out the software code
- Qitai Meng (Control)
  - ✓ Programed the Arduino code
  - ✓ Calibrated and tested out the code for the robot to be able to run at different starting points
  - ✓ Concept design
  - ✓ Programed the code for CNCing our wood and polycarbonate structures
  - ✓ Helped debugged the circuit

3.2 Team Strategies

- Working Together
  We believe that the three roles are not separate from each other. The electrical person should understand the mechanism to set up the positions of servo motor and solenoid valve. The control member has to understand both mechanical and electrical concepts to find the most effective control method of our robot. For these reasons, even though we have different roles but we kept updating each other with details of every changes we make on the robot along the way.
- Think Widely and Not Afraid of Trying out New Methods
  Our team believes that think widely is necessary especially when we are facing problems on our robot. As we building the robot, a situation that we run into very often was that the reality didn't turn out the be what we expected when we designing it; and that's the time we needed to think widely and trying out new methods.