# Path planning robot with obstacle avoidance

A capstone project report submitted in partial fulfillment of the requirement for the degree of

**Masters of Engineering**

In the Department of Aerospace Engineering and Engineering Mechanics Graduate Program,
College of Engineering & Applied Science

11/24/2021

Rodriguez Arechabala, Olatz

## Index

## Figure index

## 1. Abstract

The aim of this project is to get a wheel robot to reach a destination successfully while avoiding any obstacles it may find in its way. The robot is equipped with an ultrasonic sensor to detect any obstacle and an Inertial Measurement Unit (IMU) that measures the velocity and acceleration to determine its position. To achieve that goal, a control system has been designed and implemented that creates a path from the starting point to the destination and, if it finds an obstacle in its way, it is able to avoid it and go back to the path.

## 2. Introduction

This document contains the necessary aspects for the development of a path planning program with obstacle avoidance using Arduino Uno robot kit. It has been structured in four parts.

The first part presents the context of the project, its objectives, and benefits. In this section it can be found the state of art of wheeled robots and the characteristics of the particular robot used and the software used to program it.

In the second part, the methodology followed to design and implement the program is collected.

Later, the results and the conclusions are gathered, including the problems faced.

Finally, this report includes an insight of future work that can be done to improve the program or take it to the next level of complexity. There is also an appendix with the code developed.

# 3. Context

## 3.1. Wheeled robots

A wheeled robot is a type of mobile robot, that is, a robot that can move from a location to another without human assistance.

The development of mobile robots answered the need to broaden the area of robotics applications, which had hitherto been limited by the scope of a mechanical structure attached at one of its ends. It is also about increasing autonomy and reducing human interference to the greatest extent possible.

They started being implemented in industry in the sixties with line following navigation. Nowadays, mobile robots have the intelligence to observe unknown environments and react and according to that.

## 3.2. Arduino

The robot used for this project is based on Arduino. Arduino is an open-source platform that is used for building electronics projects. It consists of both a microcontroller, a physical programmable circuit board, and an IDE, Integrated Development Environment, used to write and upload code into the board.

The Arduino family offers a while variety of boards, being Arduino Uno one of the most popular, that is, the one used on this project.

### 3.2.1. Hardware

The Arduino Uno microcontroller board contains 6 analog inputs, 14 digital input/output pins (six of which can be used as PWM outputs), a 16 MHz ceramic resonator, a power jack, a USB connection, a reset button and an ICSP header.

In order to build the robot, several sensors and actuators have been connected to the board, using a base to shape it.

The robot is what is called a differential wheeled robot. That means that the movement is based on two separately controlled wheels. There are two motors on either side of the robot connected each to a wheel. The motors receive a voltage from the board, and they can be set to different speeds on a range from 0 to 255, minimum and maximum speed respectively. There is also an additional wheel for support with free turning.

There is an IMU, Inertia Measurement Unit, which includes an accelerometer, a gyroscope and a magnetometer. The first, measures proper acceleration; the second, the orientation and, the third, the magnetic field. Combining the data of all three an approximation of the position and velocity can be obtained.

As the robot must be able to avoid obstacles it must first be able to detect them. An Ultrasonic sensor is used for that purpose. It uses ultrasonic sound waves to determine the distance to the nearest object.

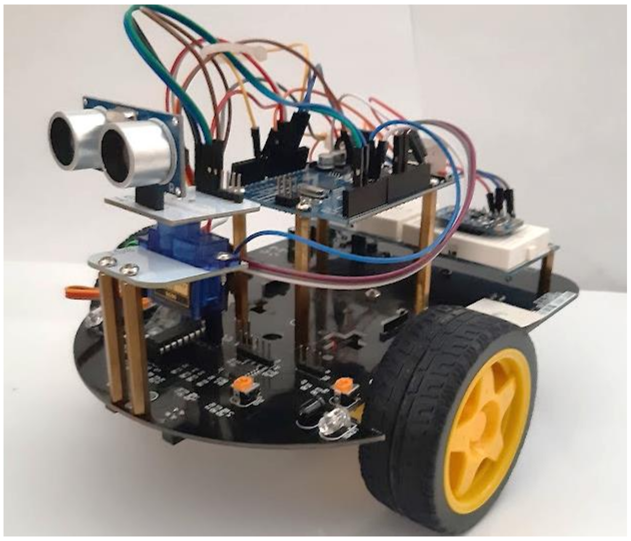

*Figure 1. Image of the robot used.*

### 3.2.2. Software

The Arduino Integrated Development Environment (IDE) is an open-source software used to write code and upload it to the board. It is written in functions from C and C++ and it also includes a software library with many common input and output procedures. The code is uploaded by connecting the board to the computer using a USB connection.

# 4. Implementation

## 4.1. Library

A library is a collection of functions that can be called several times during a program instead of having to implement them repeatedly. They are also useful to use them in different independent programs.

In this project a library has been created with basic wheeled robot functions so that they can be reused in other projects if there is a need for so.

There are three functions that have been added to the library: Moving, turn and distance.

#### Moving

This function is in charge of the forward and backwards movement of the robot. It asks for an input of the desired speed, in a range from -255 to 255. When the input is positive the robot will move forward and when it is negative it will move backwards. Using that information, the pins of the board connected to that movement of the robots will send that signal.

#### Turn

In order for the robot to turn, this function needs two information: the direction to turn and the speed at which it should be done. When turning to one side the opposite wheel must go faster than the inside one. The corresponding speeds are sent to the motors and the robot is able to turn.

#### Distance

The distance to an object must be read from the ultrasonic sensor. This information is gathered using two pins the echo pin and the trigger pin. First, an ultrasound is generated by setting the trigger pin to HIGH. Then, the travel time is measured using the `pulseIn()` function.

Knowing that the speed of the sound is 340 m/s, the distance to the object can be calculated with the following formula:

$$distance\ (cm) = duration \cdot \frac{0.034}{2}$$

## 4.2. Path planning

The aim of the project is to get to a certain goal. To do so, the robot must find the fastest path to arrive to the destination. To simplify, it has been considered that the robot can only move in either the x axis or the y axis at a time, that is, it cannot move diagonally.

The flowchart in Figure 2 shows the sequence of steps and decisions that the robot takes to reach the goal.

First, the robot goes along the y axis until it reaches the height of the goal. There, the robot decides to turn right or left. If the value of the goal in x axis is negative it will turn left and if it is positive it will turn right. Once turned, it will go along the x axis until it reaches the final destination.

To achieve so, the location of the robot must be known at all times. Since the robot does not include a GPS or any other sensor that will monitor the position, that information must be obtained using the IMU readings. The gyroscope, accelerometer and magnetometer readings can be used to extract the orientation of the robot. To do so, the Madgwick filter has been used, an algorithm developed by Sebastian Madgwick as part of his PhD research (Madgwick,n.d.). From it, the quaternion representation is derived and with them a rotation matrix can be formed to transform the accelerometer readings from the IMU body frame to the world frame:

$$xfmAccelerometerReading = R \cdot AccelerometerReading$$

Once the transformed accelerometer output is acquired, it can be used to obtain the speed using numeric integration, and the position integrating again:

$$speed += xfmAccelerometerReading \cdot \Delta Time$$

$$position += speed \cdot \Delta Time + 0.5 xfmAccelerometerReading \cdot \Delta Time \cdot \Delta Time$$

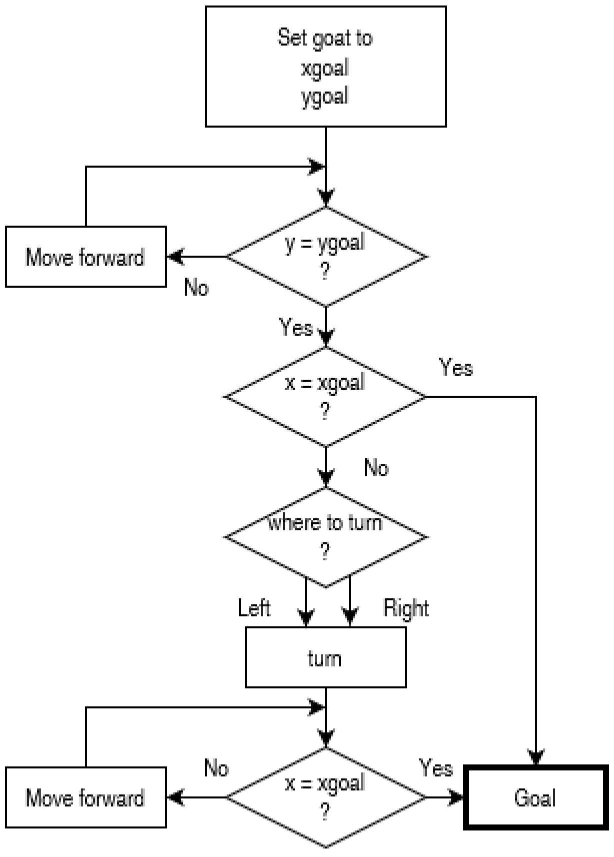

*Figure 2. No obstacles flowchart.*

## 4.3. Obstacle avoidance

There is an added complexity when the robot must also avoid obstacles in its way. In the path towards the goal there are two possibilities in which it can find an obstacle, when it is already done with the y axis and when it is not. In the first scenario, the robot must go around the object and go back to the original path. On the other one, however, it is more efficient to do a different path going through the x axis first. Every time the robot moves the distance to any object is read from the ultrasonic sensor, that way it will avoid being crashed.

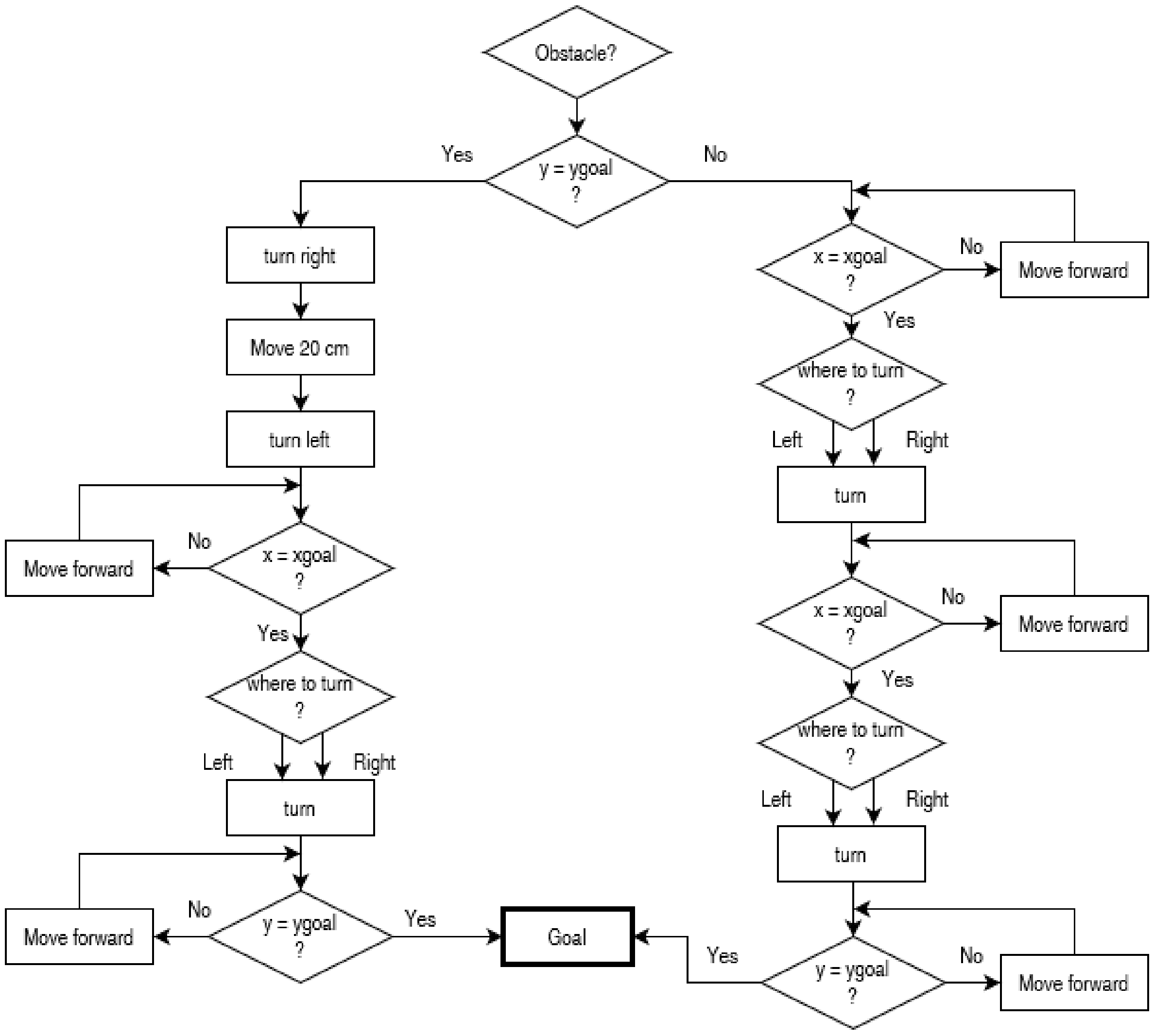

*Figure 3. Flowchart with obstacles.*

## 5. Results and conclusions

There were several problems that had to be overcame in the development of this project. First, there was a problem with getting the robot to go on a straight line. As explained earlier, there are two different motors that control each wheel. Theoretically, if both motors are set to the same speed, the robot should move in a straight line. In reality, there were some disparities between both motors, which caused one wheel to move faster than the other and making the robot turn slowly to one side. To solve this issue, a coefficient had to be implemented in the speed of the fastest wheel.

In addition, due to the fact that the position is obtained integrating, it will always have some kind of error that will increment with time. That is the reason why, a range of acceptability had to be set.

The project has been implemented in a wood floor. It has been tried with different goals and different positions for the obstacle. In overall, the robot was able to reach the goal within an error range of approximately 20 cm. The error increases the farther the goal is and it is also more precise if there is no obstacle in its way.

## 6. Future scope

This project leaves scope for many improvements. A very easy way to improve it would be adding a GPS device. As it is now, there is only so much accuracy that be obtained. An IMU is not very reliable for position estimation and a GPS would improve that. Furthermore, it would also allow this project to be done at a larger scale with the goal in a much further position.

Another addition that would help with the accuracy would be an IR proximity sensor. That would allow to count the RPM (revolutions per minute) of the wheels, making it possible to predict the reach of the robot better.

# 7. Bibliography

[1] Arduino Uno REV3. Arduino Online Shop. (n.d.). Retrieved November 29, 2021, from https://store-usa.arduino.cc/products/arduino-uno-rev3/?selectedStore=us.

[2] Banzi, M., &amp; Shiloh, M. (2014). Getting started with Arduino: The Open Source Electronics Prototyping Platform. Make Community.

[3] Madgwick , S. (n.d.). Open source imu and AHRS algorithms. x-io. Retrieved November 29, 2021, from https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/.

[4] Ollero Baturone Aníbal. (2007). Robótica: Manipuladores y robots móviles. Marcombo Boixareu.

[5] Tzafestas, S. G. (2014). Introduction to mobile robot control. Elsevier.

[6] What is an Arduino? Sparkfun. (n.d.). Retrieved November 29, 2021, from https://learn.sparkfun.com/tutorials/what-is-an-arduino/all.

# I. Appendix

This appendix includes the code that has been implemented in this project.

```
// ROBOT LIBRARY
// wrobot.h

#ifndef WROBOT_H
#define WROBOT_H

#include "Arduino.h"
#include <MatrixMath.h>
  class wrobot
  {
  public:
    wrobot (bool a);
    // Functions in the library:
    void moving(int wantedspeed);
    void turn(char dir,int wantedspeed);
    int distance();

  private:

  };
#endif
// wrobot.cpp

#include "Arduino.h"
#include "wrobot.h"
#include <math.h>

wrobot::wrobot(bool a)
{

}

void wrobot::moving(int wantedspeed) {
// Moves the robot forward or backwards
 int pin_r;
 int pin_l;

 if (wantedspeed >= 0) { // Positive speed -> forward
          pin_r = 5;
          pin_l = 6;

         }
         else { //  Negative speed -> backward
          pin_r = 10;
          pin_l = 11;
          wantedspeed = -wantedspeed;
         }
 pinMode(pin_r, OUTPUT);
 pinMode(pin_l, OUTPUT);
 if (wantedspeed >= 0 && wantedspeed <= 255) // Check that the speed is in the accepted
range
      {
       analogWrite(pin_r,wantedspeed);
       analogWrite(pin_l,0.84*wantedspeed);
      }
}

void wrobot::turn(char dir,int wantedspeed){
  // Turns in the direction requested
  // r - Right , l - left
  int pin_l = 5;
  int pin_r = 6;
  int speedr, speedl,ttime;

  pinMode(pin_r, OUTPUT);
  pinMode(pin_l, OUTPUT);
  analogWrite(pin_r,0); // First,the robot stops
  analogWrite(pin_l,0);
  delay(100);
  ttime = 850; // Time needed to turn 90°
  if (wantedspeed >= 0 && wantedspeed <= 255) // Check that the speed is in the accepted
range
      {
       if (dir=='r')
       {
         speedr = wantedspeed/3;
         speedl = wantedspeed;
       }
       if (dir=='l')
       {
         speedr = wantedspeed*1.03;
         speedl = wantedspeed/3;
       }
       analogWrite(pin_r,speedr);
       analogWrite(pin_l,speedl);
       delay(ttime);
       analogWrite(pin_r,0); // After turning it stops
       analogWrite(pin_l,0);
      }
}

int wrobot::distance(){
  // Measures the distance to any obstacles
  const int trigPin = 7; // Trigger Pin of Ultrasonic Sensor
  const int echoPin = 6; // Echo Pin of Ultrasonic Sensor
  long duration; // variable for the duration of sound wave travel
  int dist; // variable for the distance measurement

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
 // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);
 // Calculating the distance
 dist = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
 return dist;
}
// CAPSTONE PROJECT
// Path planning robot with obstacle avoidance
// Author: Olatz Rodriguez Arechabala
// Project.ino

#include "wrobot.h" // Created library
#include <MatrixMath.h> // Library for matrices operations
#include <math.h> // Library for mathematical operations
#include <Arduino_LSM9DS1.h> // IMU library
```

```
// Initialize "wrobot.h" library:

wrobot wrobot(true);


// VARIABLE DEFINITIONS


float pos[2]; // Position of the robot

float goal[2]; // Goal that wants to be reached


// MadgwickAHRSupdate definitions:

// Definitions

#define sampleFreq  512.0f    // sample frequency in Hz

#define betaDef   0.1f    // 2 * proportional gain

// Variable definitions

volatile float beta = betaDef;                // 2 * proportional gain (Kp)

volatile float q0 = 1.0f, q1 = 0.0f, q2 = 0.0f, q3 = 0.0f;  // quaternion of sensor
frame relative to auxiliary frame

long t=0;


// SET UP


void setup() {

  delay(5000);

  goal[0] = 50; // Goal

  goal[1] = 90;

  getposition(); // Getting initial position (0,0)

  delay(100);

}


// MAIN PROGRAM


void loop() {

  // Is the robot at the goal?

  thegoal:

    if (isatgoal(pos,1,goal)== true){

      if (isatgoal(pos,0,goal)==true){
        goto goalreached;

      }

     }


    while (isatgoal(pos,1,goal)== false){ // While the robot is not at the y axis goal
keep going

       if (obstacle() == true){ // If it finds an obstacle jump to ThereisanObstacle
later in the code

        goto ThereisanObstacle;

       }

       wrobot.moving(100);

       getposition(); // Keep checking position

    }

    wrobot.moving(0);

    getposition();

    wrobot.turn(wheretoturn(pos,0,goal),100); //Turn the robot to the direction to go
on x axis

    while (isatgoal(pos,0,goal)== false){ // Get to the goal on x axis

       if (obstacle() == true){  // Look for obstacles

        goto ThereisanObstacle;

       }

      wrobot.moving(100);

      getposition();

    }

    goto thegoal; // If there were no obstacles the robot should have reached the goal


    // The robot has faced an obstacle:

    ThereisanObstacle:

      wrobot.moving(0);

      getposition();

      // Has the robot already reached the goal on y axis?

      if (isatgoal(pos,1,goal)){ // If it has it must surround it

        wrobot.turn('r',100);

        wrobot.moving(100);

        delay(1000);

        wrobot.moving(0);

        getposition();
        wrobot.turn('l',100);
        getposition();
        while (isatgoal(pos,0,goal)== false){
          wrobot.moving(100);
          getposition();
        }
        wrobot.moving(0);
        wrobot.turn('l',100);
        while (isatgoal(pos,1,goal)== false){
          wrobot.moving(100);
          getposition();
        }
        wrobot.moving(0);
        goto thegoal;
      }
      else{ // If the robot was still on the y axis part of the path...
       wrobot.turn(wheretoturn(pos,0,goal),100); // It turn to the direction where the
goal on x axis is
       getposition();
       while (isatgoal(pos,0,goal)== false){ // Reach the x axis goal first
          wrobot.moving(100);
          getposition();
       }
       wrobot.moving(0);
       getposition();
       wrobot.turn(wheretoturn(pos,1,goal),100);
       getposition();
       while (isatgoal(pos,1,goal)== false){
          wrobot.moving(100);
          getposition();
       }
       goto thegoal;
      }
 goalreached:
    wrobot.moving(0); //Goal reached
}
```

```
// SUBPROGRAMS & FUNCTIONS

bool isatgoal(float posi[2], int xory, float goal[2]){ // 0: x - 1: y
  // Verifies if the robot is at the goal of one of the axis with a margin of 5 cm
  // If it is it returns true if not false
  bool atgoal;

  if ((posi[xory] < goal[xory] + 5.0)&(posi[xory] > goal[xory] - 5.0)){
    atgoal = true;
  }
  else{
    atgoal = false;
  }
}

char wheretoturn(float posi[2], int xory, float goal[2]){
  // It checks which direction to turn
  // If position (x or y) > goal (x or y) the robot must turn right
  // If position (x or y) < goal (x or y) the robot must turn left
  char rol; // Right or left

  if ((goal[xory] - posi[xory]) > 0){
    rol = 'r';
  }
  else{
    rol = 'l';
  }
}
bool obstacle(){
  // It verifies if the robot has an obstacle at a distance of 15 cm or closer
  // It returns true or false
  bool anobstacle;

  if (wrobot.distance()<15){
    anobstacle = true;
  }
  else{
    anobstacle = false;
  }
 return anobstacle;
}

void getposition(){
// Updates the position of the robot
// For that it turns the IMU readings into acceleration, speed and position
  float ax, ay, az, gx, gy, gz, mx, my, mz;
  long prevt;
  mtx_type R[3][3];
  mtx_type   readacc[3][1],acc[3][1],   deltaspeed[3][1],speed[3][1],   deltasp[3][1],
deltapos[3][1], position[3][1];
  mtx_type deltatime;

  prevt = t;
  t = millis();
  deltatime = (t - prevt)/1000; // Time passed

  // Initialize matrices
  for (int i = 0; i < 3; i++) {
      acc[i][0] = 0;
      speed[i][0] = 0;
      position[i][0] = 0;
  }
  // Read the IMU
  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(ax, ay, az);
  }

  if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(gx, gy, gz);
  }
  if (IMU.magneticFieldAvailable()) {
    IMU.readMagneticField(mx, my, mz);

  }

  MadgwickAHRSupdate(gx,gy,gz,ax,ay,az,mx,my,mz);  //  Function  to  obtain  quartenion
angles


  //Rotation matrix

  R[0][0] = 2*(sq(q0)+sq(q1))-1;

  R[0][1] = 2*(q1*q2-q0*q3);

  R[0][2] = 2*(q1*q3+q0*q2);

  R[1][0] = 2*(q1*q2+q0*q3);

  R[1][1] = 2*(sq(q0)+sq(q2))-1;

  R[1][2] = 2*(q2*q3-q0*q1);

  R[2][0] = 2*(q1*q3-q0*q2); ;

  R[2][1] = 2*(q2*q3+q0*q1);

  R[2][2] = 2*(sq(q0)+sq(q3))-1;


  readacc[0][0]=ax;

  readacc[1][0]=ay;

  readacc[2][0]=az;


  // acc = R*readacc

  Matrix.Multiply((mtx_type*)R, (mtx_type*)readacc, 3, 3, 1, (mtx_type*)acc);



  // speed += acc*deltatime

  MatrixConstant((mtx_type*) acc, 3, 1, deltatime,(mtx_type*) deltaspeed);

  Matrix.Add((mtx_type*) speed, (mtx_type*) deltaspeed, 3, 1, (mtx_type*) speed);



  //position += speed*deltatime +0.5 * acc * sq(deltatime)

  MatrixConstant((mtx_type*) speed, 3, 1, deltatime,(mtx_type*) deltasp);

  MatrixConstant((mtx_type*) acc, 3, 1, deltatime*deltatime*0.5,(mtx_type*) deltapos);

  Matrix.Add((mtx_type*) position, (mtx_type*) deltapos, 3, 1, (mtx_type*) position);


  pos[0]=position[0][0];

  pos[1]=position[1][0];
```

```
}



float invSqrt(float x) {

// Fast inverse square-root

// See: http://en.wikipedia.org/wiki/Fast_inverse_square_root

  float halfx = 0.5f * x;

  float y = x;

  long i = *(long*)&y;

  i = 0x5f3759df - (i>>1);

  y = *(float*)&i;

  y = y * (1.5f - (halfx * y * y));

  return y;

}


void MadgwickAHRSupdate(float gx, float gy, float gz, float ax, float ay, float az,
float mx, float my, float mz) {

  // Algorithm developed by Sebastian Madgwick (https://x-io.co.uk/open-source-imu-and-
ahrs-algorithms/)

  float recipNorm;

  float s0, s1, s2, s3;

  float qDot1, qDot2, qDot3, qDot4;

  float hx, hy;

  float _2q0mx, _2q0my, _2q0mz, _2q1mx, _2bx, _2bz, _4bx, _4bz, _2q0, _2q1, _2q2, _2q3,
_2q0q2, _2q2q3, q0q0, q0q1, q0q2, q0q3, q1q1, q1q2, q1q3, q2q2, q2q3, q3q3;


  // Rate of change of quaternion from gyroscope

  qDot1 = 0.5f * (-q1 * gx - q2 * gy - q3 * gz);

  qDot2 = 0.5f * (q0 * gx + q2 * gz - q3 * gy);

  qDot3 = 0.5f * (q0 * gy - q1 * gz + q3 * gx);

  qDot4 = 0.5f * (q0 * gz + q1 * gy - q2 * gx);


  // Compute feedback only if accelerometer measurement valid (avoids NaN in
accelerometer normalisation)

  if(!((ax == 0.0f) && (ay == 0.0f) && (az == 0.0f))) {
```

```
// Normalise accelerometer measurement
recipNorm = invSqrt(ax * ax + ay * ay + az * az);
ax *= recipNorm;
ay *= recipNorm;
az *= recipNorm;

// Normalise magnetometer measurement
recipNorm = invSqrt(mx * mx + my * my + mz * mz);
mx *= recipNorm;
my *= recipNorm;
mz *= recipNorm;

// Auxiliary variables to avoid repeated arithmetic
_2q0mx = 2.0f * q0 * mx;
_2q0my = 2.0f * q0 * my;
_2q0mz = 2.0f * q0 * mz;
_2q1mx = 2.0f * q1 * mx;
_2q0 = 2.0f * q0;
_2q1 = 2.0f * q1;
_2q2 = 2.0f * q2;
_2q3 = 2.0f * q3;
_2q0q2 = 2.0f * q0 * q2;
_2q2q3 = 2.0f * q2 * q3;
q0q0 = q0 * q0;
q0q1 = q0 * q1;
q0q2 = q0 * q2;
q0q3 = q0 * q3;
q1q1 = q1 * q1;
q1q2 = q1 * q2;
q1q3 = q1 * q3;
q2q2 = q2 * q2;
q2q3 = q2 * q3;
q3q3 = q3 * q3;

// Reference direction of Earth's magnetic field
```

```
    hx = mx * q0q0 - _2q0my * q3 + _2q0mz * q2 + mx * q1q1 + _2q1 * my * q2 + _2q1 * mz
* q3 - mx * q2q2 - mx * q3q3;

    hy = _2q0mx * q3 + my * q0q0 - _2q0mz * q1 + _2q1mx * q2 - my * q1q1 + my * q2q2 +
_2q2 * mz * q3 - my * q3q3;

    _2bx = sqrt(hx * hx + hy * hy);

    _2bz = -_2q0mx * q2 + _2q0my * q1 + mz * q0q0 + _2q1mx * q3 - mz * q1q1 + _2q2 * my
* q3 - mz * q2q2 + mz * q3q3;

    _4bx = 2.0f * _2bx;

    _4bz = 2.0f * _2bz;


    // Gradient decent algorithm corrective step

    s0 = -_2q2 * (2.0f * q1q3 - _2q0q2 - ax) + _2q1 * (2.0f * q0q1 + _2q2q3 - ay) - _2bz
* q2 * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q3 + _2bz *
q1) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q2 * (_2bx * (q0q2 +
q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);

    s1 = _2q3 * (2.0f * q1q3 - _2q0q2 - ax) + _2q0 * (2.0f * q0q1 + _2q2q3 - ay) - 4.0f
* q1 * (1 - 2.0f * q1q1 - 2.0f * q2q2 - az) + _2bz * q3 * (_2bx * (0.5f - q2q2 - q3q3)
+ _2bz * (q1q3 - q0q2) - mx) + (_2bx * q2 + _2bz * q0) * (_2bx * (q1q2 - q0q3) + _2bz *
(q0q1 + q2q3) - my) + (_2bx * q3 - _4bz * q1) * (_2bx * (q0q2 + q1q3) + _2bz * (0.5f -
q1q1 - q2q2) - mz);

    s2 = -_2q0 * (2.0f * q1q3 - _2q0q2 - ax) + _2q3 * (2.0f * q0q1 + _2q2q3 - ay) - 4.0f
* q2 * (1 - 2.0f * q1q1 - 2.0f * q2q2 - az) + (-_4bx * q2 - _2bz * q0) * (_2bx * (0.5f
- q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (_2bx * q1 + _2bz * q3) * (_2bx * (q1q2 -
q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q0 - _4bz * q2) * (_2bx * (q0q2 + q1q3) +
_2bz * (0.5f - q1q1 - q2q2) - mz);

    s3 = _2q1 * (2.0f * q1q3 - _2q0q2 - ax) + _2q2 * (2.0f * q0q1 + _2q2q3 - ay) + (-
_4bx * q3 + _2bz * q1) * (_2bx * (0.5f - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-
_2bx * q0 + _2bz * q2) * (_2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q1
* (_2bx * (q0q2 + q1q3) + _2bz * (0.5f - q1q1 - q2q2) - mz);

    recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3); // normalise step
magnitude

    s0 *= recipNorm;

    s1 *= recipNorm;

    s2 *= recipNorm;

    s3 *= recipNorm;


    // Apply feedback step

    qDot1 -= beta * s0;

    qDot2 -= beta * s1;

    qDot3 -= beta * s2;

    qDot4 -= beta * s3;

  }
  // Integrate rate of change of quaternion to yield quaternion
  q0 += qDot1 * (1.0f / sampleFreq);
  q1 += qDot2 * (1.0f / sampleFreq);
  q2 += qDot3 * (1.0f / sampleFreq);
  q3 += qDot4 * (1.0f / sampleFreq);

  // Normalise quaternion
  recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3);
  q0 *= recipNorm;
  q1 *= recipNorm;
  q2 *= recipNorm;
  q3 *= recipNorm;
}

void MatrixConstant(mtx_type* A, int m, int n, mtx_type k,mtx_type* B)
// Multiplies a matrix by a constant
{
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      B[n * i + j] = A[n * i + j] * k;
}

void arrayadd(float a[2],float b[2], float c[2]){
  // Adds 2 arrays
  for (int i=0;i<2;i++)
    c[i] = a[i] + b[i];
}
```