Compass Code: In-Depth Explanation

Introduction:

For this project we were asked to develop a way for the servomotor to correct its direction through the degrees with respect to the north established by the magnetometer, for this we must manage to create a base and understand how both the servomotor and the magnetometer work, likewise understand how to modify the different parameters of each part.

Prototype components:

Potentiometer

Consists of an adjustable resistor with three terminals, in which the resistance between two terminals varies when a dial or shaft is turned. It is commonly used to adjust brightness, volume, or position in electronic devices such as radios, amplifiers and lighting controls.

Functionality in the prototype:

- The potentiometer will act as an input, where the user will be able to modify the orientation to which the ship's rudder will steer.

Magnetometer

A magnetometer is a sensor that measures the strength and direction of a magnetic field in its environment. It is used in a variety of applications such as navigation, geology, geophysics

and in the electronic compass in smartphones to detect orientation with respect to the Earth's magnetic field.

Functionality in the prototype:

- The magnetometer will be constantly interacting with the magnetic field of the planet Earth, which will allow the instantaneous orientation at which the rudder is located to be identified.

LCD display

A Liquid Crystal Display (LCD) is a type of display that uses liquid crystals to show visual information. The liquid crystals can be electrically controlled to block or allow backlighting, allowing text, images and video to be displayed on a variety of devices, such as televisions, computer monitors, mobile phones and digital watches.

Functionality in the prototype:

- The LCD screen will act as an OUTPUT, whereby the user can identify the previously set orientation via the potentiometer.

Servomotor

A servomotor is a device that converts a control signal into precise motion in response to that signal. These motors are used in applications that require position and velocity control, such as robots, 3D printers, drones and industrial automation systems.

Functionality in the prototype:

- The servo motor will allow the rudder to be moved precisely so that the orientation obtained by the magnetometer matches the orientation proposed by the user.

Arduino

Development board containing a microcontroller and a series of input/output pins that allow the connection of sensors, actuators, and other electronic components. In turn, Arduino provides a development environment that facilitates the programming of microcontrollers.

Functionality in the prototype:

- The Arduino board together with the IDE, will allow to organize all the data obtained by the sensors or inputs, to be able to assign different positions to the servomotor.

Programming section:

Earlier in the description of the materials used for the development of the prototype, it was mentioned about the Arduino board. Arduino is a company in charge of the design and construction of hardware for the creation of digital devices. However, in order to achieve communication between hardware (arduino) and external devices (sensors), a programming environment is necessary for the execution of various tasks. For this reason, for the development of this prototype, Arduino IDE based on the C++ language was used.

The code used for the program is as follows:

- General libraries:

Libraries are a set of files, which can be provided through the Arduino IDE or, in turn, through third parties such as on GitHub repositories. Within the set of files, there are functions, classes, and objects, designed for developers to use and implement in their programs. To import libraries into the Arduino IDE, we use the reserved word "#include" with the name of the header file located in the folder of the imported library.

The following libraries were used for the prototype:

#include <LiquidCrystal.h>: It provides the necessary functions for the efficient use of an LCD display.

#include <QMC5883LCompass.h>: It provides the necessary functions for the configuration and reading of data from the magnetometer.

#include <Servo.h>: It provides the necessary functions for the rotation of the servomotor.

- Global variables:

We declare and initialise the sensors with the imported libraries, or by defining the pin where they are connected to the arduino.

```
LiquidCrystal lcd(8, 3, 4, 5, 6, 7);
#define potenciometro A1
Servo motor;
```

QMC5883LCompass compass;

We declare important variables for the development of the prototype.

float declinacion = -4.48; //Valor fundamental para obtener el norte geográfico.

int anguloServo = 90; // Start angle of the servomotor which will change angle as the sensor moves.

int contador = 0;// Variable that carries the time, according to the delay() function used in the program.

bool error = false;// Variables that "activate" to true, when the limit angles offered by the servomotor are exceeded.

- Execution code:

Section of code that is executed at program startup once at program startup.

```
void setup() {
  Serial.begin(9600);
  compass.init();
  // Inicializar el LCD con el número de  columnas y filas del LCD
  lcd.begin(16,3);
  lcd.print("Barco V1.0");
  compass.setCalibrationOffsets(-158.00, -129.00, 396.00);
  compass.setCalibrationScales(1.29, 0.91, 0.89);
  delay(5000);
  motor.attach(10);
  motor.write(0);
```

```
  delay(1000);
  motor.write(90);
  delay(1000);
}
```

- Looped code:

```
void loop() {
```

A loop is created that will be executed throughout the code.

```
int x, y, z;
 // Read compass values
 compass.read();
 // Return XYZ readings
 x = compass.getX();
 y = compass.getY();
 z = compass.getZ();
```

Magnetometer data are obtained in the x, y, and z axes.

```
float northMag = -((atan2(-x,-y) * 180)/ M_PI);
 int nortGeo = int(northMag + declinacion);
 if(nortGeo < 0)
 nortGeo +=360;
```

```
Serial.print("Brujula: ");
Serial.println(int(nortGeo));
```

The angle that determines the orientation of the magnetic north is obtained, followed by the declination, a value that varies depending on the place where the programme is run, obtaining the geographic north.

```
int input = analogRead(potenciometro);
input = map(input, 0, 1000, 0, 359);
if(input > 359){
  input = 359;
```

The potentiometer value is read and mapped to a range of 0 to 359 to represent the degrees that exist within a compass.

```
if(contador%100 == 0){
  lcd.clear();
  lcd.print("Bru:");
  lcd.print(nortGeo);
  lcd.print("       ");
  lcd.setCursor(0,1);
  lcd.print("Ent:");
  lcd.print(input);
  if(error){
```

```
    lcd.setCursor(10, 0);
    lcd.print("E");
  }
```

The LCD display is updated every 100 cycles of the loop, so you can avoid visual errors when loading data.

```
        int distancia1 = distanciaHoraria(nortGeo,input);
  int distancia2 = 360 - distancia1;
  bool giroHorario = (distancia1 <= distancia2)? true:false;
  if(giroHorario){
   anguloServo-=1;
   if(anguloServo < 0){
    error = true;
    anguloServo+=1;
   }else if(nortGeo == input){
    error = false;
    anguloServo+=1;
   }else{
    error = false;
    motor.write(anguloServo);
   }
```

```
  }else{
    anguloServo+=1;
    if(anguloServo >180 ){
      error = true;
      anguloServo-=1;
    }else if(nortGeo == input){
      error = false;
      anguloServo-=1;
    }else{
      error = false;
      motor.write(anguloServo);
    }
  }
  delay(100);
  contador+=10;
```

The hourly distances between magnetic north and the potentiometer value are calculated, followed by adjusting the location of the motor to match the magnetometer value.

```
int distanciaHoraria(int sensor, int input){
  int indice = 0;
  while(indice < 360){
    int info = indice + sensor;
```

```
    if(info >=360)
      info-=360;
    if(info == input)
      break;
    indice +=1;
   }
   return indice;
}
```

Function which determines the direction of the servo adjustment, either clockwise or counterclockwise, the purpose of this setting is to determine the fastest direction in which the motor can reach the direction sent, in the next section (Principle of operation) is explained in detail the function, which was previously created in Python to emulate it more easily.

Principle of operation.

The principle of the servomotor movement, so that the magnetometer (digital compass) coincides with the course set by the user via the potentiometer, is based on a distance calculation. For a deeper understanding of this principle, a Python simulator was made to estimate the possible correction movements of the servomotor.

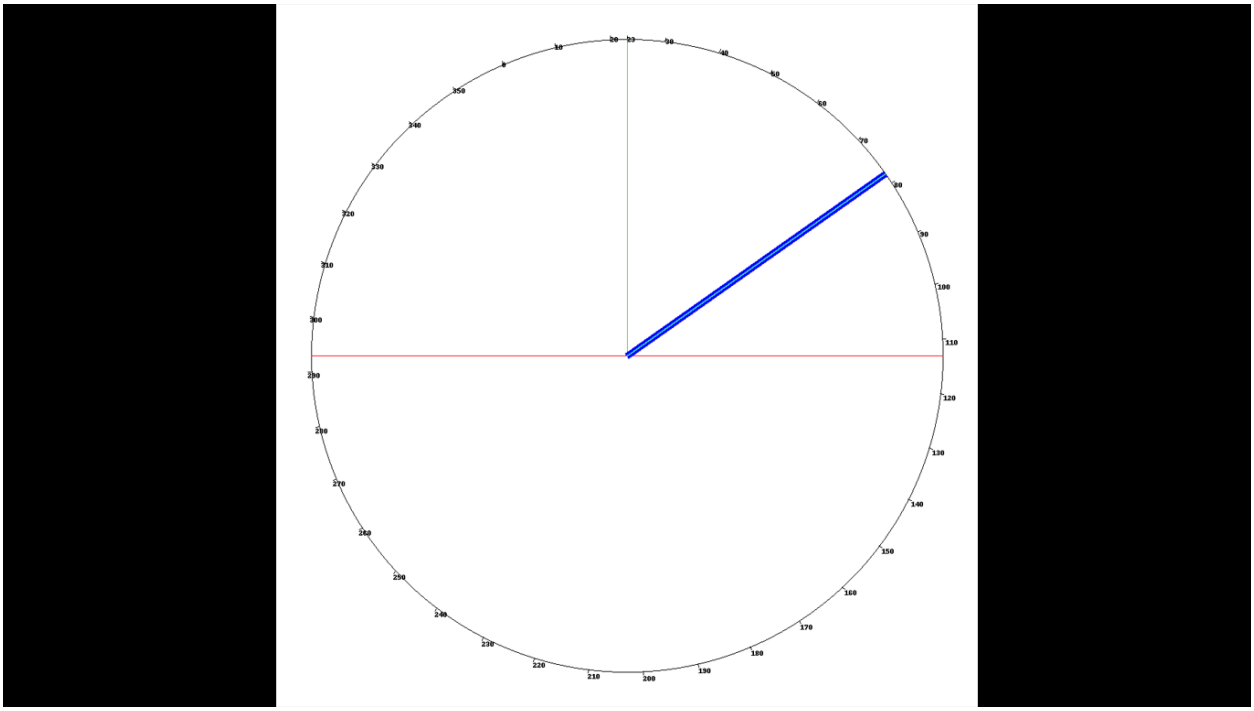
The following image is a result from the mentioned simulator. Where the values around the circle represent the possible data coming from the magnetometer. In order to identify the parts of the prototype, a colour scheme has been used:

1. Red colour: Represents the rotational limits of the servomotor [0-180].

2. Light green colour: This represents the initial position of the servomotor. It also indicates the initial position of the magnetometer.

3. Blue colour: This represents the course set by the user.

4. Turquoise colour: Represents the possible final position of the actuator.



Apart from the image, another result is presented which informs how the servomotor will act, so that the magnetometer data matches the user's assigned heading:

OutPut:

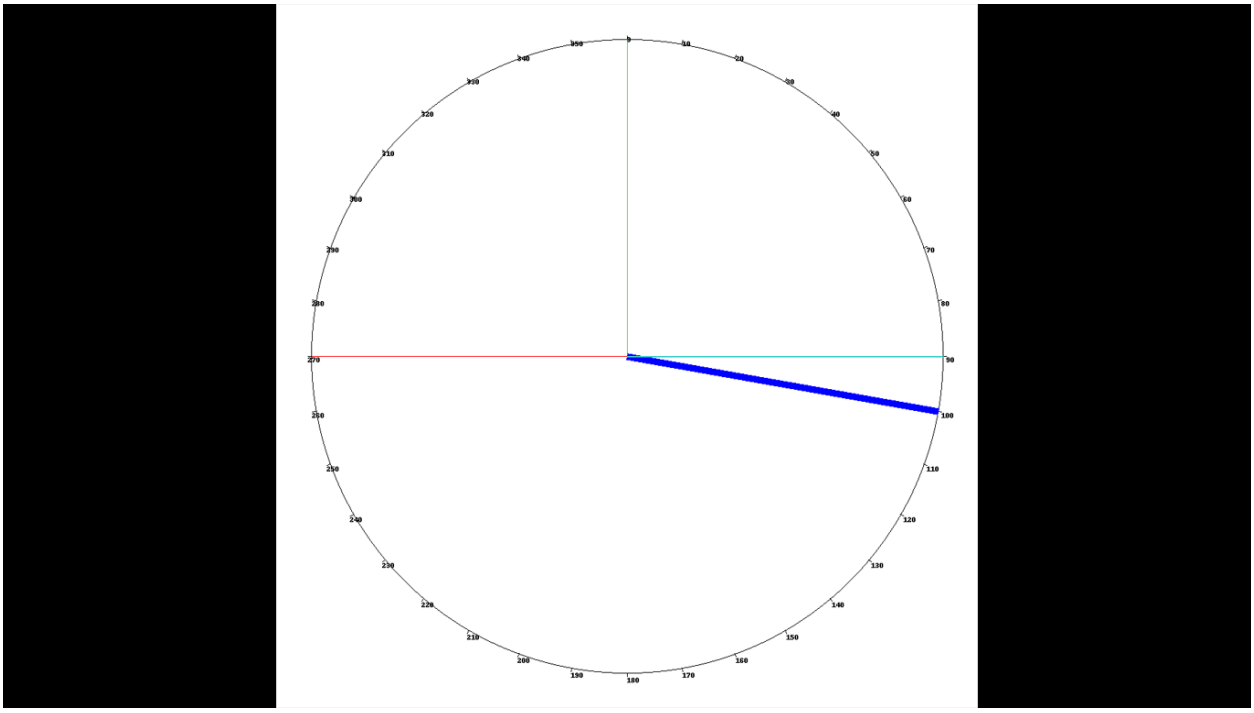
[Image].

Servo turns clockwise

Servo Angle: 33

Servo status: No limits have been exceeded.

The image shows how the servo motor will move as the user changes course, the distance calculation allows us to identify which direction of turn is appropriate to reach the user's proposed course quickly. If the heading is outside the turning limits of the servomotor, the simulator will show the following:



OutPut:

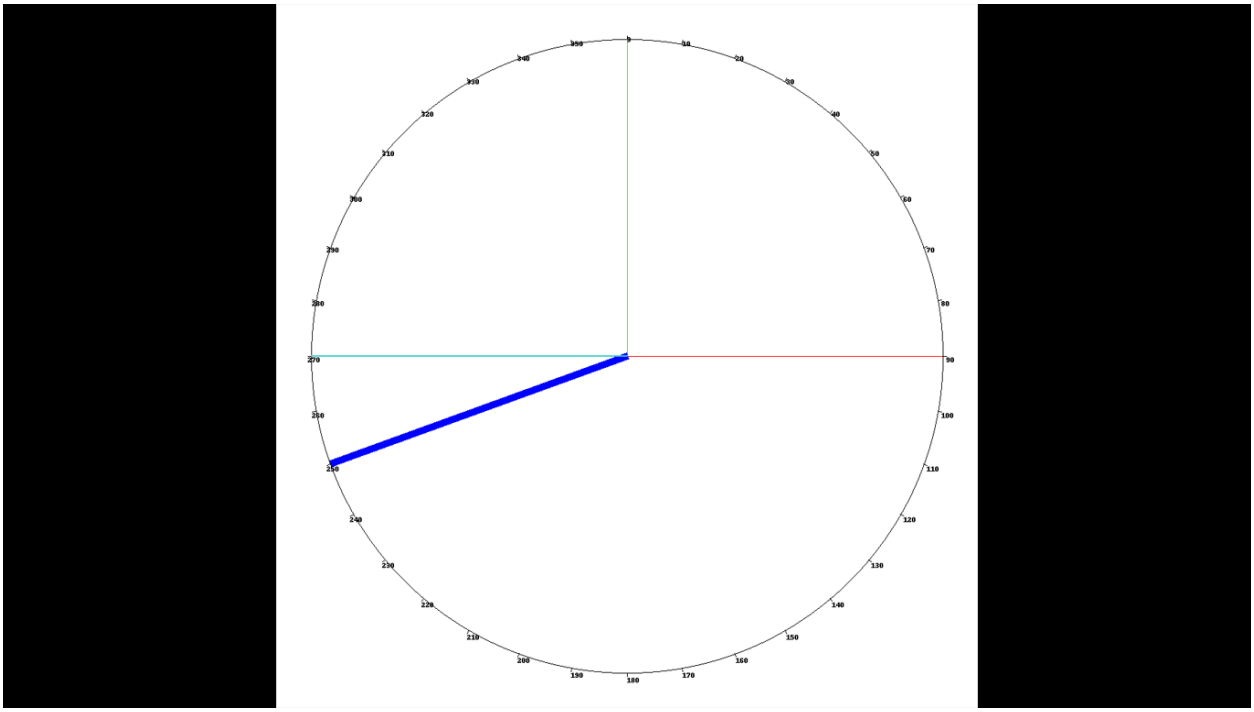
[image]

Servo rotates clockwise

Servo angle: 0

ERROR limits have been exceeded

Regardless of the error, we still find which direction of rotation is the most convenient and the servo position will be at the limit closest to the user set bearing.



Conclusion:

This project develops a navigation correction system based on servomotors and magnetometers. By integrating potentiometers, magnetometers, LCD displays, servo motors and an Arduino board, we built a prototype that responds to user inputs and accurately adjusts the steering direction.

Using the Arduino IDE and the C++ language, we developed a code library that enables efficient communication between the hardware and external actuators. In addition, we created a Python simulator to predict the corrective motion of the servomotor, which proved extremely useful during the design phase of the system.

The successful implementation of this project demonstrates that the combination of electronics and programming can create efficient navigation control systems. The applications of such systems are not limited to laboratory prototypes, but can also be extended to real navigation, positioning, and control systems, such as unmanned aerial vehicles and other automated equipment. Future work may include further optimizing the system's performance, reducing its size, and testing and validating it in a real-world environment.