Artificial Horizon

Introducere

*Proiectul utilizează senzorul MPU9250 9 axis pentru a crea un orizont artificial, indicând dinamic tangajul și ruliul în timp real, si poate include si funcționalitatea de cap compas cu ajutorul unei busole implementate prin citirea de date de pe magnetometrul senzorului.

*Scopul proiectului este de a oferi o experienta vizuala a orientării in spatiu a unui aparat de zbor, prin afisarea dinamica a tangajului, ruliului si giratie intr-un mod interactiv.

*Ideea mea a pornit de la dorinta de simula celebrul instrument de zbor, prezent pe toate aeronavele.

*Cred ca proiectul meu este util pentru altii deoarece poate fi folosit în scopuri educaționale sau practice in domeniul aviatiei si automatizării. Pentru mine, acest proiect reprezintă o modalitate de a explora și înțelege mai profund tehnologiile și conceptele din spatele orientarii în spatiu, in timp ce ne oferă satisfacția de a crea ceva functional si interactiv.

Descriere generală

Proiectul implică utilizarea unui Arduino Uno în combinație cu un senzor MPU9250 și un afișaj OLED i2c 128×64 pentru a crea un orizont artificial și o busolă care să fie afișate pe ecranul unui calculator folosind software-ul Processing. Acest sistem integrează hardware-ul și software-ul pentru a oferi o reprezentare vizuală intuitivă a orientării și direcției utilizatorului.

Componente Utilizate: Arduino Uno - platforma principală de microcontroler pentru citirea senzorilor și trimiterea datelor către PC. MPU9250 - senzorul care măsoară accelerația, giroscopul și câmpul magnetic pentru a determina orientarea (pitch, roll și heading). Adafruit OLED Display i2c 128×64 - afișaj pentru a vizualiza orizontul artificial și datele busolei. PC (Pentru GUI în Processing) - utilizat pentru a afisa datele într-o interfață grafică prietenoasă cu utilizatorul. Funcționalități Principale: Orizont Artificial:

Afișat pe OLED și în Processing, orizontul artificial indică înclinația (pitch) și rulajul (roll) al dispozitivului. Pe OLED, orizontul este reprezentat grafic cu o linie de orizont și un simbol al avionului. În Processing, orizontul este redat într-un mod grafic avansat, cu efecte de perspectivă. Busolă:

Calcularea și afișarea azimutului (heading) bazat pe datele magnetometrului din MPU9250. Afișat pe OLED și în Processing, busola indică direcția nordului și este completată cu marcaje pentru direcții cardinale și intercardinale. Fluxul de Date și Funcționarea: Arduino:

Citește datele de la senzorul MPU9250 (accelerometru, giroscop și magnetometru). Procesează datele pentru a calcula pitch, roll și heading. Afișează rezultatele pe ecranul OLED și trimite datele prin serial către PC. Processing:

Primește datele prin portul serial de la Arduino. Procesează și afișează grafic orizontul artificial și busola într-o fereastră grafică.

Hardware Design

-Arduino UNO Rev3, ATmega328P

-MPU-9250 9 axis Sensor

-Jumper wires male to male

-Adafruit OLED I2C Display 128×64

-Breadboard

Software Design

*Medii de dezvoltare: Arduino IDE, Processing

libs:

```
      github.com/bolderflight/invensense-imu
      github.com/adafruit/Adafruit_SSD1306
      github.com/adafruit/Adafruit_BusIO
      github.com/adafruit/Adafruit-GFX-Library
```

Rezultate Obţinute

Concluzii

Proiectul de creare a unui orizont artificial și a unei busole utilizând Arduino și Processing a fost un succes partial, demonstrând capabilitățile senzorului MPU9250 în monitorizarea orientării în spațiu. Implementarea a oferit o experiență vizuală intuitivă și interactivă, utilă atât pentru educație, cât și pentru aplicații practice în domeniul aviației. În viitor, proiectul poate fi extins pentru a include mai multe funcționalități și optimizări. Probleme intampinate: nu am reusit sa construiesc o busola care sa redea exact azimutul corect.

Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

cod arduino

#include <Wire.h> #include <Adafruit_GFX.h> #include <Adafruit_SSD1306.h> #include “mpu9250.h”

OLED display settings #define SCREEN_WIDTH 128 #define SCREEN_HEIGHT 64 #define OLED_RESET -1 #define OLED_I2C_ADDRESS 0x3C Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET); MPU9250 settings bfs::Mpu9250 imu;

Calibration offsets float pitchOffset = 0; float rollOffset = 0; void setup() { Serial.begin(115200); while (!Serial); Wait for Serial to be ready

```
  Wire.begin();
```

```
  // Initialize the MPU9250 sensor with the primary I2C address
  imu.Config(&Wire, bfs::Mpu9250::I2C_ADDR_PRIM);
  imu.Begin();
```

```
  // Optionally configure sensor ranges and other settings
  imu.ConfigAccelRange(bfs::Mpu9250::ACCEL_RANGE_2G);
  imu.ConfigGyroRange(bfs::Mpu9250::GYRO_RANGE_250DPS);
  imu.ConfigDlpfBandwidth(bfs::Mpu9250::DLPF_BANDWIDTH_184HZ);
```

```
  Serial.println("MPU9250 initialized and configured.");
```

```
  // Initialize OLED display
  if (!display.begin(SSD1306_SWITCHCAPVCC, OLED_I2C_ADDRESS)) {
      Serial.println(F("SSD1306 allocation failed"));
      for (;;);
  }
  display.display();
  delay(2000);
  display.clearDisplay();
```

```
  // Calibration step: capture initial offsets
  if (imu.Read()) {
      pitchOffset = atan2(imu.accel_y_mps2(), sqrt(imu.accel_x_mps2() *
imu.accel_x_mps2() + imu.accel_z_mps2() * imu.accel_z_mps2())) * 180.0 / PI;
      rollOffset = atan2(-imu.accel_x_mps2(), imu.accel_z_mps2()) * 180.0 /
PI;
  }
```

```
}
```

void loop() {

```
  // Update sensor readings
  if (!imu.Read()) {
      Serial.println("Failed to read from MPU9250!");
      return;
  }
```

```
  // Get accelerometer and gyroscope data
  float accelX = imu.accel_x_mps2();
  float accelY = imu.accel_y_mps2();
  float accelZ = imu.accel_z_mps2();
```

```
  // Get magnetometer data
  float magX = imu.mag_x_ut();
  float magY = imu.mag_y_ut();
```

```
  // Calculate pitch, roll, and heading
  float pitch = atan2(accelY, sqrt(accelX * accelX + accelZ * accelZ)) *
180.0 / PI - pitchOffset;
  float roll = atan2(-accelX, accelZ) * 180.0 / PI - rollOffset;
  float heading = atan2(magY, magX) * 180.0 / PI;
  //Normalize heading between 0-360 degrees
  if (heading < 0) {
      heading += 360;
  }
  // Normalize roll to be within -180 to 180 degrees
  if (roll > 180) {
      roll -= 360;
```

```
  } else if (roll < -180) {
      roll += 360;
  }
  // Display the results on Serial Monitor
  Serial.print(roll, 2);
  Serial.print(" ");
  Serial.print(pitch, 2);
  Serial.print(" ");
  Serial.print(heading, 2);
  Serial.println();
```

```
  // Redraw the horizon and airplane only if pitch or roll changes
significantly
  static float lastPitch = 0;
  static float lastRoll = 0;
  static float lastheading = 0;
  if (abs(lastPitch - pitch) > 0.01 || abs(lastRoll - roll) > 0.01 ||
abs(lastheading - heading) > 0.01) {
      display.clearDisplay();  // Clear the display only when necessary
      drawArtificialHorizon(roll, pitch);
      drawAirplaneSymbol();
      drawAzimuth(heading);
      display.display();
      lastPitch = pitch;
      lastRoll = roll;
      lastheading = heading;
  }
```

}

void drawArtificialHorizon(float roll, float pitch) {

```
  int centerX = SCREEN_WIDTH / 2;
  int centerY = SCREEN_HEIGHT / 2;
```

```
  // Convert pitch and roll to radians
  float pitchRad = pitch * PI / 180.0;
  float rollRad = roll * PI / 180.0;
```

```
  // Calculate the y-position of the horizon line based on pitch
  int horizonY = centerY - (int)(pitchRad * 33); // scale pitch to screen
coordinates
```

```
  // Calculate the x-offset of the horizon line based on roll
  int rollOffset = (int)(rollRad * (SCREEN_WIDTH / 2)); // scale roll to
screen coordinates
```

```
  // Draw the horizon line
  display.drawLine(0, horizonY + rollOffset, SCREEN_WIDTH, horizonY -
rollOffset, SSD1306_WHITE);
```

```
  // Draw perspective lines
  display.drawLine(0, horizonY + rollOffset, centerX, SCREEN_HEIGHT,
SSD1306_WHITE);
  display.drawLine(SCREEN_WIDTH, horizonY - rollOffset, centerX,
SCREEN_HEIGHT, SSD1306_WHITE);
```

```
  // Draw additional lines parallel to the horizon line for perspective
effect
```

```
}
```

void drawAirplaneSymbol() {

```
  int centerX = SCREEN_WIDTH / 2;
  int centerY = SCREEN_HEIGHT / 2;
```

```
  display.drawLine(centerX - 5, centerY, centerX + 5, centerY,
SSD1306_WHITE); // Body
  display.drawLine(centerX - 10, centerY, centerX - 5, centerY - 3,
SSD1306_WHITE); // Left wing up
  display.drawLine(centerX + 10, centerY, centerX + 5, centerY - 3,
SSD1306_WHITE); // Right wing up
  display.drawLine(centerX - 10, centerY, centerX - 5, centerY + 3,
SSD1306_WHITE); // Left wing down
  display.drawLine(centerX + 10, centerY, centerX + 5, centerY + 3,
SSD1306_WHITE); // Right wing down
  display.drawLine(centerX, centerY - 2, centerX, centerY - 7,
SSD1306_WHITE); // Vertical tail
```

```
}
```

void drawAzimuth(float heading) {

```
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(32, 0); // Top center
  display.print("Bearing: ");
  display.print(heading, 1);
```

```
}
```

cod processing

import processing.serial.*;

float Pitch; float Bank; float Azimuth; float ArtificialHoizonMagnificationFactor = 0.7; float CompassMagnificationFactor = 0.85; float SpanAngle = 120; int NumberOfScaleMajorDivisions; int NumberOfScaleMinorDivisions; PVector v1, v2;

Serial port; float Phi; float Theta; float Psi;

void setup() {

```
size(1366, 768);
rectMode(CENTER);
smooth();
strokeCap(SQUARE);
```

```
println(Serial.list());
port = new Serial(this, Serial.list()[0], 115200);
port.bufferUntil('\n');
```

}

void draw() {

```
background(0, 0, 102); // Dark blue background
translate(W / 4, H / 2.1);
MakeAnglesDependentOnMPU9250();
Horizon();
rotate(-Bank);
PitchScale();
Axis();
rotate(Bank);
Borders();
Plane();
Compass();
```

}

void serialEvent(Serial port) *Reading the data by Processing. { String input = port.readStringUntil('\n'); if (input != null) { input = trim(input); String[] values = split(input, ” ”); if (values.length == 3) { try { float phi = float(values[0]); float theta = float(values[1]); float psi = float(values[2]); println(phi, theta, psi); Phi = phi; Theta = theta; Psi = psi; } catch (Exception e) { println(“Error parsing values: ” + e.getMessage()); } } } } void MakeAnglesDependentOnMPU9250() { Bank = radians(-Phi);* Convert degrees to radians for Processing

```
Pitch = Theta;
Azimuth = Psi;
```

}

void Horizon() {

```
scale(ArtificialHoizonMagnificationFactor);
noStroke();
```

```
fill(0, 180, 255);
rect(0, -100, 900, 1000);
fill(95, 55, 40);
pushMatrix(); // Save the current transformation matrix
rotate(Bank);
rect(0, 400 + Pitch * 10, 900, 800);
popMatrix(); // Restore the previous transformation matrix
rotate(-PI - PI / 6);
SpanAngle = 120;
NumberOfScaleMajorDivisions = 12;
NumberOfScaleMinorDivisions = 24;
CircularScale();
rotate(PI + PI / 6);
rotate(-PI / 6);
CircularScale();
rotate(PI / 6);
```

}

void Compass() {

```
translate(2 * W / 3, 0);
scale(CompassMagnificationFactor);
noFill();
stroke(100);
strokeWeight(80);
ellipse(0, 0, 750, 750);
strokeWeight(50);
stroke(50);
fill(0, 0, 40);
ellipse(0, 0, 610, 610);
for (int k = 255; k > 0; k = k - 5)
{
  noStroke();
  fill(0, 0, 255 - k);
  ellipse(0, 0, 2 * k, 2 * k);
}
strokeWeight(20);
NumberOfScaleMajorDivisions = 18;
NumberOfScaleMinorDivisions = 36;
SpanAngle = 180;
CircularScale();
rotate(PI);
SpanAngle = 180;
CircularScale();
rotate(-PI);
fill(255);
textSize(60);
textAlign(CENTER);
text("W", -375, 0, 100, 80);
```

```
text("E", 370, 0, 100, 80);
text("N", 0, -365, 100, 80);
text("S", 0, 375, 100, 80);
textSize(30);
text("COMPASS", 0, -130, 500, 80);
rotate(PI / 4);
textSize(40);
text("NW", -370, 0, 100, 50);
text("SE", 365, 0, 100, 50);
text("NE", 0, -355, 100, 50);
text("SW", 0, 365, 100, 50);
rotate(-PI / 4);
CompassPointer();
```

}

void CompassPointer() {

```
rotate(PI + radians(Azimuth));
stroke(0);
strokeWeight(4);
fill(100, 255, 100);
triangle(-20, -210, 20, -210, 0, 270);
triangle(-15, 210, 15, 210, 0, 270);
ellipse(0, 0, 45, 45);
fill(0, 0, 50);
noStroke();
ellipse(0, 0, 10, 10);
triangle(-20, -213, 20, -213, 0, -190);
triangle(-15, -215, 15, -215, 0, -200);
rotate(-PI - radians(Azimuth));
```

}

void Plane() {

```
fill(0);
strokeWeight(1);
stroke(0, 255, 0);
triangle(-20, 0, 20, 0, 0, 25);
rect(110, 0, 140, 20);
rect(-110, 0, 140, 20);
```

}

void CircularScale() {

```
float GaugeWidth = 800;
textSize(GaugeWidth / 30);
float StrokeWidth = 1;
float an;
```

```
float DivxPhasorCloser;
float DivxPhasorDistal;
float DivyPhasorCloser;
float DivyPhasorDistal;
strokeWeight(2 * StrokeWidth);
stroke(255);
float DivCloserPhasorLenght = GaugeWidth / 2 - GaugeWidth / 9 - StrokeWidth;
float DivDistalPhasorLenght = GaugeWidth / 2 - GaugeWidth / 7.5 -
StrokeWidth;
for (int Division = 0; Division < NumberOfScaleMinorDivisions + 1;
Division++)
{
  an = SpanAngle / 2 + Division * SpanAngle / NumberOfScaleMinorDivisions;
  DivxPhasorCloser = DivCloserPhasorLenght * cos(radians(an));
  DivxPhasorDistal = DivDistalPhasorLenght * cos(radians(an));
  DivyPhasorCloser = DivCloserPhasorLenght * sin(radians(an));
  DivyPhasorDistal = DivDistalPhasorLenght * sin(radians(an));
  line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
DivyPhasorDistal);
}
DivCloserPhasorLenght = GaugeWidth / 2 - GaugeWidth / 10 - StrokeWidth;
DivDistalPhasorLenght = GaugeWidth / 2 - GaugeWidth / 7.4 - StrokeWidth;
for (int Division = 0; Division < NumberOfScaleMajorDivisions + 1;
Division++)
{
  an = SpanAngle / 2 + Division * SpanAngle / NumberOfScaleMajorDivisions;
  DivxPhasorCloser = DivCloserPhasorLenght * cos(radians(an));
  DivxPhasorDistal = DivDistalPhasorLenght * cos(radians(an));
  DivyPhasorCloser = DivCloserPhasorLenght * sin(radians(an));
  DivyPhasorDistal = DivDistalPhasorLenght * sin(radians(an));
  if (Division == NumberOfScaleMajorDivisions / 2 || Division == 0 ||
Division == NumberOfScaleMajorDivisions)
  {
    strokeWeight(15);
    stroke(0);
    line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
DivyPhasorDistal);
    strokeWeight(8);
    stroke(100, 255, 100);
    line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
DivyPhasorDistal);
  }
  else
  {
    strokeWeight(3);
    stroke(255);
    line(DivxPhasorCloser, DivyPhasorCloser, DivxPhasorDistal,
DivyPhasorDistal);
  }
}
```

```
}
```

void Axis() {

```
stroke(255, 0, 0);
strokeWeight(3);
line(-115, 0, 115, 0);
line(0, 280, 0, -280);
fill(100, 255, 100);
stroke(0);
triangle(0, -285, -10, -255, 10, -255);
triangle(0, 285, -10, 255, 10, 255);
```

```
}
```

void Borders() {

```
noFill();
stroke(0);
strokeWeight(400);
rect(0, 0, 1100, 1100);
strokeWeight(200);
ellipse(0, 0, 1000, 1000);
fill(0);
noStroke();
rect(4 * W / 5, 0, W, 2 * H);
rect(-4 * W / 5, 0, W, 2 * H);
```

```
}
```

void PitchScale() {

```
stroke(255);
fill(255);
strokeWeight(3);
textSize(24);
textAlign(CENTER);
for (int i = -4; i < 5; i++)
{
  if ((i == 0) == false)
  {
    line(110, 50 * i, -110, 50 * i);
  }
  text("" + i * 10, 140, 50 * i, 100, 30);
  text("" + i * 10, -140, 50 * i, 100, 30);
}
textAlign(CORNER);
strokeWeight(2);
for (int i = -9; i < 10; i++)
{
  if ((i == 0) == false)
```

```
  {
    line(25, 25 * i, -25, 25 * i);
  }
}
```

}

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.

Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

Insipiratie: https://mfurkanbahat.blogspot.com/2014/11/artificial-horizon-and-compass-using.html

Datasheet mpu9250:
https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

Export to PDF