實習專題報告

新生兒髖關節超音波檢測

許博森 黃子庭

困境

困境

1. 通信：時常斷訊、變動性大，造成往後使用上無法檢測或設定問題

- 硬件測試 => 已焊接及固定
  - ✓ USB Micro線
  - ✓ MPU6050
  - ✓ Arduino UNO
- 目前利用Reset鍵不斷重啟

系統介紹

研究流程

數據處理與分析

實際測試

1

2

3

4

硬體設定

姿態估計

系統概括

感測器採樣

以Arduino UNO 控制
MPU6050六軸感測器
測量繞軸旋轉的變化量

數據處理

將採集到的數據進行前置處理
並運算旋轉情形

顯示姿態估計

匯出姿態估計結果
並顯示兩感測器的相對狀態

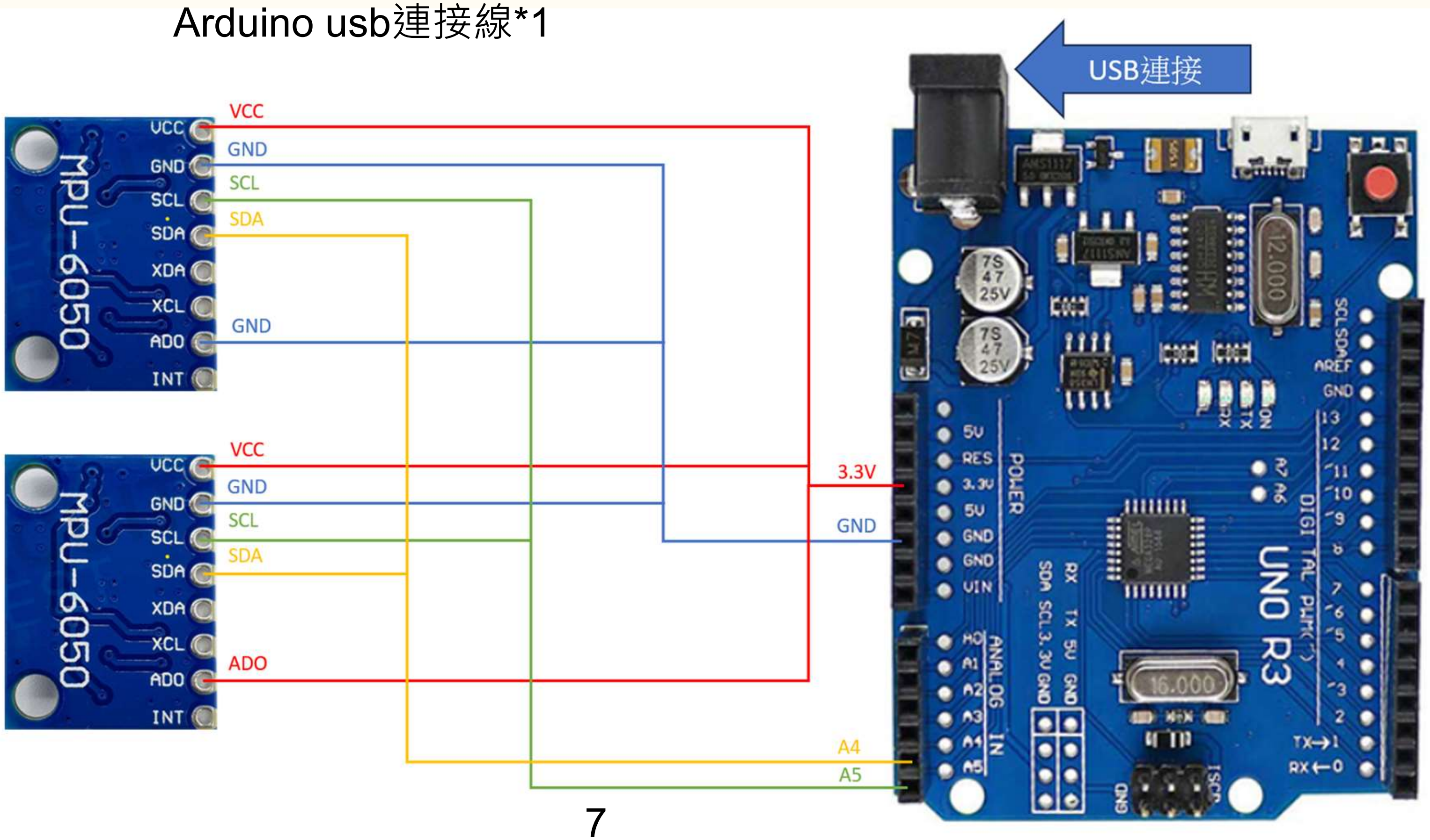
硬體設定

硬體設備

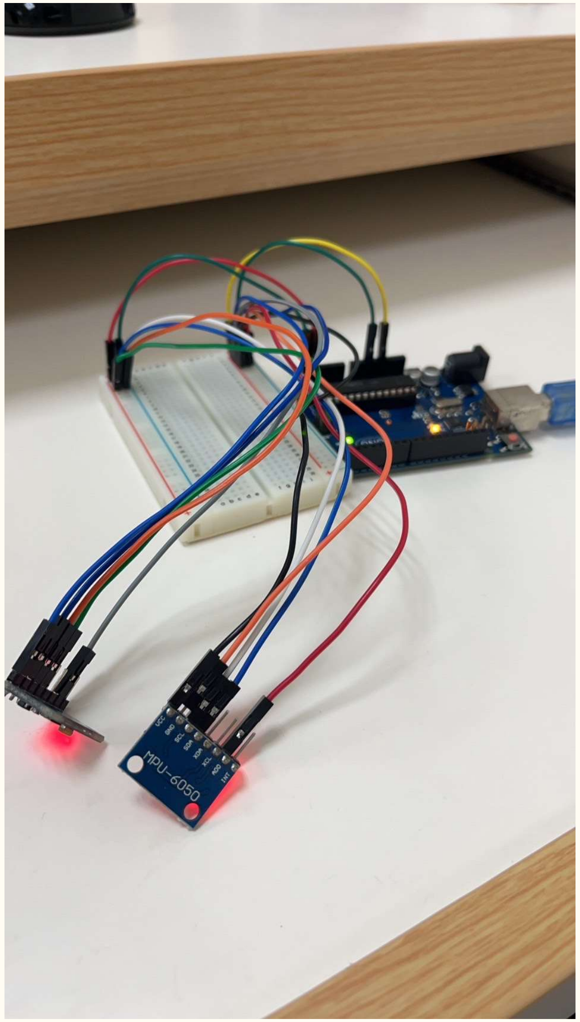
Arduino UNO R3板*1
MPU6050加速度陀螺儀傳感器*2
跳線數條
麵包板*1
Arduino usb連接線*1

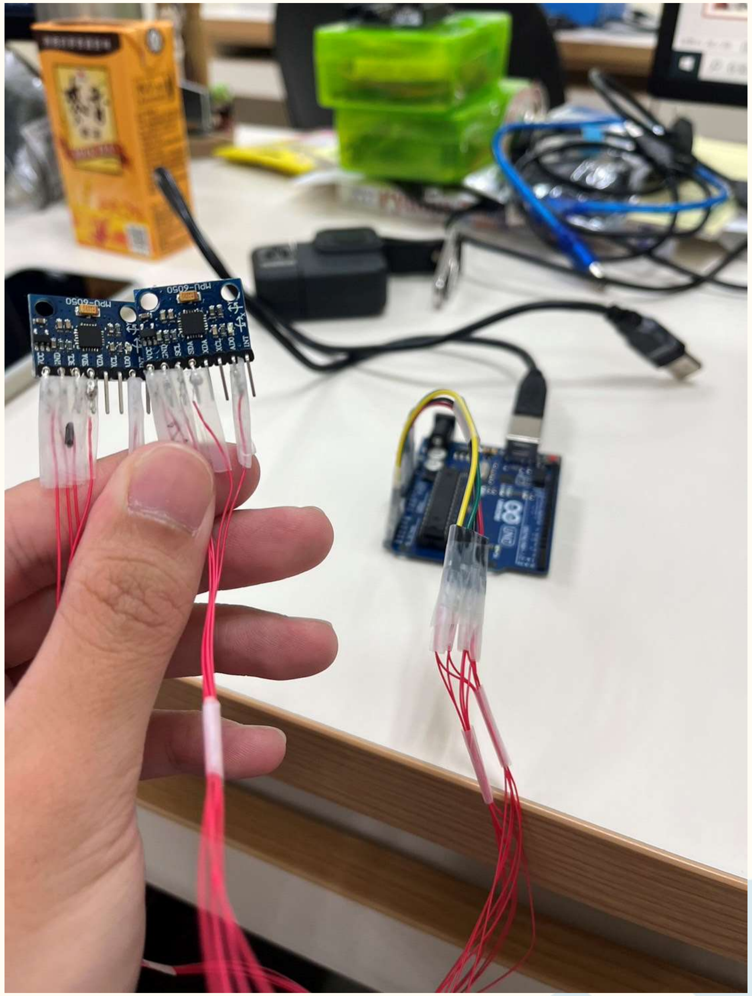
Sensor1
(新生兒髖部)

Sensor2
(超音波探頭)



硬體設備





硬體設定

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu1;
MPU6050 mpu2;

// Timers
unsigned long timer = 0;
float timeStep = 0.01;

// Pitch, Roll and Yaw values
float pitch1 = 0, roll1 = 0, yaw1 = 0;
float pitch2 = 0, roll2 = 0, yaw2 = 0;
```

```
void setup()
{
  Serial.begin(115200);

  // Initialize MPU6050 mpu1
  while(!mpu1.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G, 0x68))
  {
    Serial.println("Could not find a valid MPU6050 sensor1, check wiring!");
    delay(500);
  }
  mpu1.calibrateGyro();
  mpu1.setThreshold(3);

  // Initialize MPU6050 mpu2
  while(!mpu2.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G, 0x69))
  {
    Serial.println("Could not find a valid MPU6050 sensor1, check wiring!");
    delay(500);
  }
  mpu2.calibrateGyro();
  mpu2.setThreshold(3);

  Serial.println("start computing!");
}
```

硬體設定

```
void loop()
{
  timer = millis();

  Vector norm1 = mpu1.readNormalizeGyro();
  pitch1 = pitch1 + norm1.YAxis * timeStep;
  roll1 = roll1 + norm1.XAxis * timeStep;
  yaw1 = yaw1 + norm1.ZAxis * timeStep;

  Vector norm2 = mpu2.readNormalizeGyro();
  pitch2 = pitch2 + norm2.YAxis * timeStep;
  roll2 = roll2 + norm2.XAxis * timeStep;
  yaw2 = yaw2 + norm2.ZAxis * timeStep;

  Serial.print("(");
  Serial.print(pitch1);
  Serial.print(",");
  Serial.print(roll1);
  Serial.print(",");
  Serial.print(yaw1);
  Serial.print(")");
  Serial.print("(");
  Serial.print(pitch2);
  Serial.print(",");
  Serial.print(roll2);
  Serial.print(",");
  Serial.print(yaw2);
  Serial.println(")");

  // Wait to full timeStep period
  delay(1);
}
```

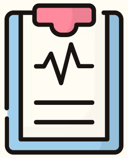
(pitch,roll,yaw) (pitch,roll,yaw)

Sensor1 Sensor2

```
void loop()
{
  timer = millis();

  Vector norm1 = mpu1.readNormalizeGyro();
  pitch1 = norm1.YAxis * timeStep;
  roll1 = norm1.XAxis * timeStep;
  yaw1 = norm1.ZAxis * timeStep;

  Vector norm2 = mpu2.readNormalizeGyro();
  pitch2 = norm2.YAxis * timeStep;
  roll2 = norm2.XAxis * timeStep;
  yaw2 = norm2.ZAxis * timeStep;

  Serial.print("(");
  Serial.print(pitch1,NUM);
  Serial.print(",");
  Serial.print(roll1,NUM);
  Serial.print(",");
  Serial.print(yaw1,NUM);
  Serial.print(")");
  Serial.print("(");
  Serial.print(pitch2,NUM);
  Serial.print(",");
  Serial.print(roll2,NUM);
  Serial.print(",");
  Serial.print(yaw2,NUM);
  Serial.println(")");

  // Wait to full timeStep period
  delay(1);
}
```

數據處理與分析

數據處理



Online

檢測同時顯示兩感測器相對姿態結果
後續規劃匯出csv數據
以利與超音波截圖做比對

處理程序

1

取得陀螺儀數據

pitch、roll、yaw
(radians)

處理非線性系統
估計系統狀態
denoising

擴展卡爾曼濾波

2

3

四元數

代表繞軸旋轉量
方便計算且避免萬向死鎖

旋轉矩陣

4

5

顯示影像

套用到三軸上的單位
向量以顯示影像

森

運算方式改變

感測器
讀取歐拉角度

感測器
疊加歐拉角

電腦
收data顯示圖形

原始

更新

感測器
讀取歐拉角度

電腦
收data

電腦
將歐拉角轉換成
四元數並疊加

電腦
輸出目前累積角度值

運算方式由原先的歐拉角運算，改成先轉換成四元數運算再使用歐拉角顯示，如此可以避免萬象死鎖

森

運算方式改變

```
pitch1 = pitch1 + norm1.YAxis * timeStep;//degree
roll1 = roll1 + norm1.XAxis * timeStep;
yaw1 = yaw1 + norm1.ZAxis * timeStep;
/*
pitch1 = norm1.YAxis * timeStep;//degree
roll1 = norm1.XAxis * timeStep;
yaw1 = norm1.ZAxis * timeStep;
*/

// Read normalized values 2
Vector norm2 = mpu2.readNormalizeGyro();

// Calculate Pitch, Roll and Yaw 2

pitch2 = pitch2 + norm2.YAxis * timeStep;//degree
roll2 = roll2 + norm2.XAxis * timeStep;
yaw2 = yaw2 + norm2.ZAxis * timeStep;
```

arduino code

```
pitch1 = norm1.YAxis * timeStep;//degree
roll1 = norm1.XAxis * timeStep;
yaw1 = norm1.ZAxis * timeStep;


// Read normalized values 2
Vector norm2 = mpu2.readNormalizeGyro();

// Calculate Pitch, Roll and Yaw 2
/*
pitch2 = pitch2 + norm2.YAxis * timeStep;//degree
roll2 = roll2 + norm2.XAxis * timeStep;
yaw2 = yaw2 + norm2.ZAxis * timeStep;
*/

pitch2 = norm2.YAxis * timeStep;//degree
roll2 = norm2.XAxis * timeStep;
yaw2 = norm2.ZAxis * timeStep;
```

arduino code

```
pitch ,roll ,yaw = pitch2, roll2, yaw2
#print(pitch, roll, yaw)

handler.update_with_new_euler(roll, pitch, yaw)


# 印出累計的歐拉角值
sum_roll, sum_pitch, sum_yaw = handler.get_current_euler()
sum_roll, sum_pitch, sum_yaw = np.degrees(sum_roll),np.degrees(sum_pitch),np.degrees(sum_yaw)
print("sum:",sum_roll, sum_pitch, sum_yaw)
```

主程式內迴圈執行

```
def update_with_new_euler(self, roll, pitch, yaw):
    new_quaternion = self.euler_to_quaternion(roll, pitch, yaw)
    self.accumulated_quaternion = self.quaternion_multiply(new_quaternion, self.accumulated_quaterni
    self.normalize_quaternion()
    #print(self.accumulated_quaternion)
```

呼叫的函式

森

四元數運算

```python
def euler_to_quaternion(self, roll, pitch, yaw):
    roll = np.radians(roll)
    pitch = np.radians(pitch)
    yaw = np.radians(yaw)
    cr = np.cos(roll * 0.5)
    sr = np.sin(roll * 0.5)
    cp = np.cos(pitch * 0.5)
    sp = np.sin(pitch * 0.5)
    cy = np.cos(yaw * 0.5)
    sy = np.sin(yaw * 0.5)

    w = cy * cp * cr + sy * sp * sr
    x = cy * cp * sr - sy * sp * cr
    y = sy * cp * sr + cy * sp * cr
    z = sy * cp * cr - cy * sp * sr

    return np.array([w, x, y, z])
```

歐拉角轉成四元數

```python
def quaternion_multiply(self, q1, q2):
    w1, x1, y1, z1 = q1
    w2, x2, y2, z2 = q2

    w = w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2
    x = w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2
    y = w1 * y2 - x1 * z2 + y1 * w2 + z1 * x2
    z = w1 * z2 + x1 * y2 - y1 * x2 + z1 * w2

    return np.array([w, x, y, z])
```

累積角度

```python
def normalize_quaternion(self):
    norm = np.linalg.norm(self.accumulated_quaternion)
    if norm == 0:
        return
    self.accumulated_quaternion / norm
```

四元數正規化

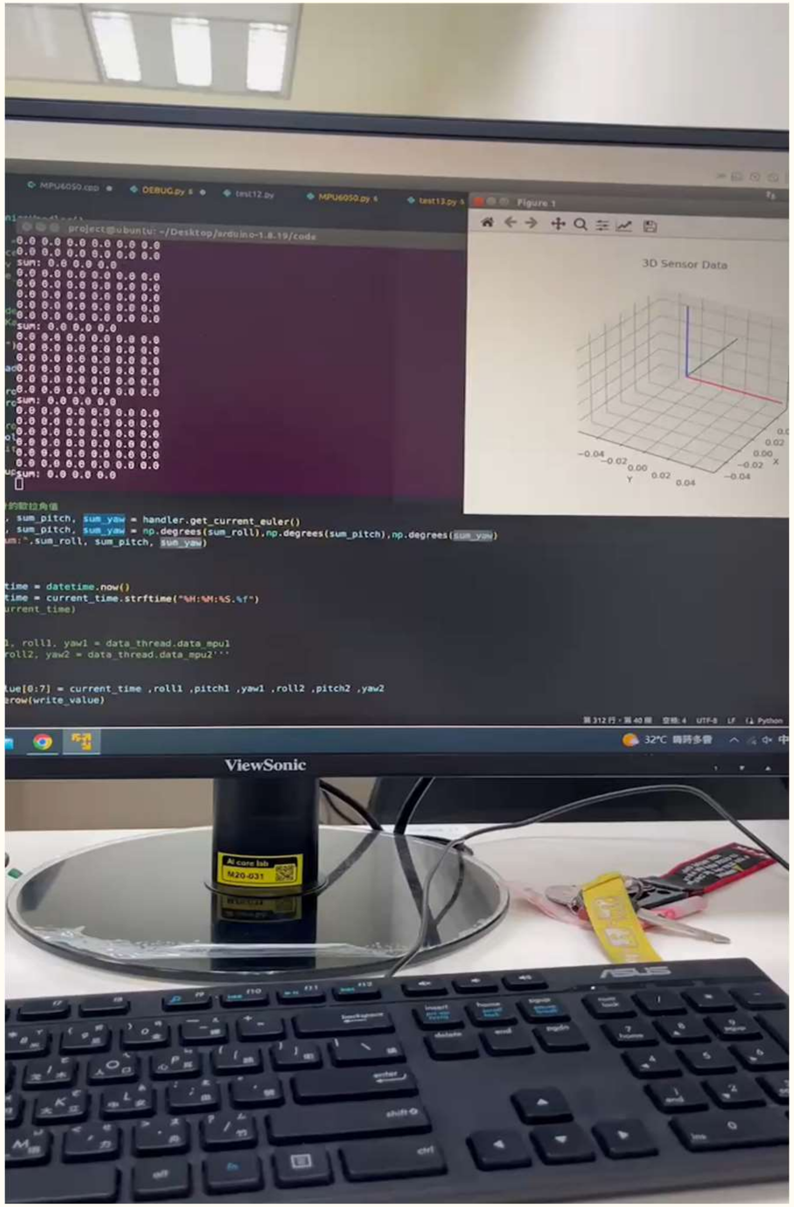
```python
def get_current_euler(self):
    return self.quaternion_to_euler(self.accumulated_quaternion)
```

取得目前累積值

```python
def quaternion_to_euler(self, q):
    w, x, y, z = q

    sinr_cosp = 2 * (w * x + y * z)
    cosr_cosp = 1 - 2 * (x * x + y * y)
    roll = np.arctan2(sinr_cosp, cosr_cosp)

    sinp = 2 * (w * y - z * x)
    if abs(sinp) >= 1:
        pitch = np.sign(sinp) * np.pi / 2  # use 90 degrees if out of range
    else:
        pitch = np.arcsin(sinp)

    siny_cosp = 2 * (w * z + x * y)
    cosy_cosp = 1 - 2 * (y * y + z * z)
    yaw = np.arctan2(siny_cosp, cosy_cosp)

    return roll, pitch, yaw
```

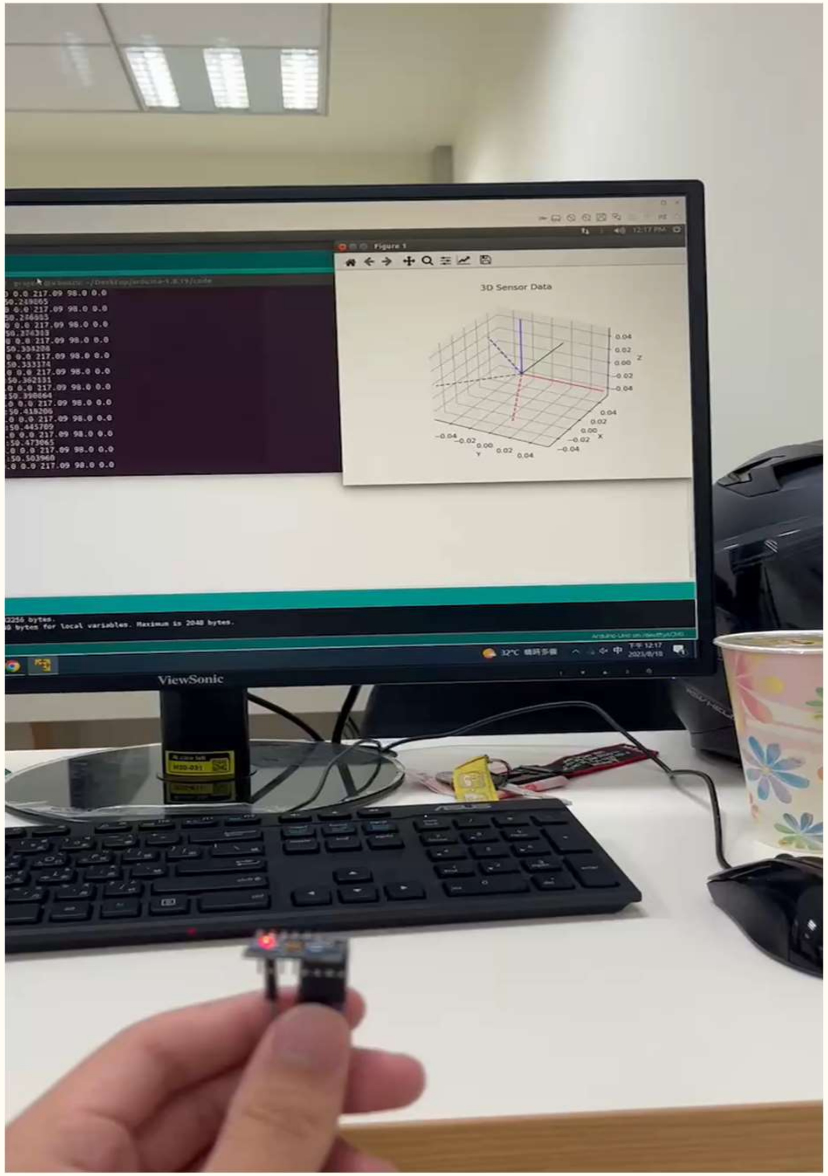
將累積值的四元數轉回歐拉角

森



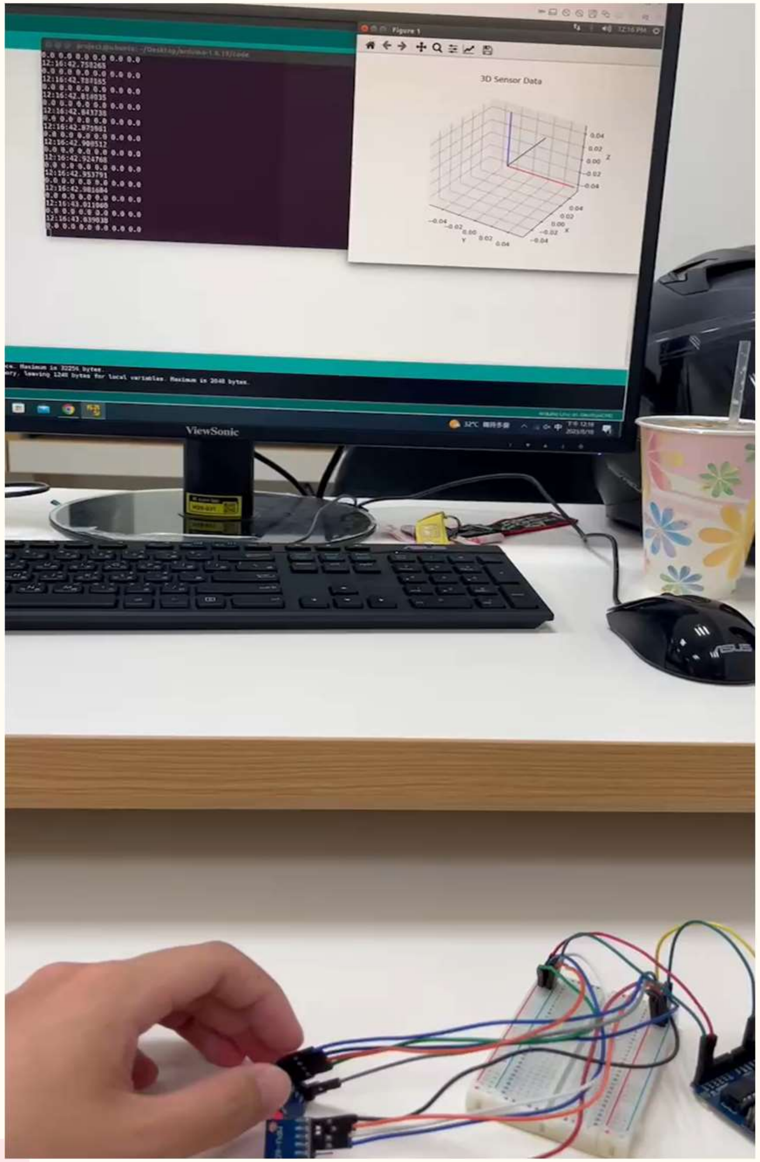
使用環境:
使用四元數累積角度
問題點:
運算出來的累積角度比實際角度還要小

森



使用環境:
使用感測器內累積的歐拉角直接輸出
問題點:
運算出來的累積角度比實際角度還要大

森



使用環境:
使用感測器內累積的歐拉角直接輸出
問題點:
輸出值異常，有在讀取但不是很流暢

庭

四元數&EKF

```
def quat_and_kalman(ekf, pitch, roll, yaw):
    # Quanternion
    quaternion = euler_to_quat(roll, pitch, yaw)
    # EKF
    ekf.predict()
    ekf.update(quaternion)
    R_est = R.from_quat(ekf.state).as_matrix()
    return R_est
```

感測器
讀取歐拉角度

電腦
收data

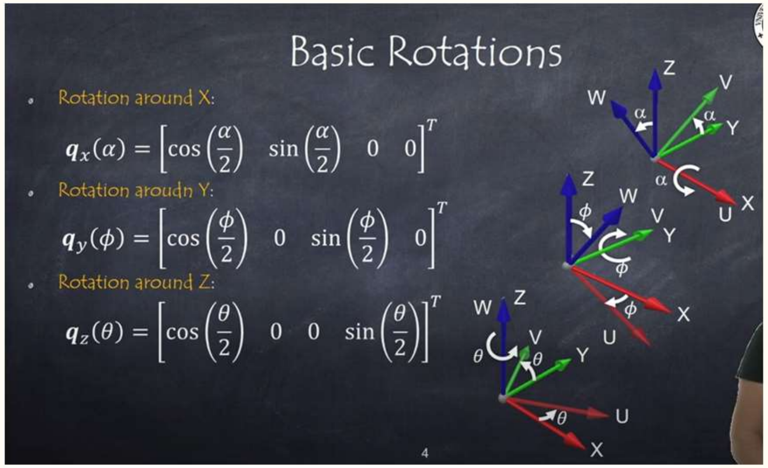
電腦
將歐拉角轉四元數

電腦
套用EKF
並輸出旋轉矩陣

庭

四元數運算

```python
def euler_to_quat(roll, pitch, yaw):
    cy = np.cos(yaw * 0.5)
    sy = np.sin(yaw * 0.5)
    cr = np.cos(roll * 0.5)
    sr = np.sin(roll * 0.5)
    cp = np.cos(pitch * 0.5)
    sp = np.sin(pitch * 0.5)

    w = cy * cr * cp + sy * sr * sp
    x = cy * sr * cp - sy * cr * sp
    y = cy * cr * sp + sy * sr * cp
    z = sy * cr * cp - cy * sr * sp

    return np.array([w, x, y, z])
```



- X 軸旋轉量：ϕ
- Y 軸旋轉量：θ
- Z 軸旋轉量：ψ

$$q_\phi = \cos\frac{\phi}{2} + i\sin\frac{\phi}{2}$$

$$q_\theta = \cos\frac{\theta}{2} + j\sin\frac{\theta}{2}$$

$$q_\psi = \cos\frac{\psi}{2} + k\sin\frac{\psi}{2}$$

$$\rightarrow \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \end{pmatrix}$$

庭

EKF運算

```
initial_state = np.array([1.0, 0.0, 0.0, 0.0])  # 初始四元數狀態
initial_covariance = np.eye(4)  # 初始協方差矩陣
process_noise_cov = np.eye(4) * 0.01  # 過程噪音協方差矩陣
measurement_noise_cov = np.eye(4) * 0.1  # 測量噪音協方差矩陣
```

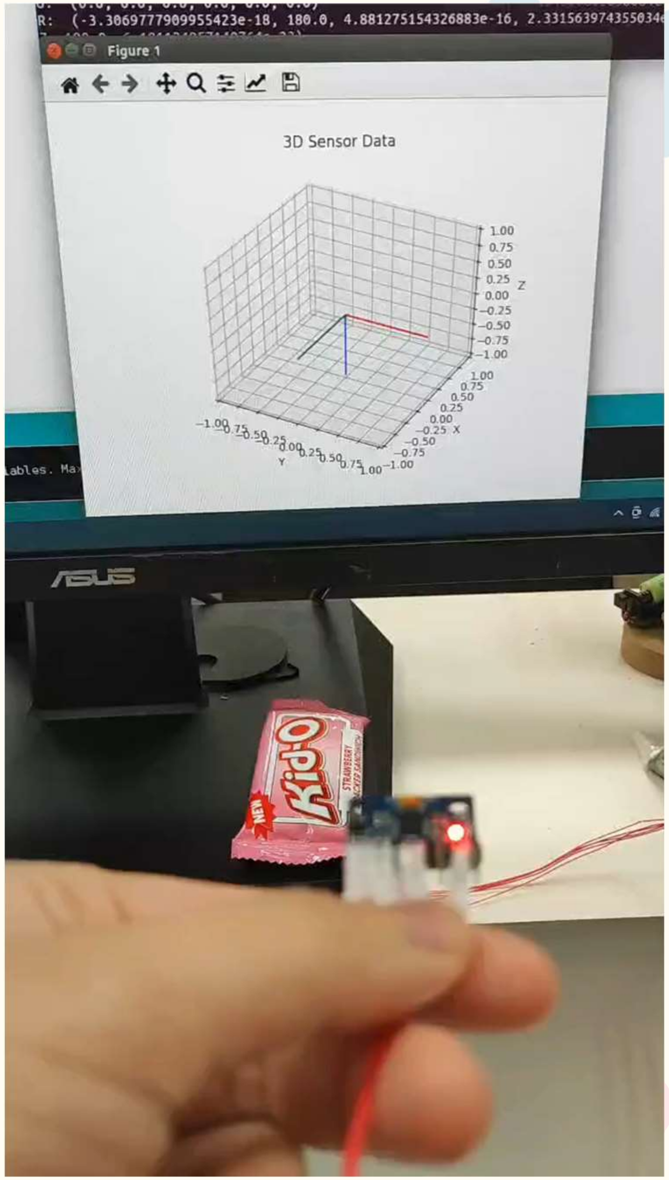
```
class ExtendedKalmanFilter:
    def __init__(self, initial_state, initial_covariance, process_noise_cov, measurement_noise_cov):
        self.state = initial_state
        self.covariance = initial_covariance
        self.process_noise_cov = process_noise_cov
        self.measurement_noise_cov = measurement_noise_cov

    def predict(self):# 預測步驟
        F = np.eye(4)  # 狀態轉移矩陣
        Q = self.process_noise_cov
        self.state = np.dot(F, self.state)
        self.covariance = np.dot(np.dot(F, self.covariance), F.T) + Q

    def update(self, measurement):# 更新步驟
        H = np.eye(4)  # 測量模型
        R = self.measurement_noise_cov
        y = measurement - np.dot(H, self.state)
        S = np.dot(np.dot(H, self.covariance), H.T) + R
        K = np.dot(np.dot(self.covariance, H.T), np.linalg.inv(S))
        self.state = self.state + np.dot(K, y)
        self.covariance = np.dot(np.eye(4) - np.dot(K, H), self.covariance)
```

庭

旋轉矩陣

```python
def matrix_to_euler(matrix):
    roll = np.arctan2(matrix[2, 1], matrix[2, 2])
    pitch = np.arctan2(-matrix[2, 0], np.sqrt(matrix[2, 1]**2 + matrix[2, 2]**2))
    yaw = np.arctan2(matrix[1, 0], matrix[0, 0])
    return np.array([pitch, roll, yaw])
```

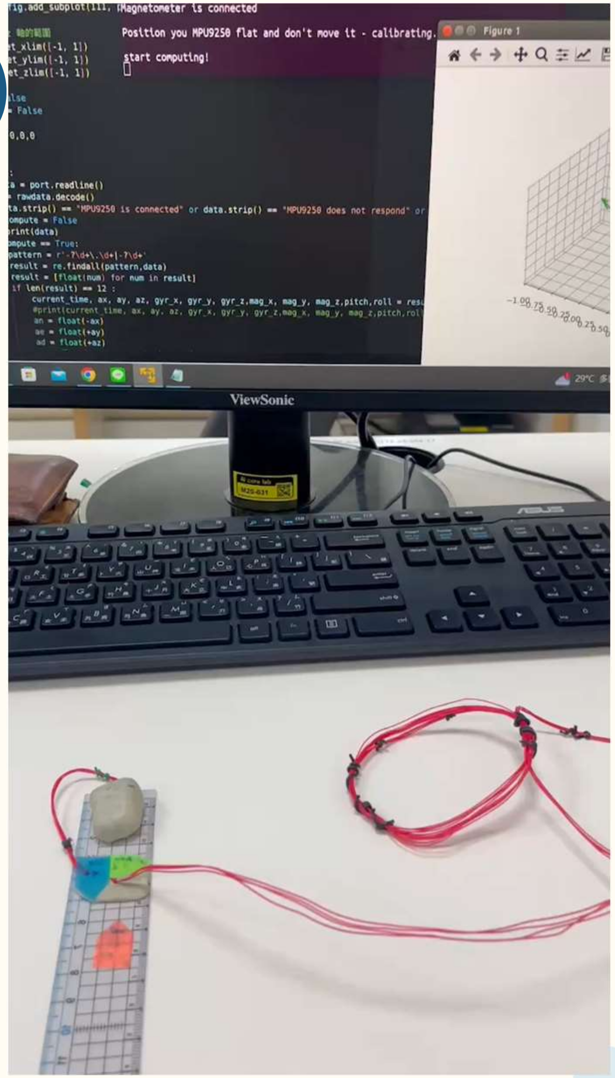
$$\begin{pmatrix}\phi\\ \theta\\ \psi\end{pmatrix}=\begin{pmatrix}\tan^{-1}(\frac{q32}{q33})\\ \sin^{-1}(-q_{31})\\ \tan^{-1}(\frac{q21}{q11})\end{pmatrix}=\begin{pmatrix}\tan^{-1}(\frac{2(yz+wx)}{1-2(x^2+y^2)})\\ \sin^{-1}(2(xz-wy))\\ \tan^{-1}(\frac{2(xy+wz)}{1-2(y^2+z^2)})\end{pmatrix}$$



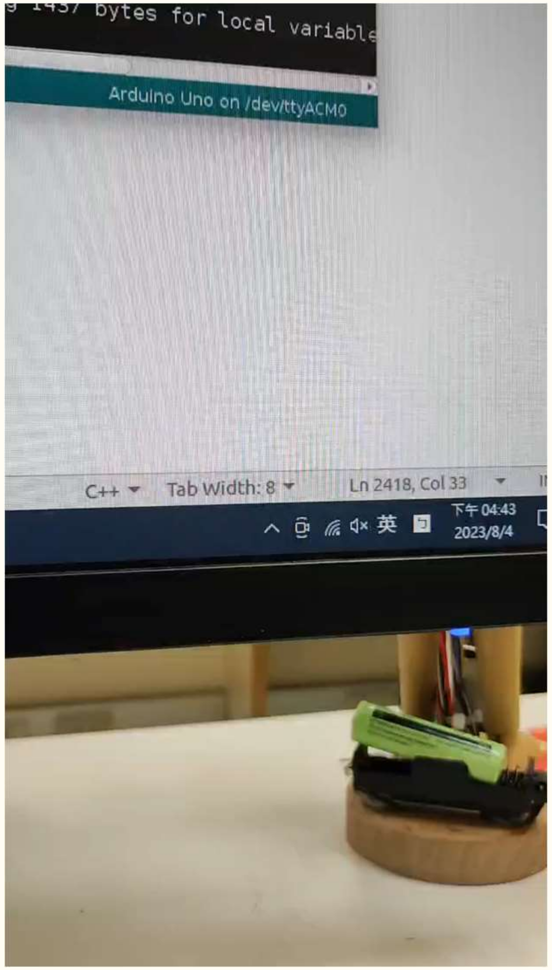
姿態估計

單一感測器(MPU9250)

```python
roll = rpy[0] ##radian
pitch = rpy[1]
yaw = rpy[2]

last_time = current_time

ax_subplot.cla()  # 清除目前的 figure

# 設定 x, y, z 軸的範圍
ax_subplot.set_xlim([-1, 1])
ax_subplot.set_ylim([-1, 1])
ax_subplot.set_zlim([-1, 1])

xyz = pitch,roll,yaw

# 建立一個旋轉矩陣
r = R.from_euler('xyz', xyz, degrees=True)

# 將旋轉矩陣應用到每個箭頭向量上
rotated_vectors = r.apply(vectors)

# 畫出旋轉後的箭頭
for v, c in zip(rotated_vectors, colors):
    ax_subplot.quiver(0, 0, 0, v[0], v[1], v[2], color=c)

plt.draw()
```



單一感測器(MPU9250)



雙感測器(MPU6050)