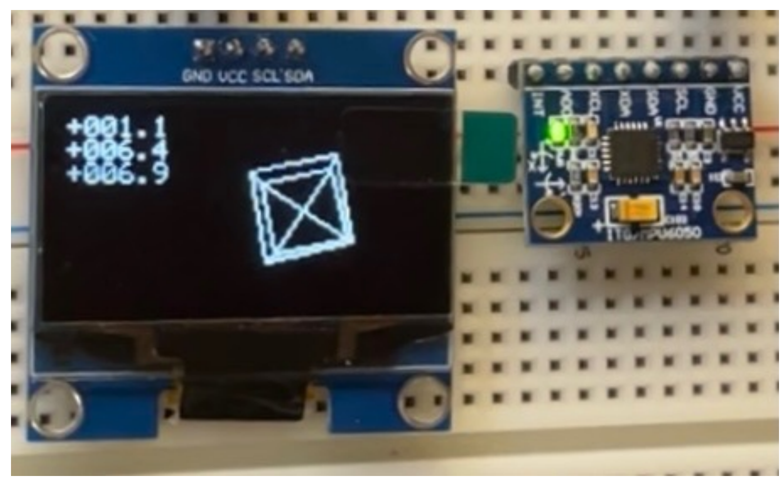
Microprocessor Systems

Final Project - Gyroscope Cube 3D Simulation



108061151 EE23 陳均豪

Motivation

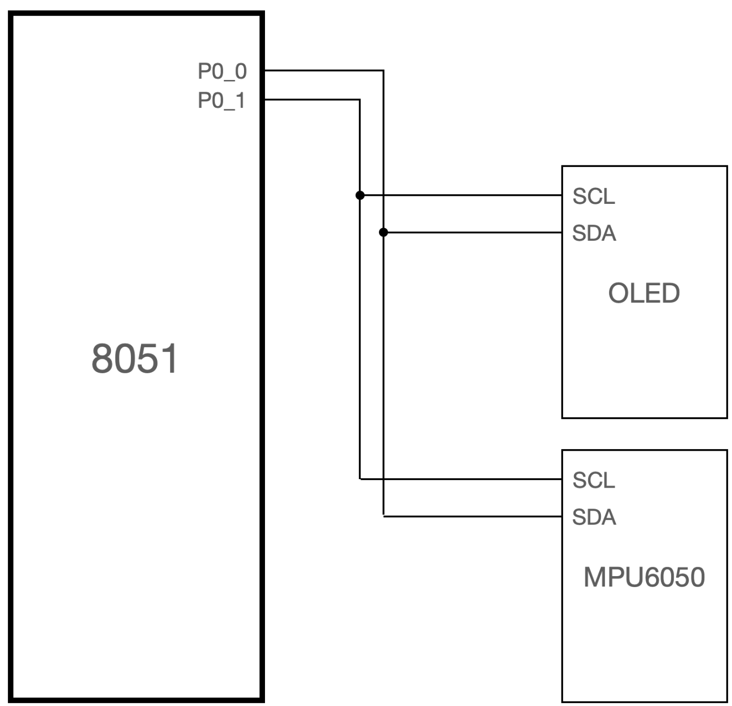
- There are many device with electrical gyroscope built in, they have lots of applications such as electronics level meter, camera stabilizer.
- I want to know how difficult it is to build a highly accuracy electrical gyroscope.
- Try to draw some complex graphic through the OLED display.
- Combine them to build a electrical gyroscope with OLED display to simulate a 3D cube rotate in the space corresponding to the gyroscope.

Feature

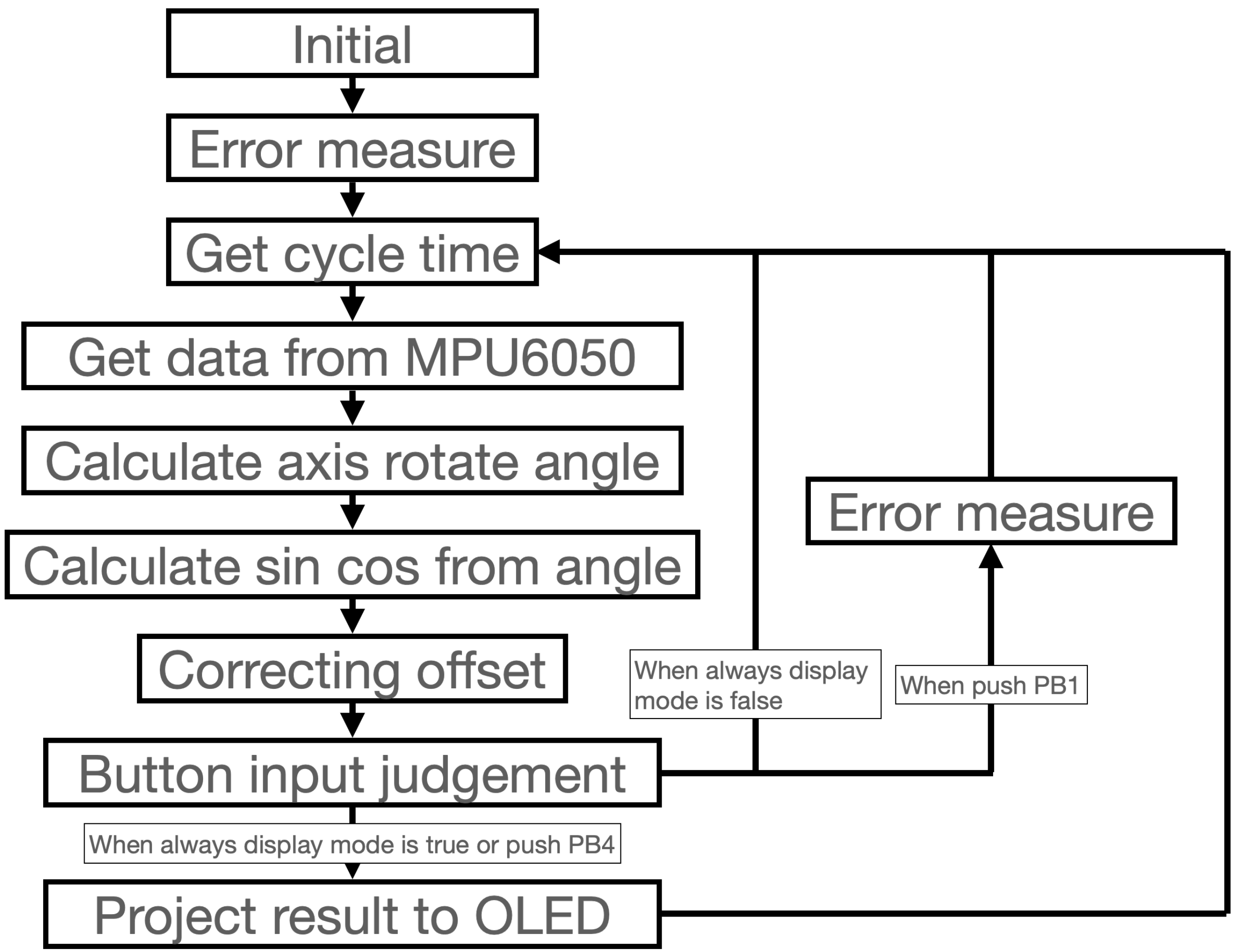
- Angle sensitive is up to 0.1 degree
- Support visual horizontal plane
- Support 3D cube simulation
- Support real time display (will loss some accuracy)
- In real time display mode, the fps is about 1

Wiring diagram and table

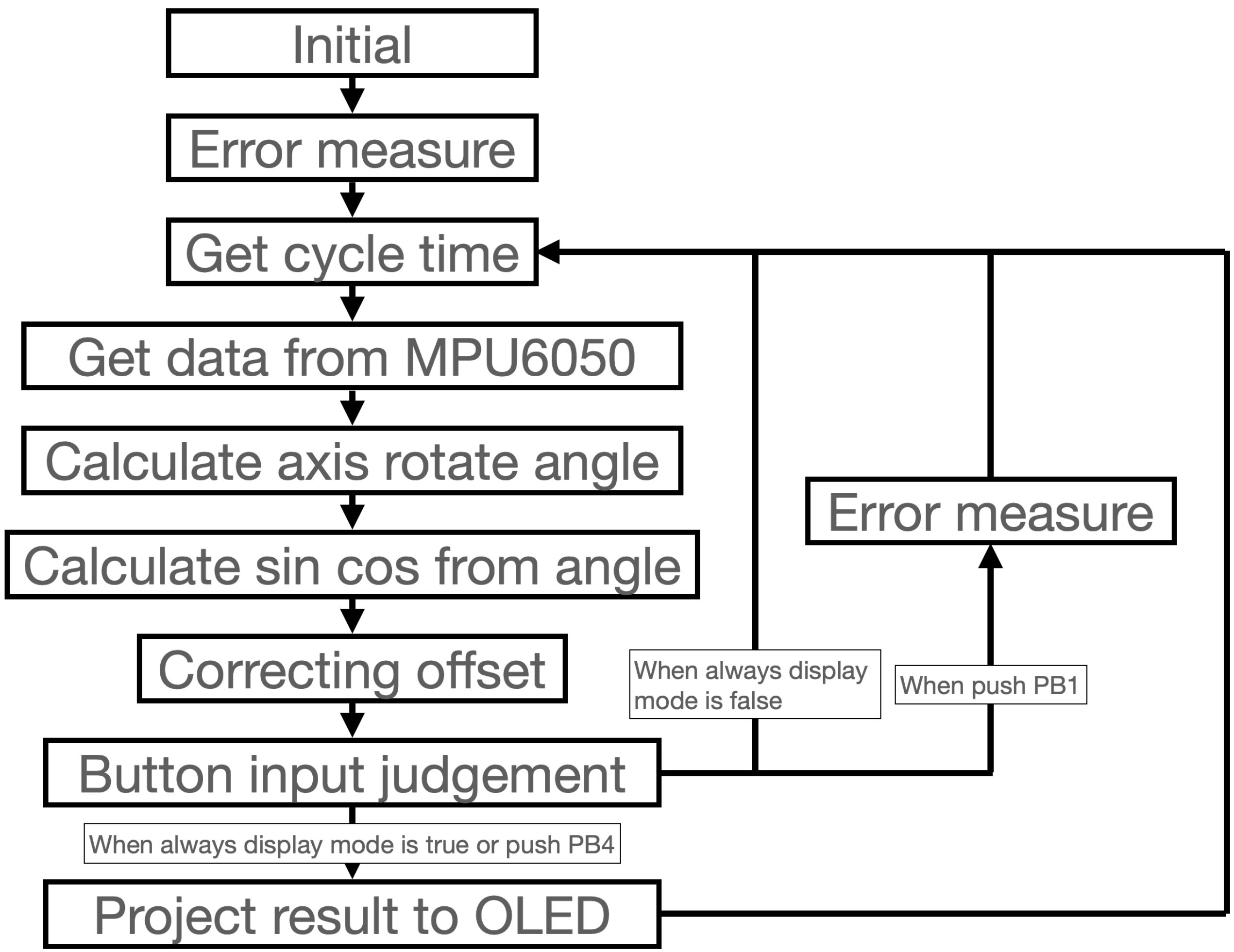
- P0_0: MPU6050's SCL and OLED's SCL
- P0_1: MPU6050's SDA and OLED's SDA



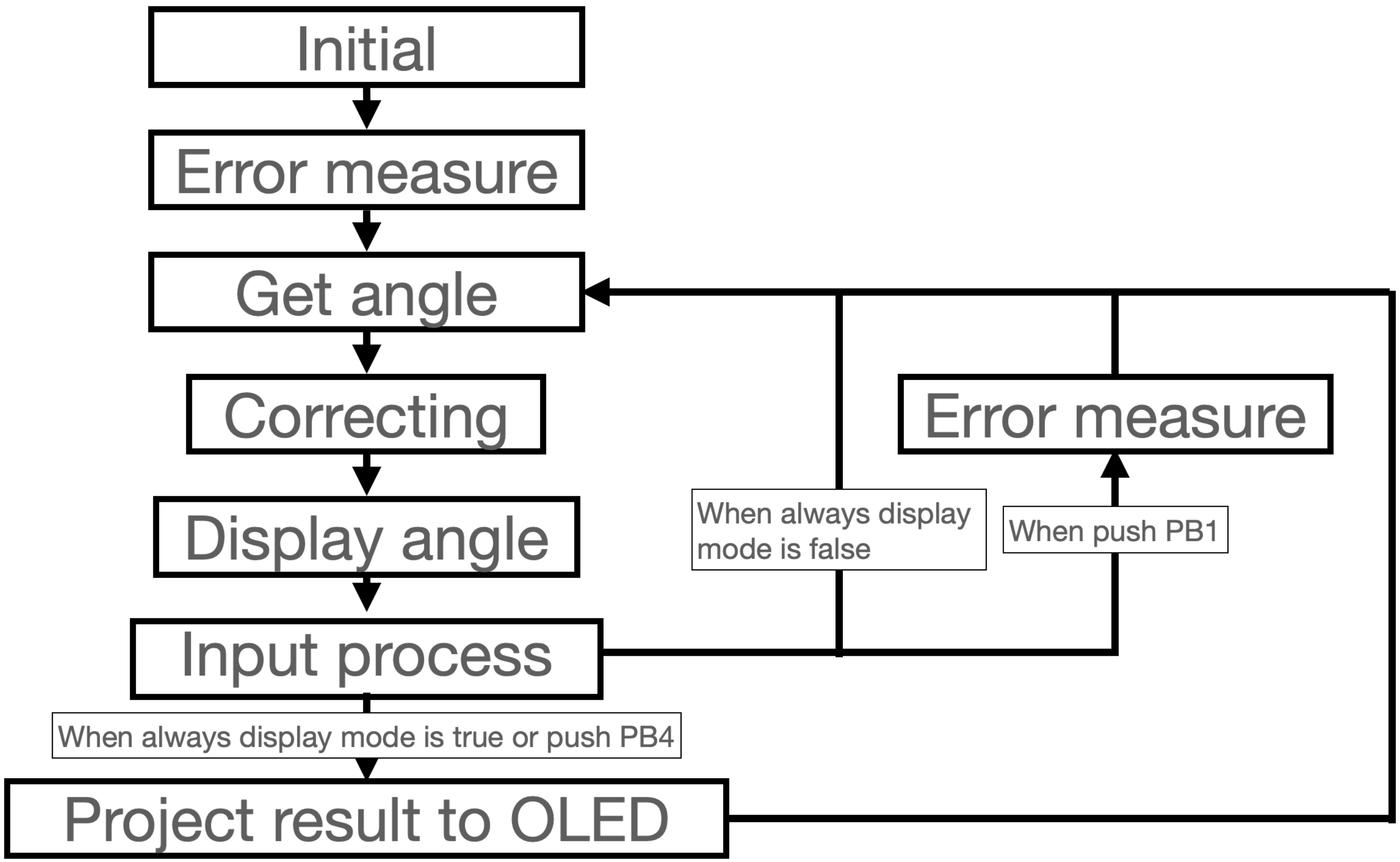
Software flow chart



Software flow chart



Software flow chart – easier



Main - source

- There are seven part in main function. (1) Initial (2) error measure (3) get angle (4) correcting (5) display angle (6) input process (7) project

```c
int main() {
    int elapsedTime;
    char temp;
    int angle;
    int angle_rotate[3] = {0};
    char always_display = 0;
    char key_down = 0;

    // initial
    // ...

    // error measure
    // ...

    while(1) {
        // get angle
        // ...

        // correcting
        // ...

        // display angle
        // ...

        // input process
        // ...

        // project
        // ...
    }
}
```

Function I made

- void get_angle(void);
- void error_measure(void);
- void project(void);
- int square_root(int num);
- char absolute(char num);
- int sin_approximate(int angle);
- void angle_sum(char a, char b);

Initial - source

```
// initial
TMOD = 0x01;                         // set Timer1 mode0 & Timer0 mode1
TH0 = (65536 - 922) / 256;           // set Timer0 for 1ms per cycle
TL0 = (65536 - 922) % 256;
ET0 = 1;                             // Enable Timer0 interrupt
EA = 1;                              // Enable all interrupt
TR0 = 1;                             // Enable Timer0

SDA = 1;
SCL = 1;
OLED_Init();                         // OLED initial
MPU6050_INIT();                      // MPU6050 initial
```

Error measure

- There are little offset of gyroscope, we need to measure it to eliminate the error.
- The calibration plane maybe not horizontal, we need to know how much angle it tilts.

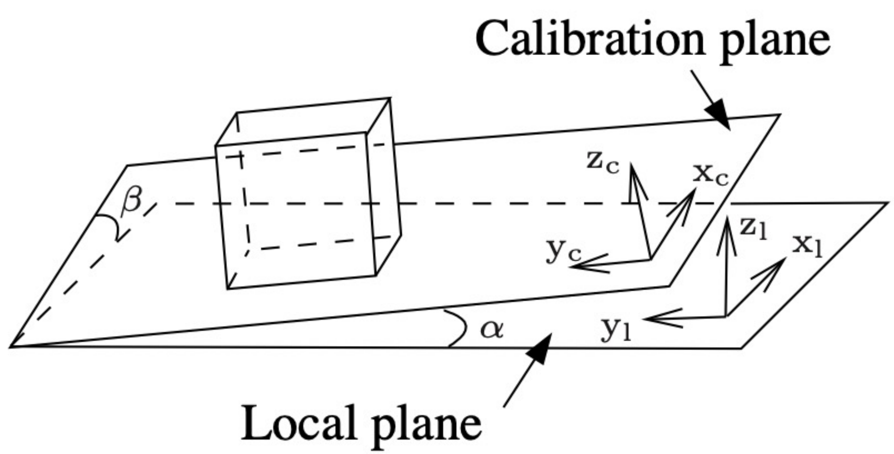


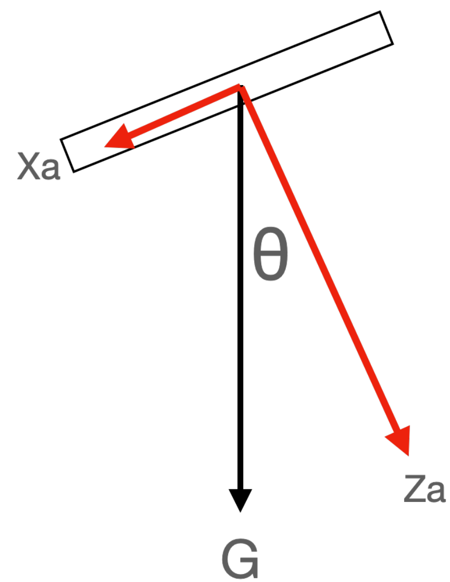
Figure from: https://tutcris.tut.fi/portal/files/11404303/FINAL_VERSION.pdf

Get angle

- The angle of rotation is very crucial to whole project, and calculate the angle in degree need the function such as $\tan^{-1} x$, when I try to find the approximate function of $\tan^{-1} x$, I find it is too complex to 8051, so I turn to calculate the $\sin \theta$ and $\cos \theta$, which is more easily implement.
- Since the range of $\sin \theta$ and $\cos \theta$ is $[-1, 1]$, it is hard to implement on 8051 (since 8051 has no FPU), I multiply it by 64 to make the range to be $[-64, 64]$ for convenient integer calculating.

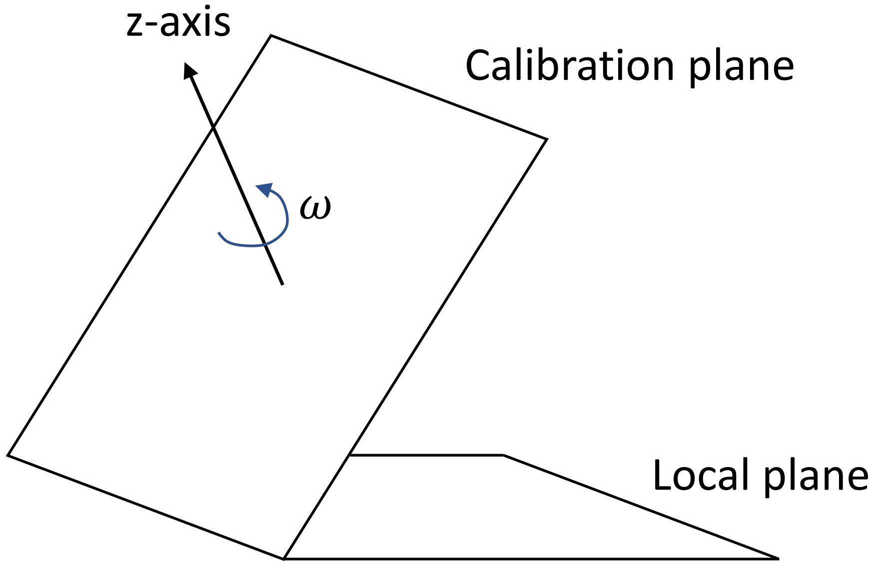
Get angle

- We use the accelerometer to measure how much angle it tilts. From bellow figure, we can calculate cos(θ) and sin(θ) by measure the accelerate of x-axis and z-axis and which means the rotation of y-axis.
- X-axis is similarly.



Get angle

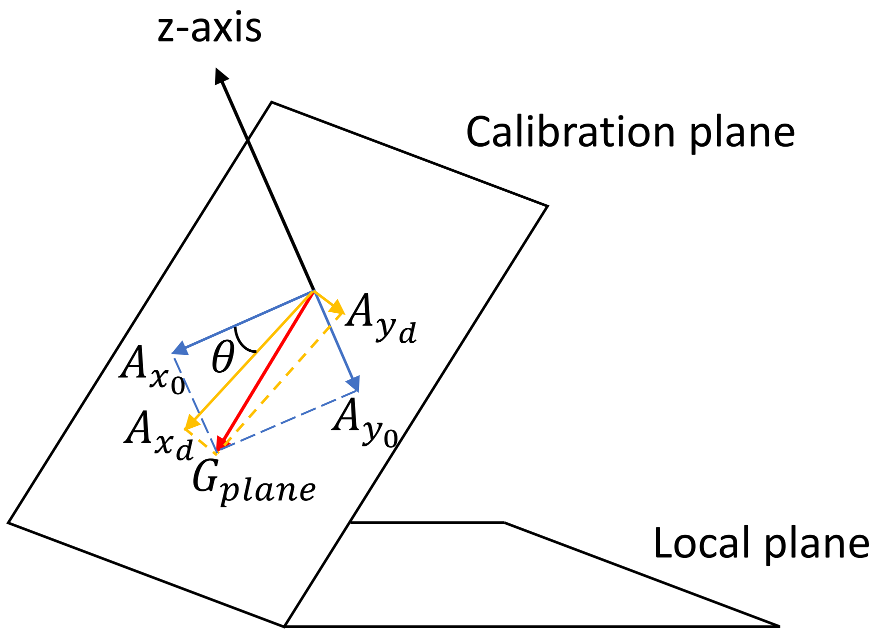
- We cannot use accelerator to calculate the angle of z-axis rotate, we turn to integral the z-axis's angular velocity by time to get the angle of z-axis rotate.
- We do the same thing on x-axis and y-axis, in order to replace $\tan^{-1} x$ and show the result on OLED



$$\theta_z = \int_{t_1}^{t_2} \omega dt$$

Get angle

- When the calibration plane is not horizontal, or we say we crate a visual horizontal plane, the rotate of z-axis will influence the accelerator of x-axis and y-axis, we need to correct it.



A_{x_0}: The accelerator of x-axis where no rotate z-axis

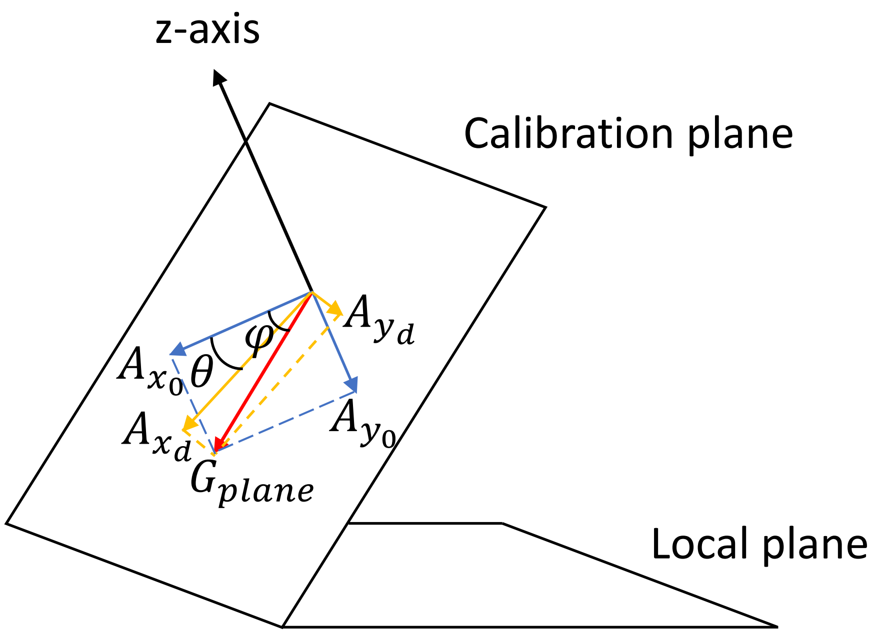
A_{y_0}: The accelerator of y-axis where no rotate z-axis

A_{x_d}: The accelerator of x-axis where rotate z-axis

A_{y_c}: The accelerator of y-axis where rotate z-axis

θ : The rotated angle of z-axis

Get angle



$$\begin{cases} A_{x_0} = G_{plane} \times \cos(\varphi) \\ A_{y_0} = G_{plane} \times \sin(\varphi) \end{cases}$$

$$\begin{cases} A_{x_d} = G_{plane} \times \cos(\varphi + \theta) \\ A_{y_d} = G_{plane} \times \sin(\varphi + \theta) \end{cases}$$

From two relation above and we know the θ, we can conclude the result

$$\begin{cases} A_{x_0} = A_{x_d}\cos(\theta) + A_{y_d}\sin(\theta) \\ A_{y_0} = -A_{x_d}\sin(\theta) + A_{y_d}\cos(\theta) \end{cases}$$

Sine approximate

- The sine approximate formula is found on https://en.wikipedia.org/wiki/Bhaskara_I%27s_sine_approximation_formula.
- The formula is $\sin x \approx \frac{4x(180-x)}{40500-x(180-x)}, x\ in\ degree$
- I want the range of $\sin x$ to be $[-64, 64]$, so I modified it to $\sin x \approx \frac{2x(180-x)}{\frac{(40500-x(180-x))}{128}}, x\ in\ degree$

Sine approximate – source code

- The precedence of calculation is import since it may out of range of integer.

```
int sin_approximate(int angle) {
    return (2 * angle * (180 - angle)) / ((40500 - angle * (180 - angle)) / 128);
}
```

Correcting

- The calibration plane maybe not horizontal, we need to correct them by the data that collect by error measure.
- The method is just do some easy angle sum.
- The correction include the x-axis and y-axis angle display, since the value of angle display is come from the integral of angular velocity, the error will expand through the time it works, so I use the accelerator the correct them.

Correcting – source code

```
// correcting
angle_sum(0, 0);
angle_sum(2, 2);

if (array[2] == 0 && (angle_rotate[0] > 100 || angle_rotate[0] < -100)) {
    angle_rotate[0] = 0;
}

if (array[0] == 0 && (angle_rotate[1] > 100 || angle_rotate[1] < -100)) {
    angle_rotate[0] = 0;
}
```

Angle sum - source code

- In the visual horizontal correction, I need to do some angle sum in sine and cosine.
- The formula

$$\begin{cases} \sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi \\ \cos(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi \end{cases}$$

```c
void angle_sum(char a, char b) {
    char temp;

    temp = array[a];
    array[a] = (array[a] * accel_error[b + 1] - array[a + 1] * accel_error[b]) >> 6;
    array[a + 1] = (array[a + 1] * accel_error[b + 1] + temp * accel_error[b]) >> 6;
}
```

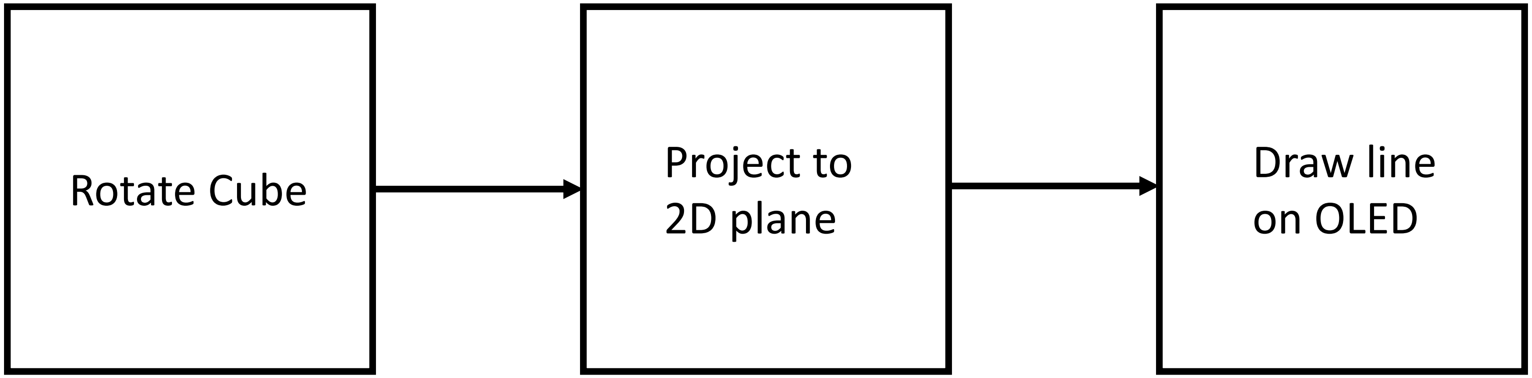
Input process – source code

- Process the button input to decide how the program function.

```
// input process
if (!P3_2) {                                                    // create visual horizontal plane
    error_measure();
    angle_rotate[0] = 0;
    angle_rotate[1] = 0;
    angle_rotate[2] = 0;
}
if (!P2_0 && key_down == 0 && always_display == 1) {     // switch off real time display
    always_display = 0;
    key_down = 1;
}
if (!P2_0 && key_down == 0 && always_display == 0) {     // switch on real time display
    always_display = 1;
    key_down = 1;
}
if (P2_0) {
    key_down = 0;
}
```

Project

- The final step is projecting the cube on OLED with corresponding rotation.
- There are three step in projecting. (1)Rotate the 3D cube. (2)Project it to 2D plane. (3)Use algorithm to draw line on OLED.

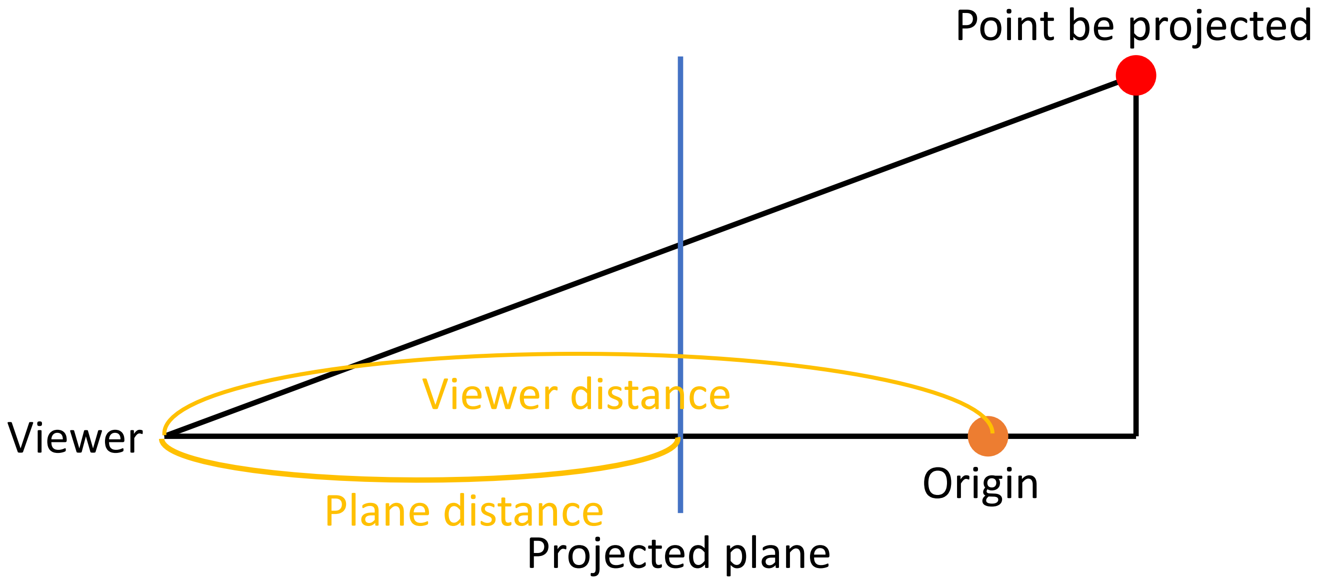


Project

- First step – Rotate the 3D cube.
- Using three rotate matrix to rotate the cube.
- $R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$
- $R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$
- $R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Project

- Second step – Project 3D point to 2D plane
- We assume that the point of view in above the origin with some distance.
- $(x_p,\ y_p) = \dfrac{plane\ distance}{viewer\ distance + z\ coordinate\ of\ point}(x,\ y)$

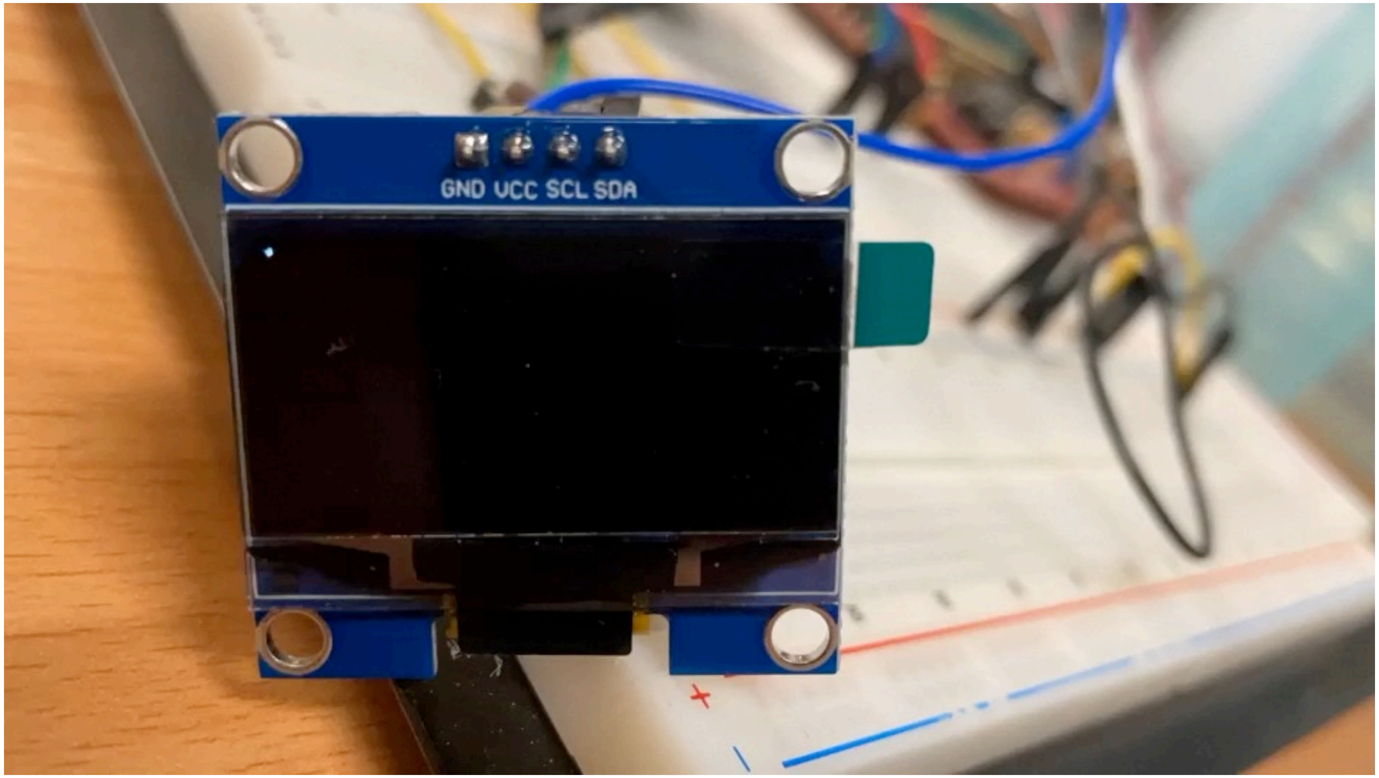


Project

- Third step – Draw line on OLED
- We need to know how to draw a line between two point, I try two algorithm, (1)Calculating the vector between two point, slicing the vector to get each point between them. (2) Bresenham's algorithm.
- First method is not working well because I can not calculate the point accurately since it may not be integer. So I turn to Bresenham's algorithm.
- The Bresenham's algorithm is introduced on https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

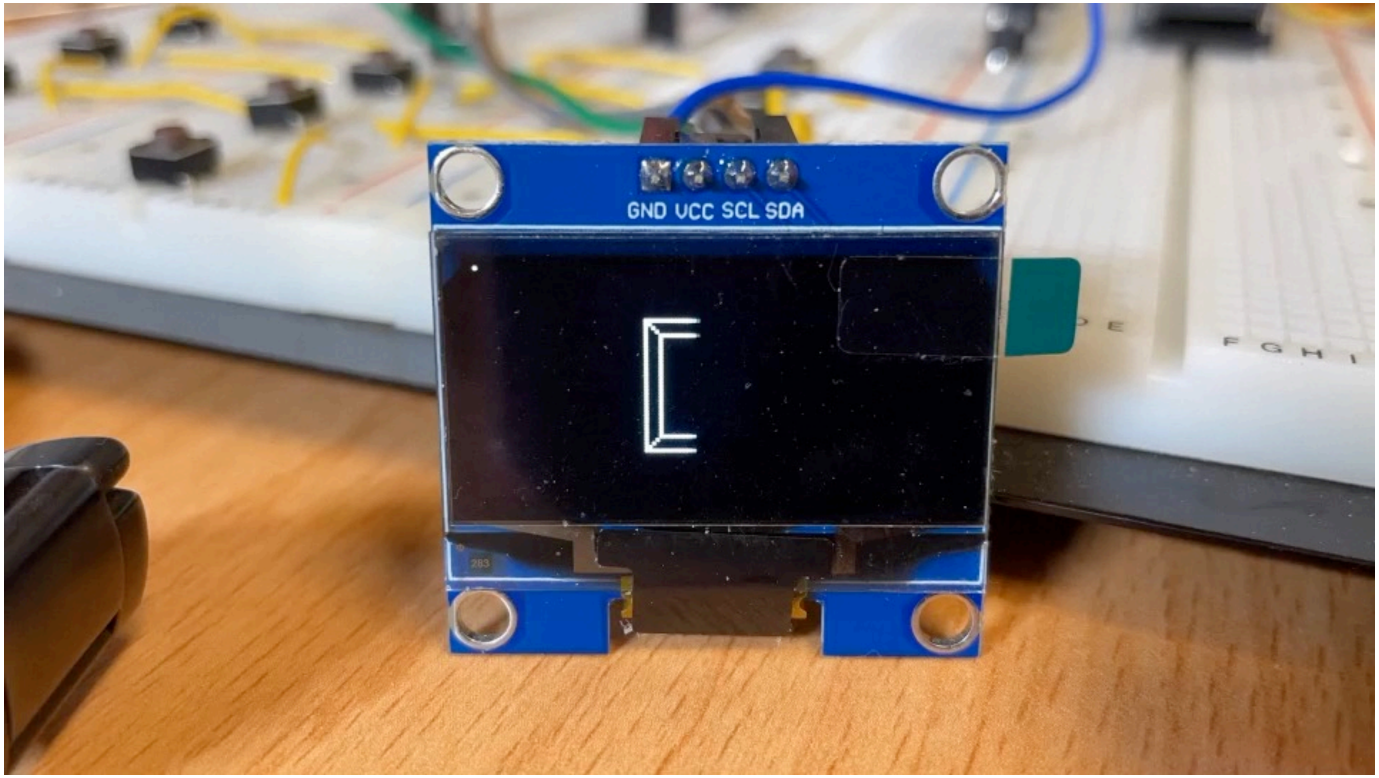
Project

- The vector slicing method on a spinning cube.



Project

- The Bresenham's algorithm on a spinning cube.



Project

- After three steps, it still have some problems, the RAM of 8051(8052) is not big enough to store all points we want to show on the OLED.
- To solve this problem, I separate the row to four parts, each part will run all points and edges, so it takes time but I have no ideas.
- To determined the edge, I use a matrix to store the relation between edges and points.

Project – source code

```
// project
if (!P2_1 || always_display) {
    project();
}
```

Project code in main function

```
__code const char Point[8][3] = {        // 8 point
    {-16, -16,  16},    // 0
    {-16,  16,  16},    // 1
    { 16, -16,  16},    // 2
    { 16,  16,  16},    // 3
    {-16, -16, -16},    // 4
    {-16,  16, -16},    // 5
    { 16, -16, -16},    // 6
    { 16,  16, -16}     // 7
};

__code const char edge[14][2] = {        // relation of edges and points
    {0, 1}, {1, 3}, {3, 2}, {2, 0},
    {4, 5}, {5, 7}, {7, 6}, {6, 4},
    {0, 4}, {1, 5}, {2, 6}, {3, 7},
    {1, 2}, {0, 3}
};
```

The coordinate of cube and relation of edges and points

Project – source code

```
void project(void) {
    unsigned char row_counter;
    unsigned char edge_counter;
    unsigned char slice;
    char project_x, project_y;
    int factor;
    char project_flag;
    char temp;
    char deltax;
    char deltay;
    char error;
    char x;
    char y;

    char Rotated_Point[2][3];

    char print_buffer[100] = {0};

    // run four part of OLED
    for (row_counter = 0; row_counter < 8; row_counter += 2) {
        // run all edge
        for (edge_counter = 0; edge_counter < 14; edge_counter++) {

            // Rotate 3D cube
            // ...

            // Project to 2D plane
            // ...

            // Bresenham's algorithm
            // ...
        }
        // OLED output
        // ...
    }
}
```

Square root / absolute - source code

- I need a square root the calculate vector length and convert sine into cosine, which has integer accuracy.
- In Bresenham's algorithm need to compare two numbers’ absolute.

```c
int square_root(int num) {
    int N = 1;

    while (N * N <= num) {
        N++;
    }
    return N - 1;
}
```

```c
char absolute(char num) {
    if (num < 0)
        return -num;
    return num;
}
```

The code from class

- 8051.h
- Lab3_OLED.h with some modified
- Lab4_MPU6050.h with some modified
- i2c.h with some modified
- stdutils.h

Demo video