ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Модель системи трекінгу кінцівок людини за допомогою сенсорів

Кваліфікаційна робота

Другий (магістерський) рівень

Автор:
Дашков Д.Є.
ст. гр. СПм-21-2

Керівник:
Ляшенко О.С
доц. каф. ЕОМ

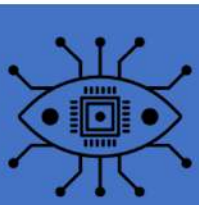
Актуальність

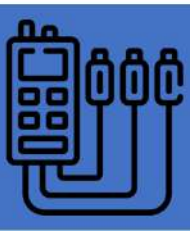
Захват руху кінцівок людини може надати цінну інформацію про продуктивність і поведінку людини. Таким чином покращити наше розуміння того, як рухається людське тіло, захоплення руху може призвести до прогресу в спортивній науці, медицині та віртуальній реальності, а також покращити способи взаємодії люди з різними електронними пристроями.

Методи захвату кінцівок людини

Існують різні підходи до фіксації руху людських кінцівок:

- комп'ютерного зору (CV)
- інерційні вимірювальні датчики (IMU)
- електроміографія (EMG)



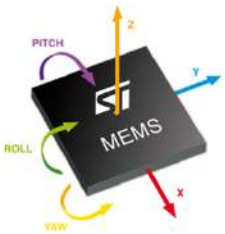


Мета та постанова задачі

Метою роботи є створення моделі для фіксації руху кінцівок, яка буде:

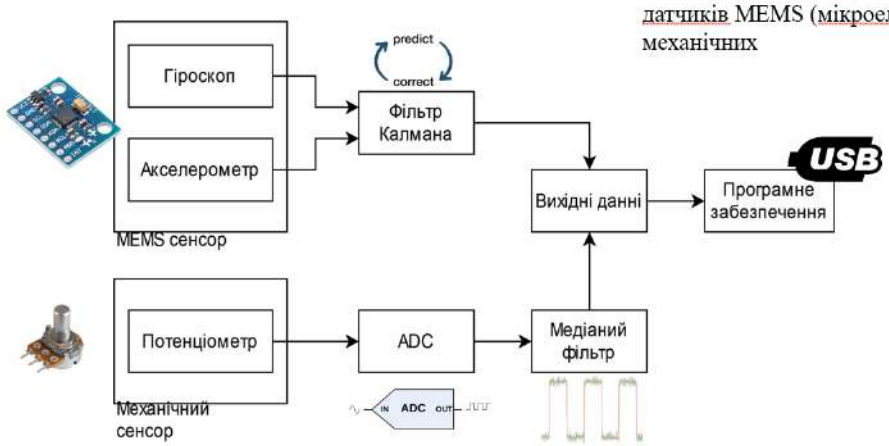
- універсальна та придатна для різних застосувань
- мати універсальний інтерфейс
- доступним
- дешевим

Задача даної роботи складається з розробки моделі та прототипу пристрою за допомогою мікроелектромеханічних та більш простих механічних датчиків



Опис моделі

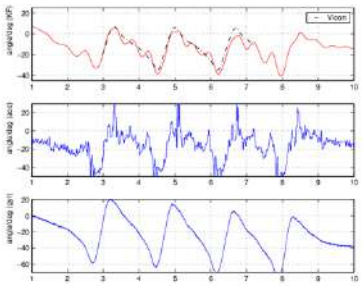
Модель відстеження людських кінцівок за допомогою датчиків MEMS (мікроелектромеханічних систем) та механічних



Методи обробки інформації

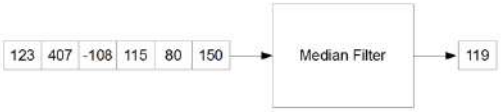
Фільтр Калмана

Використовуючи як математичну модель, так і фактичні вимірювання, фільтр Калмана може забезпечити точнішу та надійнішу оцінку стану системи, ніж будь-яке з цих джерел окремо.

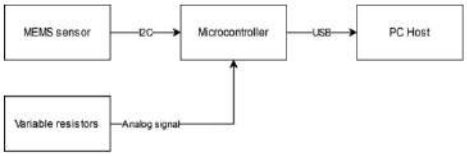


Медіанний фільтр

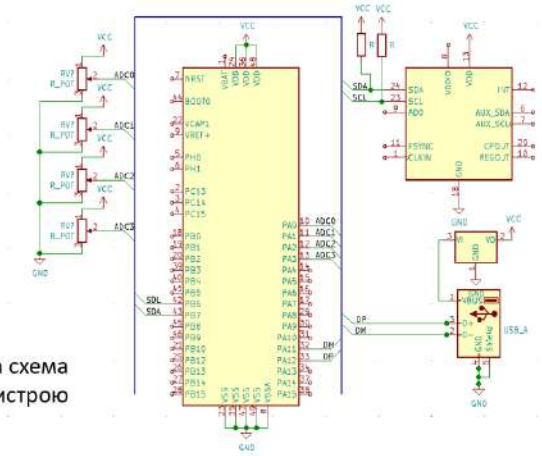
Замінюючи кожну точку даних середнім значенням, медіанний фільтр ефективно усуває викиди або шум, який може бути присутнім у вихідному сигналі.



Проектування пристрою

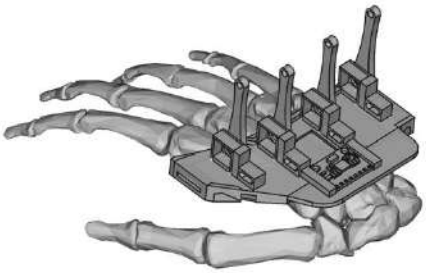


Спрощена модель яка демонструє підключення основних елементів системи



Електрична схема пристрою

Розробка прототипу



Розроблена 3D модель

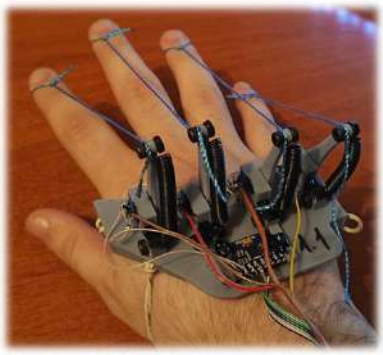
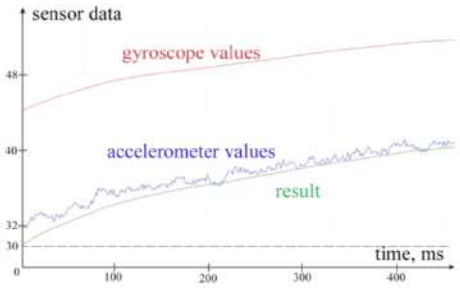
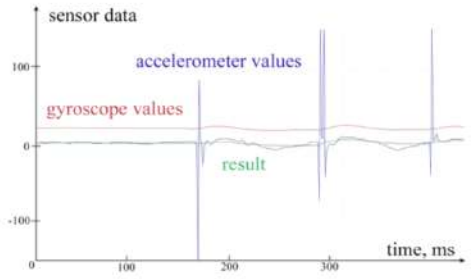


Фото зібраного прототипу

Перевірка роботи моделі

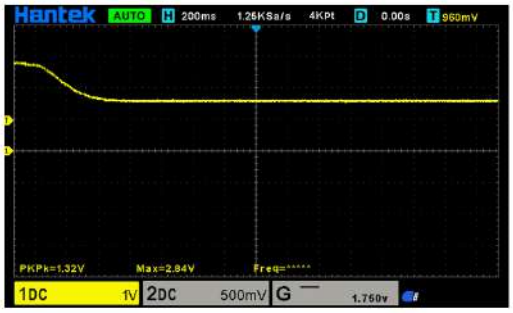


Приклад роботи моделі з MEMS сенсором наглядно демонструє роботу систему по зменшенню шумів



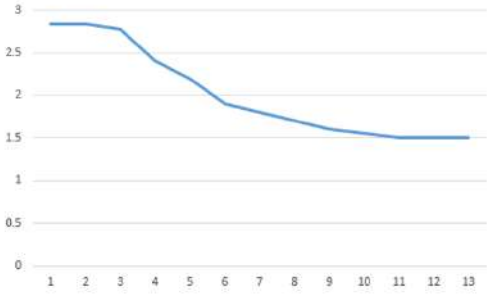
Крім того, сумісний підхід показує гарний результат при ударах датчиків або інших позаштатних різких відхилень.

Перевірка роботи моделі



Значення механічного датчику, зняті за допомогою осцилографу

Графік побудований по результатам роботи моделі показує, що отриманий сигнал на виході моделі є досить точним у порівнянні з тим що є насправді.



Висновок

Гіроскопи зазвичай використовуються в системах відстеження руху людини для вимірювання орієнтації та кутової швидкості сегментів тіла під час різних видів діяльності, наприклад аналізу ходи, моніторингу спортивних результатів. Що дало змогу розробити модель системи для точного стеження за кінцівками людини.

Також представлений прототип демонструє, що поєднуючи вимірювання від гіроскопів з вимірюваннями від інших датчиків, можна отримати досить точне уявлення про рух тіла.

ДОДАТОК Б

Програмний код

Б.1 Реалізація роботи пристрою

Б.1.1 Файл main.c

```
#include "main.h"
#include "i2c.h"
#include "usb_device.h"
#include "gpio.h"

//#include "mpu6050.h"
#include "MPU6050_6Axis_MotionApps_V6_12.h"

void SystemClock_Config(void);

typedef struct USB_REPORT {
int8_t x;
int8_t y;
int8_t z;
int8_t rx;
int8_t ry;
int8_t rz;
int8_t sl;
int8_t dl;
uint8_t buttons;
} USB_REPORT_X;

uint8_t map(uint16_t OldValue, uint16_t OldMin, uint16_t OldMax, uint8_t
NewMin, uint8_t NewMax)
{
uint16_t OldRange = OldMax - OldMin;
uint16_t NewRange = NewMax - NewMin;
return (((OldValue - OldMin) * NewRange) / OldRange) + NewMin;
}

void calibration(MPU6050 mpu) {
    const uint8_t BUFFER_SIZE = 100;
long offsets[6];
long offsetsOld[6];
int16_t mpuGet[6];

mpu.setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
mpu.setFullScaleGyroRange(MPU6050_GYRO_FS_250);

mpu.setXAccelOffset(0);
mpu.setYAccelOffset(0);
mpu.setZAccelOffset(0);
mpu.setXGyroOffset(0);
mpu.setYGyroOffset(0);
mpu.setZGyroOffset(0);
HAL_Delay(10);
for (uint8_t n = 0; n < 10; n++) {
      for (uint8_t j = 0; j < 6; j++) {
            offsets[j] = 0;
      }
```

```
            for (uint8_t i = 0; i < 100 + BUFFER_SIZE; i++) {
                  mpu.getMotion6(&mpuGet[0], &mpuGet[1], &mpuGet[2], &mpuGet[3],
&mpuGet[4], &mpuGet[5]);

                  if (i >= 99) {
                        for (uint8_t j = 0; j < 6; j++) {
                              offsets[j] += (long)mpuGet[j];
                        }
                  }
            }
            for (uint8_t i = 0; i < 6; i++) {
                  offsets[i] = offsetsOld[i] - ((long)offsets[i] / BUFFER_SIZE);
                  if (i == 2) offsets[i] += 16384;
                  offsetsOld[i] = offsets[i];
            }

            mpu.setXAccelOffset(offsets[0] / 8);
            mpu.setYAccelOffset(offsets[1] / 8);
            mpu.setZAccelOffset(offsets[2] / 8);
            mpu.setXGyroOffset(offsets[3] / 4);
            mpu.setYGyroOffset(offsets[4] / 4);
            mpu.setZGyroOffset(offsets[5] / 4);
            HAL_Delay(2);
      }
      }

      int main(void)
      {

      HAL_Init();
      SystemClock_Config();

      MX_GPIO_Init();
      MX_I2C1_Init();
      MX_USB_DEVICE_Init();

      I2Cdev i2c_dev;

      while (MPU6050_Init(&hi2c1) == 1);
      MPU6050_t mpu_data = {0};

      USB_REPORT_X usb_data = {0};

      MPU6050 mpu;
          mpu.reset();
          HAL_Delay(100);
          mpu.initialize();

          while(!mpu.testConnection()) {};
      mpu.setRate(3);
      uint8_t rate = mpu.getRate();
      uint8_t devStatus = mpu.dmpInitialize(); //0

      calibration(mpu);
      mpu.CalibrateAccel(1);
      mpu.CalibrateGyro(1);

      mpu.setDMPEnabled(true);
      uint8_t dmpen = mpu.getDMPEnabled();

      Quaternion q;
      uint16_t fifoCount;
      uint8_t fifoBuffer[64];
      uint8_t mpuIntStatus = mpu.getIntStatus();
```

```
uint16_t packetSize = mpu.dmpGetFIFOPacketSize();
mpu.resetFIFO();

float euler[3];
while (1)
{
      fifoCount = mpu.getFIFOCount();
      if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);

            uint8_t x = euler[0] * 180 / PI;

            MPU6050_Read_All(&hi2c1, &mpu_data);
            usb_data.x = mpu_data.KalmanAngleX;
            usb_data.y = mpu_data.KalmanAngleY;

            usb_data.x += 1;
            if(usb_data.x == 100) {
                  usb_data.x  = 0;
            }

            ADC_X = get_adc_data();
            map_adc_data(&usb_data, ADC_X);
            MX_USB_Send_Report((uint8_t*)&usb_data, sizeof(USB_REPORT_X));
      }
      HAL_Delay(100);
}
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USB;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Б.1.2 Файл mpu6050.c

```
#include <math.h>
#include "mpu6050.h"

#define RAD_TO_DEG 57.295779513082320876798154814105

#define WHO_AM_I_REG 0x75
#define PWR_MGMT_1_REG 0x6B
#define SMPLRT_DIV_REG 0x19
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define TEMP_OUT_H_REG 0x41
#define GYRO_CONFIG_REG 0x1B
#define GYRO_XOUT_H_REG 0x43

// Setup MPU6050
#define MPU6050_ADDR 0xD0
const uint16_t i2c_timeout = 100;
const double Accel_Z_corrector = 14418.0;
```

```
uint32_t timer;

Kalman_t KalmanX = {
    .Q_angle = 0.001f,
    .Q_bias = 0.003f,
    .R_measure = 0.03f};

Kalman_t KalmanY = {
    .Q_angle = 0.001f,
    .Q_bias = 0.003f,
    .R_measure = 0.03f,
};

uint8_t MPU6050_Init(I2C_HandleTypeDef *I2Cx)
{
    uint8_t check;
    uint8_t Data;

    // check device ID WHO_AM_I

    HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, WHO_AM_I_REG, 1, &check, 1,
i2c_timeout);

    if (check == 104) // 0x68 will be returned by the sensor if everything
goes well
    {
        // power management register 0X6B we should write all 0's to wake
the sensor up
        Data = 0;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data, 1,
i2c_timeout);

        // Set DATA RATE of 1KHz by writing SMPLRT_DIV register
        Data = 0x07;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, SMPLRT_DIV_REG, 1, &Data, 1,
i2c_timeout);

        // Set accelerometer configuration in ACCEL_CONFIG Register
        // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 -> пїS 2g
        Data = 0x00;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data,
1, i2c_timeout);

        // Set Gyroscopic configuration in GYRO_CONFIG Register
        // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 -> пїS 250 пїS/s
        Data = 0x00;
        HAL_I2C_Mem_Write(I2Cx, MPU6050_ADDR, GYRO_CONFIG_REG, 1, &Data,
1, i2c_timeout);
        return 0;
    }
    return 1;
}

void MPU6050_Read_Accel(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct)
{
    uint8_t Rec_Data[6];

    // Read 6 BYTES of data starting from ACCEL_XOUT_H register

    HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data, 6,
i2c_timeout);

    DataStruct->Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]);
```

```
        DataStruct->Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data[3]);
        DataStruct->Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data[5]);

        /*** convert the RAW values into acceleration in 'g'
             we have to divide according to the Full scale value set in FS_SEL
             I have configured FS_SEL = 0. So I am dividing by 16384.0
             for more details check ACCEL_CONFIG Register              ****/

        DataStruct->Ax = DataStruct->Accel_X_RAW / 16384.0;
        DataStruct->Ay = DataStruct->Accel_Y_RAW / 16384.0;
        DataStruct->Az = DataStruct->Accel_Z_RAW / Accel_Z_corrector;
    }

    void MPU6050_Read_Gyro(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct)
    {
        uint8_t Rec_Data[6];

        // Read 6 BYTES of data starting from GYRO_XOUT_H register

        HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, GYRO_XOUT_H_REG, 1, Rec_Data, 6,
i2c_timeout);

        DataStruct->Gyro_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]);
        DataStruct->Gyro_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data[3]);
        DataStruct->Gyro_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data[5]);

        /*** convert the RAW values into dps (пїS/s)
             we have to divide according to the Full scale value set in FS_SEL
             I have configured FS_SEL = 0. So I am dividing by 131.0
             for more details check GYRO_CONFIG Register              ****/

        DataStruct->Gx = DataStruct->Gyro_X_RAW / 131.0;
        DataStruct->Gy = DataStruct->Gyro_Y_RAW / 131.0;
        DataStruct->Gz = DataStruct->Gyro_Z_RAW / 131.0;
    }

    void MPU6050_Read_Temp(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct)
    {
        uint8_t Rec_Data[2];
        int16_t temp;

        // Read 2 BYTES of data starting from TEMP_OUT_H_REG register

        HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, TEMP_OUT_H_REG, 1, Rec_Data, 2,
i2c_timeout);

        temp = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]);
        DataStruct->Temperature = (float)((int16_t)temp / (float)340.0 +
(float)36.53);
    }

    void MPU6050_Read_All(I2C_HandleTypeDef *I2Cx, MPU6050_t *DataStruct)
    {
        uint8_t Rec_Data[14];
        int16_t temp;

        // Read 14 BYTES of data starting from ACCEL_XOUT_H register

        HAL_I2C_Mem_Read(I2Cx, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data,
14, i2c_timeout);

        DataStruct->Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data[1]);
        DataStruct->Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data[3]);
        DataStruct->Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data[5]);
```

```
          temp = (int16_t)(Rec_Data[6] << 8 | Rec_Data[7]);
          DataStruct->Gyro_X_RAW = (int16_t)(Rec_Data[8] << 8 | Rec_Data[9]);
          DataStruct->Gyro_Y_RAW = (int16_t)(Rec_Data[10] << 8 | Rec_Data[11]);
          DataStruct->Gyro_Z_RAW = (int16_t)(Rec_Data[12] << 8 | Rec_Data[13]);

          DataStruct->Ax = DataStruct->Accel_X_RAW / 16384.0;
          DataStruct->Ay = DataStruct->Accel_Y_RAW / 16384.0;
          DataStruct->Az = DataStruct->Accel_Z_RAW / Accel_Z_corrector;
          DataStruct->Temperature  =  (float)((int16_t)temp  /  (float)340.0  +
(float)36.53);
          DataStruct->Gx = DataStruct->Gyro_X_RAW / 131.0;
          DataStruct->Gy = DataStruct->Gyro_Y_RAW / 131.0;
          DataStruct->Gz = DataStruct->Gyro_Z_RAW / 131.0;

          // Kalman angle solve
          double dt = (double)(HAL_GetTick() - timer) / 1000;
          timer = HAL_GetTick();
          double roll;
          double roll_sqrt = sqrt(
              DataStruct->Accel_X_RAW  *  DataStruct->Accel_X_RAW  +  DataStruct-
>Accel_Z_RAW * DataStruct->Accel_Z_RAW);
          if (roll_sqrt != 0.0)
          {
              roll = atan(DataStruct->Accel_Y_RAW / roll_sqrt) * RAD_TO_DEG;
          }
          else
          {
              roll = 0.0;
          }
          double    pitch    =    atan2(-DataStruct->Accel_X_RAW,    DataStruct-
>Accel_Z_RAW) * RAD_TO_DEG;
          if ((pitch < -90 && DataStruct->KalmanAngleY > 90) || (pitch > 90 &&
DataStruct->KalmanAngleY < -90))
          {
              KalmanY.angle = pitch;
              DataStruct->KalmanAngleY = pitch;
          }
          else
          {
              DataStruct->KalmanAngleY   =   Kalman_getAngle(&KalmanY,   pitch,
DataStruct->Gy, dt);
          }
          if (fabs(DataStruct->KalmanAngleY) > 90)
              DataStruct->Gx = -DataStruct->Gx;
          DataStruct->KalmanAngleX = Kalman_getAngle(&KalmanX, roll, DataStruct-
>Gx, dt);
      }

      double Kalman_getAngle(Kalman_t *Kalman, double newAngle, double newRate,
double dt)
      {
          double rate = newRate - Kalman->bias;
          Kalman->angle += dt * rate;

          Kalman->P[0][0]  +=  dt  *  (dt  *  Kalman->P[1][1]  -  Kalman->P[0][1]  -
Kalman->P[1][0] + Kalman->Q_angle);
          Kalman->P[0][1] -= dt * Kalman->P[1][1];
          Kalman->P[1][0] -= dt * Kalman->P[1][1];
          Kalman->P[1][1] += Kalman->Q_bias * dt;

          double S = Kalman->P[0][0] + Kalman->R_measure;
          double K[2];
          K[0] = Kalman->P[0][0] / S;
          K[1] = Kalman->P[1][0] / S;
```

```
    double y = newAngle - Kalman->angle;
    Kalman->angle += K[0] * y;
    Kalman->bias += K[1] * y;

    double P00_temp = Kalman->P[0][0];
    double P01_temp = Kalman->P[0][1];

    Kalman->P[0][0] -= K[0] * P00_temp;
    Kalman->P[0][1] -= K[0] * P01_temp;
    Kalman->P[1][0] -= K[1] * P00_temp;
    Kalman->P[1][1] -= K[1] * P01_temp;

    return Kalman->angle;
};
```

Б.1 Реалізація роботи додатку

Б.2.1 Файл main.cpp

```
#include <iostream>
#include <cmath>
#include <glut.h>

#include "JoyImpl.h"

JoyImpl joy;
float angle = 0.0f;

void changeSize(int w, int h) {
if (h == 0)
      h = 1;
float ratio = w * 1.0 / h;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h);
gluPerspective(45.0f, ratio, 0.1f, 100.0f);
glMatrixMode(GL_MODELVIEW);
}

void draw_base() {
glLineWidth(1);
glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL_LINE_LOOP); // top
glVertex3f(-1.5f, 1.0f, 0.0f);
glVertex3f(1.5f, 1.0f, 0.0f);
glVertex3f(1.0f, -1.0f, 0.0f);
glVertex3f(-1.0f, -1.0f, 0.0f);
glVertex3f(-1.5f, 0.0f, 0.0f);
glEnd();

glBegin(GL_LINE_LOOP); // bot
glVertex3f(-1.5f, 1.0f, -1.0f);
glVertex3f(1.5f, 1.0f, -1.0f);
glVertex3f(1.0f, -1.0f, -0.5f);
glVertex3f(-1.0f, -1.0f, -0.5f);
glVertex3f(-1.5f, 0.0f, -0.75f);
glEnd();
```

```
	glBegin(GL_LINES); // merge
	glVertex3f(-1.5f, 1.0f, 0.0f);
	glVertex3f(-1.5f, 1.0f, -1.0f);

	glVertex3f(1.5f, 1.0f, 0.0f);
	glVertex3f(1.5f, 1.0f, -1.0f);

	glVertex3f(1.0f, -1.0f, 0.0f);
	glVertex3f(1.0f, -1.0f, -0.5f);

	glVertex3f(-1.0f, -1.0f, 0.0f);
	glVertex3f(-1.0f, -1.0f, -0.5f);

	glVertex3f(-1.5f, 0.0f, 0.0f);
	glVertex3f(-1.5f, 0.0f, -0.75f);
	glEnd();
	}

	#define STAGE1_COEF 3.0f
	#define STAGE2_COEF 1.5f
	#define STAGE3_COEF 1.0f
	#define DEFAULT_Y_POS 1.0f
	#define DEFAULT_Z_POS -0.5f

	#define USE_JOY JOYSTICK_1

	void draw_finger(float x_pos, float stage1_len, float stage2_len, float
stage3_len, float value) {
	float stage1_y = cos(value / STAGE1_COEF) * stage1_len + DEFAULT_Y_POS;
	float stage1_z = sin(value / STAGE1_COEF) * stage1_len + DEFAULT_Z_POS;

	float stage2_y = cos(value / STAGE2_COEF) * stage2_len + stage1_y;
	float stage2_z = sin(value / STAGE2_COEF) * stage2_len + stage1_z;

	float stage3_y = cos(value / STAGE3_COEF) * stage3_len + stage2_y;
	float stage3_z = sin(value / STAGE3_COEF) * stage3_len + stage2_z;

	glLineWidth(9);
	glBegin(GL_LINE_STRIP);
	glVertex3f(x_pos, DEFAULT_Y_POS, DEFAULT_Z_POS);
	glVertex3f(x_pos, stage1_y, stage1_z);
	glVertex3f(x_pos, stage2_y, stage2_z);
	glVertex3f(x_pos, stage3_y, stage3_z);
	glEnd();
	}

	void draw_thumb(float x_pos, float y_pos, float x_pos_stage1, float
stage1_len, float stage2_len, float x_value, float z_value) {
	float stage1_x = sin(x_value / STAGE2_COEF) * stage1_len + x_pos_stage1;
	float stage1_z = sin(z_value / STAGE2_COEF) * stage1_len + DEFAULT_Z_POS;
	float stage1_y = cos(x_value / STAGE2_COEF) + cos(z_value / STAGE2_COEF) *
stage1_len - 0.5;

	float stage2_x = sin(x_value / STAGE3_COEF) * stage2_len + stage1_x;
	float stage2_z = sin(z_value / STAGE3_COEF) * stage2_len + stage1_z;
	float stage2_y = cos(x_value / STAGE3_COEF) + cos(z_value / STAGE3_COEF) *
stage2_len + stage1_y;

	glLineWidth(11);
	glBegin(GL_LINE_STRIP);
	glVertex3f(x_pos, y_pos, DEFAULT_Z_POS);
	glVertex3f(stage1_x, stage1_y, stage1_z);
	glVertex3f(stage2_x, stage2_y, stage2_z);
	glEnd();
```

```
}

void renderScene(void) {
joy.update_joy_value(USE_JOY);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(0.0f, 0.0f, 10.0f,
      0.0f, 0.0f, 0.0f,
      0.0f, 1.0f, 0.0f);

glRotatef(angle + 45, 0.0f, 1.0f, 0.0f);
glRotatef(-60, 1.0f, 0.0f, 0.0f);

draw_base();

float val1 = joy.get_joy_axe(USE_JOY, JOYSTICK_AXIS_LEFT_X);
val1 *= 2;
float val2 = joy.get_joy_axe(USE_JOY, JOYSTICK_AXIS_RIGHT_X);
val2 *= 2;

// 1
draw_finger(-1.25f, 1, 1, 1, val1);
// 2
draw_finger(-0.25f, 1, 1.5, 1, val2);
// 3
draw_finger(0.75f, 1, 1, 1, 0);
// 4
draw_finger(1.4f, 0.5, 0.5, 1, 0);
// 5
draw_thumb(-1.5f, 0.5f, -1.7f, 0.5f, 0.75f, 0, 0);

glutSwapBuffers();
}

void processSpecialKeys(int key, int x, int y) {
switch (key) {
case GLUT_KEY_LEFT: angle += 1.0f; break;
case GLUT_KEY_RIGHT: angle -= 1.0f; break;
default: break;
}
}

int main(int argc, char** argv)
{
joy.init();

std::cout << "-------------DEVICES INFO---------------" << std::endl;
joy.print_joy_info();
std::cout << "----------------------------------------" << std::endl <<
std::endl;;

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100, 100);
glutCreateWindow("Main window");

glutDisplayFunc(renderScene);
glutReshapeFunc(changeSize);
glutIdleFunc(renderScene);

glutSpecialFunc(processSpecialKeys);

glutMainLoop();
```

```
return 0;
}
```

Б.2.2 Файл joyimpl.h

```
#include <iostream>
#include <glfw3.h>

#define JOYSTICK_1             0
#define JOYSTICK_2             1
#define JOYSTICK_3             2
#define JOYSTICK_4             3
#define JOYSTICK_5             4
#define JOYSTICK_6             5
#define JOYSTICK_7             6
#define JOYSTICK_8             7
#define JOYSTICK_9             8
#define JOYSTICK_10            9
#define JOYSTICK_11            10
#define JOYSTICK_12            11
#define JOYSTICK_13            12
#define JOYSTICK_14            13
#define JOYSTICK_15            14
#define JOYSTICK_16            15
#define JOYSTICK_LAST          JOYSTICK_16

#define JOYSTICK_AXIS_LEFT_X        0
#define JOYSTICK_AXIS_LEFT_Y        1
#define JOYSTICK_AXIS_RIGHT_X       2
#define JOYSTICK_AXIS_RIGHT_Y       5
#define JOYSTICK_AXIS_LEFT_TRIGGER  3
#define JOYSTICK_AXIS_RIGHT_TRIGGER 4

class JoyImpl {
public:

JoyImpl() {

}

void init() {
      glfwInit();
      scan_joy();
}

void scan_joy() {
      for (int i = 0; i < JOYSTICK_LAST; ++i) {
            if (glfwJoystickPresent(i)) {
                  JOY_INFO[i].present = true;
                  strcpy_s(JOY_INFO[i].name, glfwGetJoystickName(i));
                  glfwGetJoystickAxes(i, &JOY_INFO[i].axes_count);
            }
            else {
                  JOY_INFO[i].present = false;
                  strcpy_s(JOY_INFO[i].name, "-");
                  JOY_INFO[i].axes_count = 0;
            }
      }
}

void print_joy_info() {
```

```
            std::cout << "Id\tAxes\tName" << "\r\n";
            for (int i = 0; i < JOYSTICK_LAST; ++i) {
                  if (JOY_INFO[i].present) {
                        std::cout << i << "\t" << JOY_INFO[i].axes_count << "\t"
<< JOY_INFO[i].name << "\r\n";
                  }
            }
      }

      void update_joy_value(int jid) {
            if (!JOY_INFO[jid].present) {
                  JOY_INFO[jid].axes_count = 0;
                  return;
            }

            int count;
            const float* axes = glfwGetJoystickAxes(jid, &count);
            for (int i = 0; i < count; ++i) {
                  JOY_INFO[jid].axes_value[i] = axes[i];
            }
      }

      float get_joy_axe(int jid, int axe) {
            if (!JOY_INFO[jid].present) {
                  return 0;
            }

            if (JOY_INFO[jid].axes_count < axe) {
                  return 0;
            }

            return JOY_INFO[jid].axes_value[axe];
      }

      private:
      struct joy_info {
            bool present;
            char name[32];
            int axes_count;
            float axes_value[18];
      } JOY_INFO[JOYSTICK_LAST];
      };
```

ДОДАТОК В
СХЕМА ПРИСТРОЮ

В.1 Схема плати пристрою