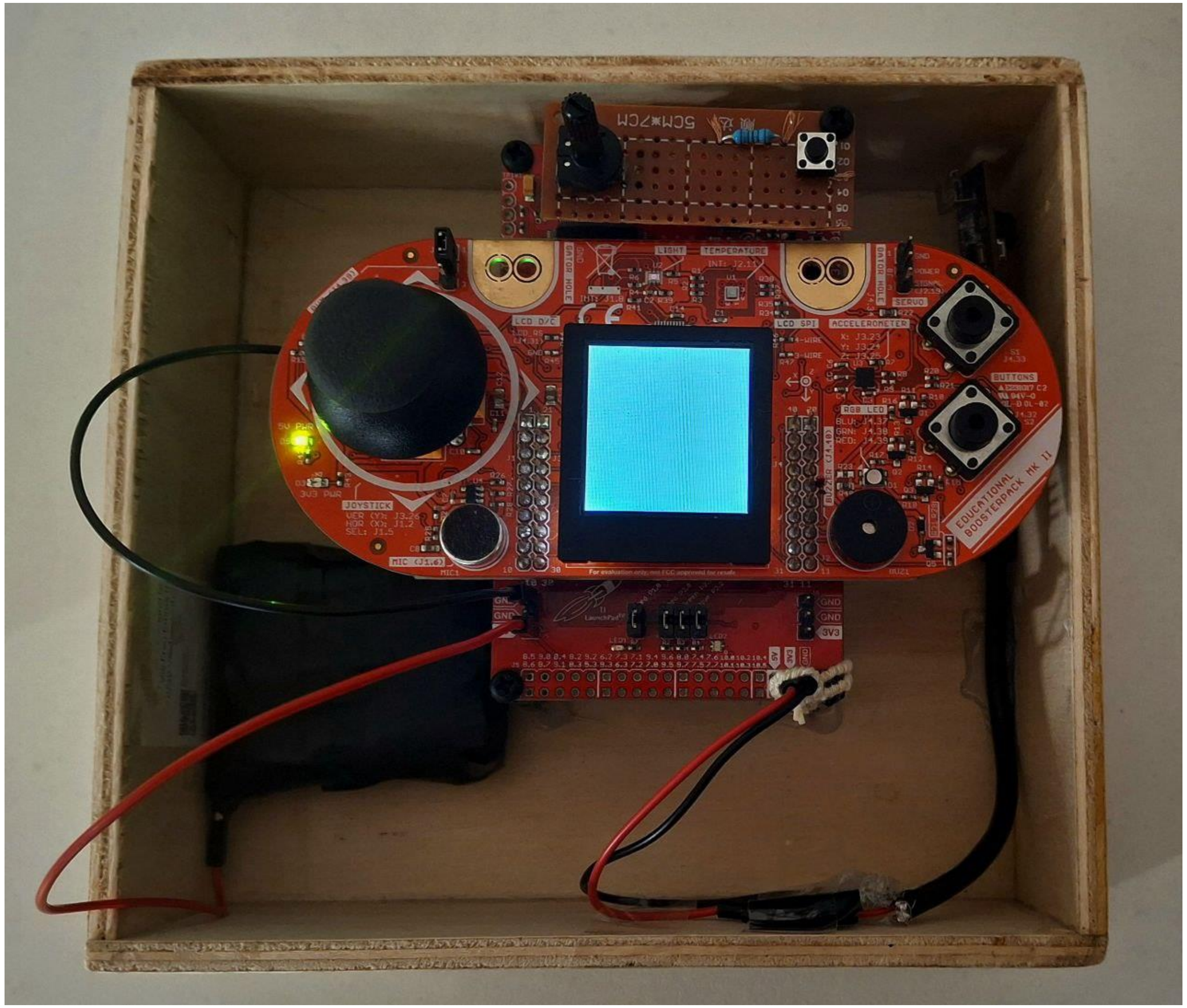
ERPC

ESSENTIAL REMOTE PC CONTROL

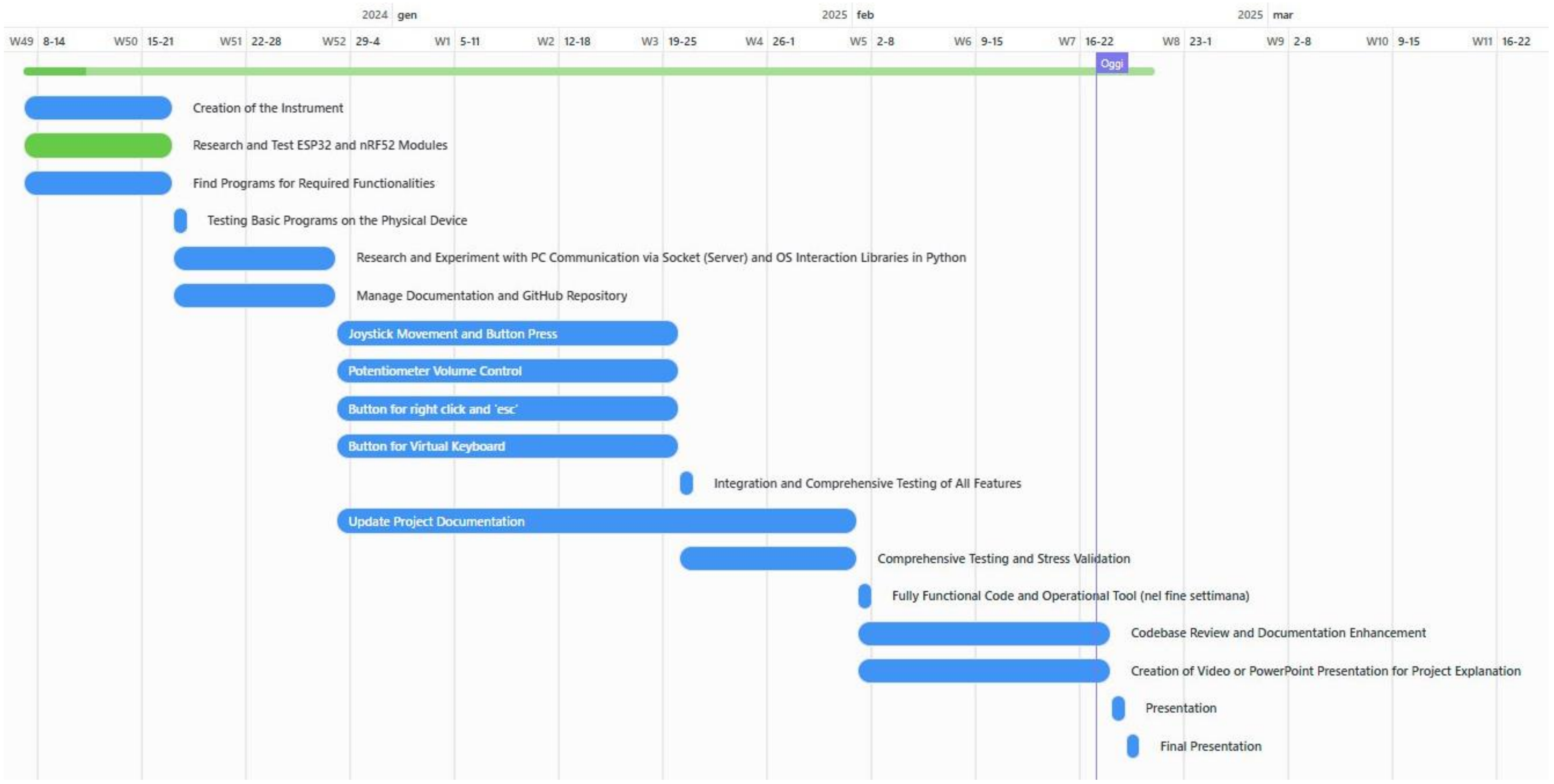
INTRODUCTION

Features:

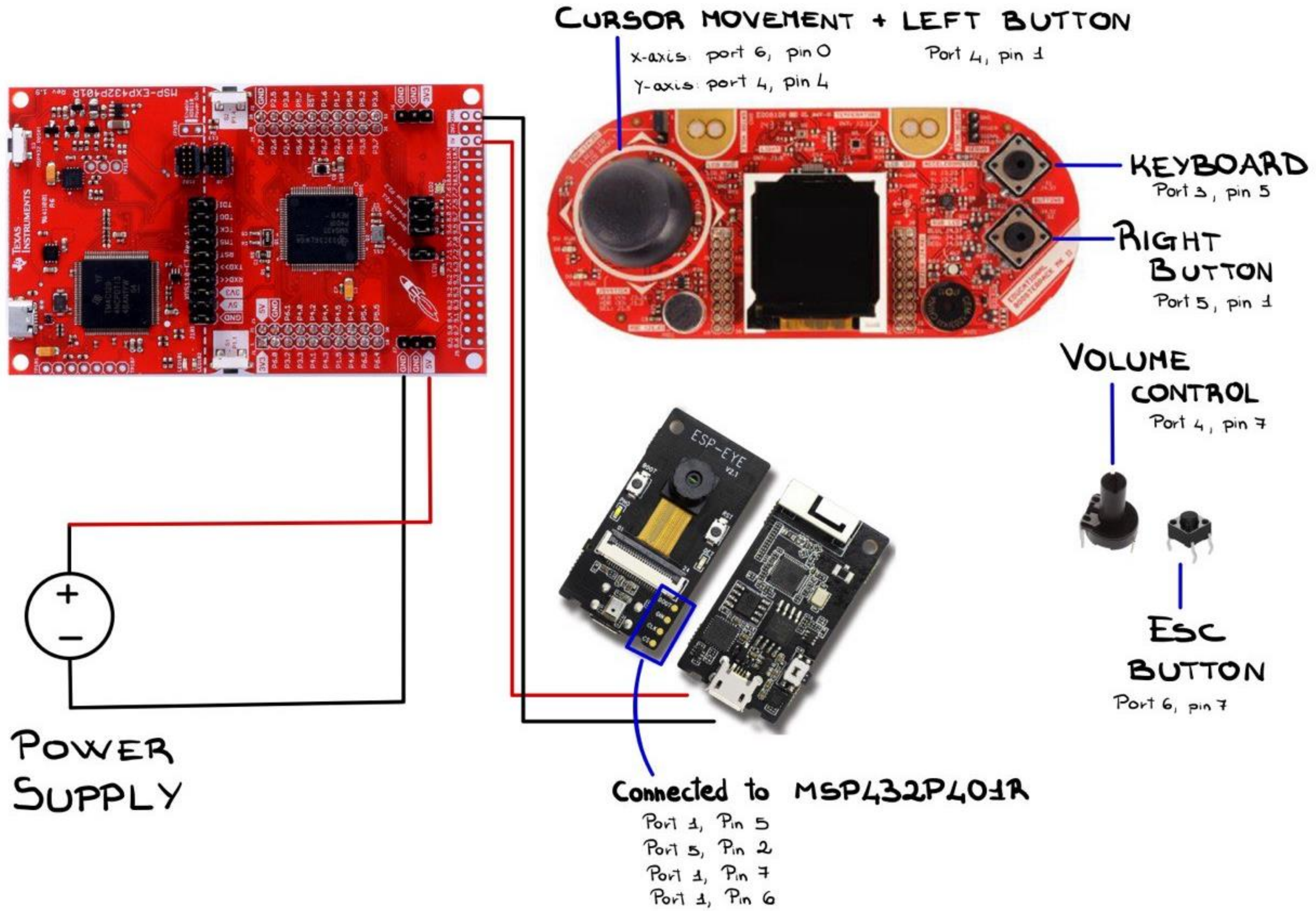
- Right and left buttons
- Cursor movement
- Volume control
- ESC button
- Virtual keyboard open button



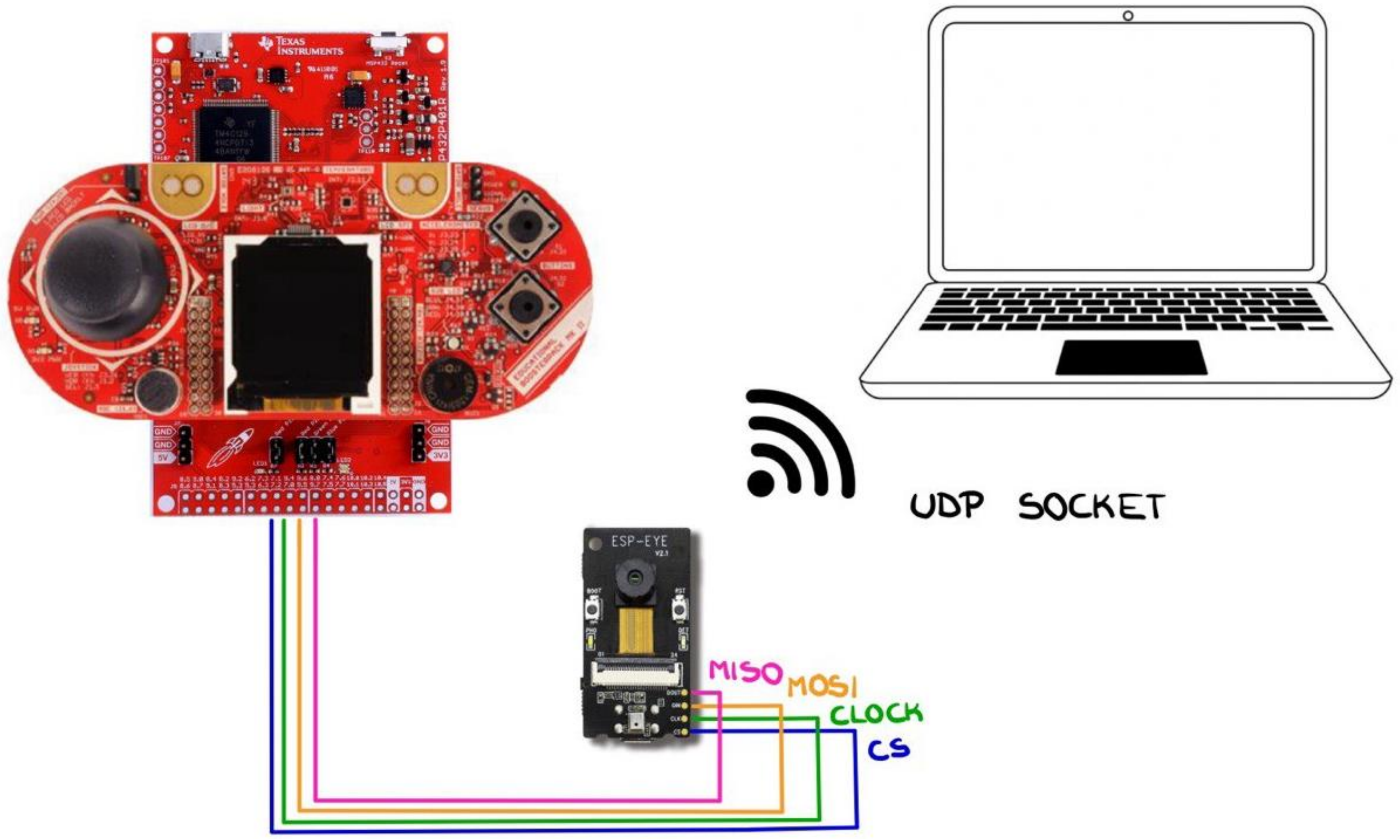
GANTT DIAGRAM



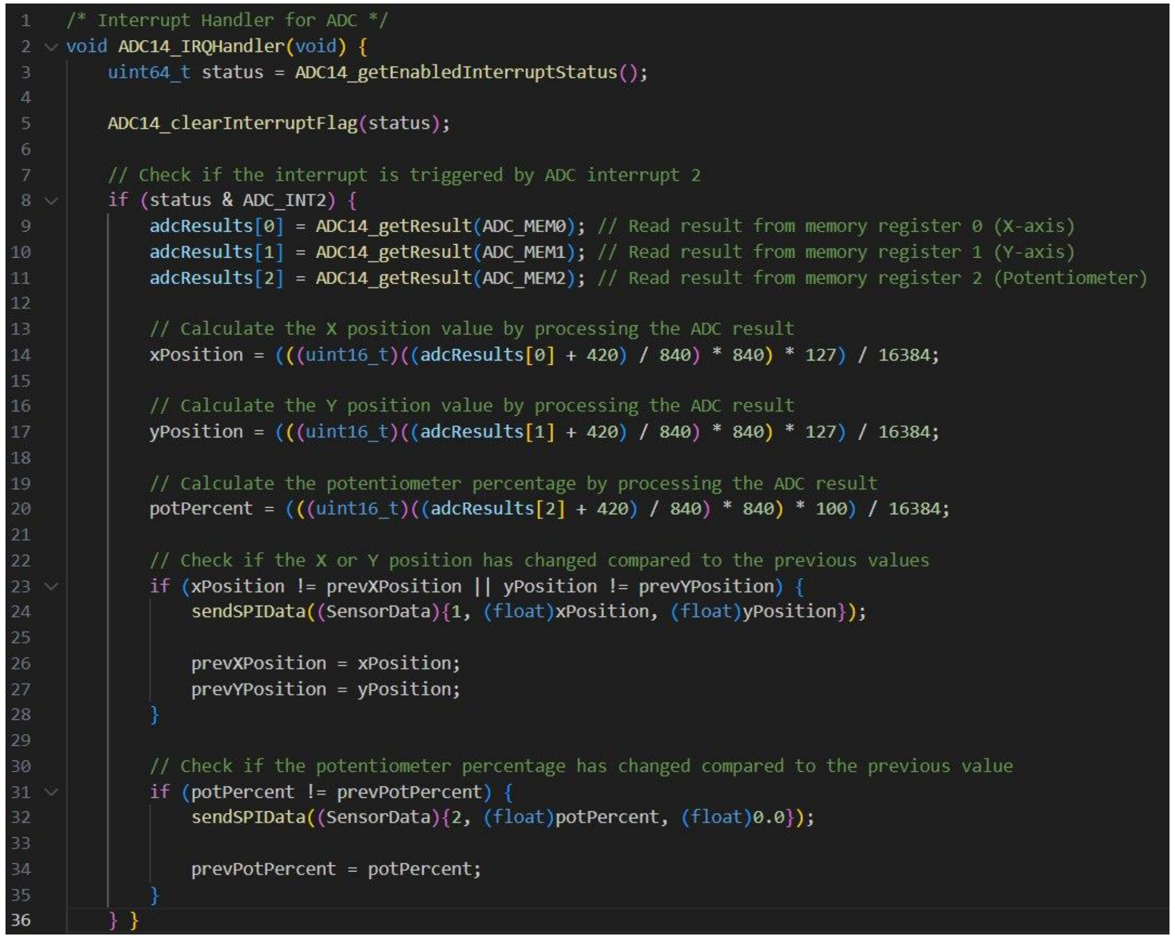
SOFTWARE-HARDWARE INTERACTION



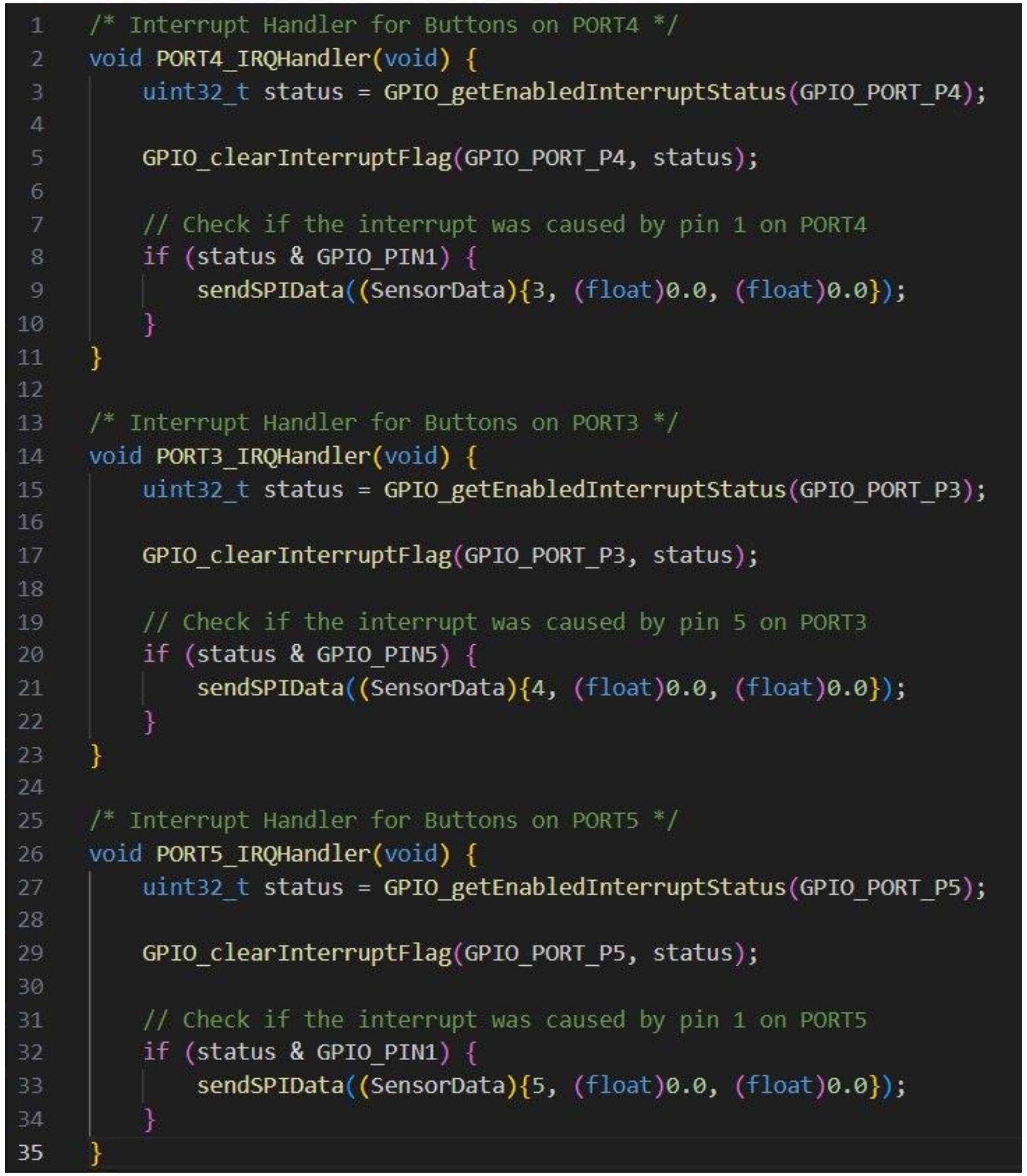
COMMUNICATION BETWEEN DEVICES



RELEVANT CODE



```c
/* Interrupt Handler for ADC */
void ADC14_IRQHandler(void) {
    uint64_t status = ADC14_getEnabledInterruptStatus();

    ADC14_clearInterruptFlag(status);

    // Check if the interrupt is triggered by ADC interrupt 2
    if (status & ADC_INT2) {
        adcResults[0] = ADC14_getResult(ADC_MEM0); // Read result from memory register 0 (X-axis)
        adcResults[1] = ADC14_getResult(ADC_MEM1); // Read result from memory register 1 (Y-axis)
        adcResults[2] = ADC14_getResult(ADC_MEM2); // Read result from memory register 2 (Potentiometer)

        // Calculate the X position value by processing the ADC result
        xPosition = (((uint16_t)((adcResults[0] + 420) / 840) * 840) * 127) / 16384;

        // Calculate the Y position value by processing the ADC result
        yPosition = (((uint16_t)((adcResults[1] + 420) / 840) * 840) * 127) / 16384;

        // Calculate the potentiometer percentage by processing the ADC result
        potPercent = (((uint16_t)((adcResults[2] + 420) / 840) * 840) * 100) / 16384;

        // Check if the X or Y position has changed compared to the previous values
        if (xPosition != prevXPosition || yPosition != prevYPosition) {
            sendSPIData((SensorData){1, (float)xPosition, (float)yPosition});

            prevXPosition = xPosition;
            prevYPosition = yPosition;
        }

        // Check if the potentiometer percentage has changed compared to the previous value
        if (potPercent != prevPotPercent) {
            sendSPIData((SensorData){2, (float)potPercent, (float)0.0});

            prevPotPercent = potPercent;
        }
    } }
```



```c
/* Interrupt Handler for Buttons on PORT4 */
void PORT4_IRQHandler(void) {
    uint32_t status = GPIO_getEnabledInterruptStatus(GPIO_PORT_P4);

    GPIO_clearInterruptFlag(GPIO_PORT_P4, status);

    // Check if the interrupt was caused by pin 1 on PORT4
    if (status & GPIO_PIN1) {
        sendSPIData((SensorData){3, (float)0.0, (float)0.0});
    }
}

/* Interrupt Handler for Buttons on PORT3 */
void PORT3_IRQHandler(void) {
    uint32_t status = GPIO_getEnabledInterruptStatus(GPIO_PORT_P3);

    GPIO_clearInterruptFlag(GPIO_PORT_P3, status);

    // Check if the interrupt was caused by pin 5 on PORT3
    if (status & GPIO_PIN5) {
        sendSPIData((SensorData){4, (float)0.0, (float)0.0});
    }
}

/* Interrupt Handler for Buttons on PORT5 */
void PORT5_IRQHandler(void) {
    uint32_t status = GPIO_getEnabledInterruptStatus(GPIO_PORT_P5);

    GPIO_clearInterruptFlag(GPIO_PORT_P5, status);

    // Check if the interrupt was caused by pin 1 on PORT5
    if (status & GPIO_PIN1) {
        sendSPIData((SensorData){5, (float)0.0, (float)0.0});
    }
}
```

TESTING & DEBUGGING

- No documentation for pin settings
- Measurement frequency too high

IMPROVEMENTS

- Security
- Bluetooth connection
- External devices:
  - Display
  - Microphone
  - Touch screen