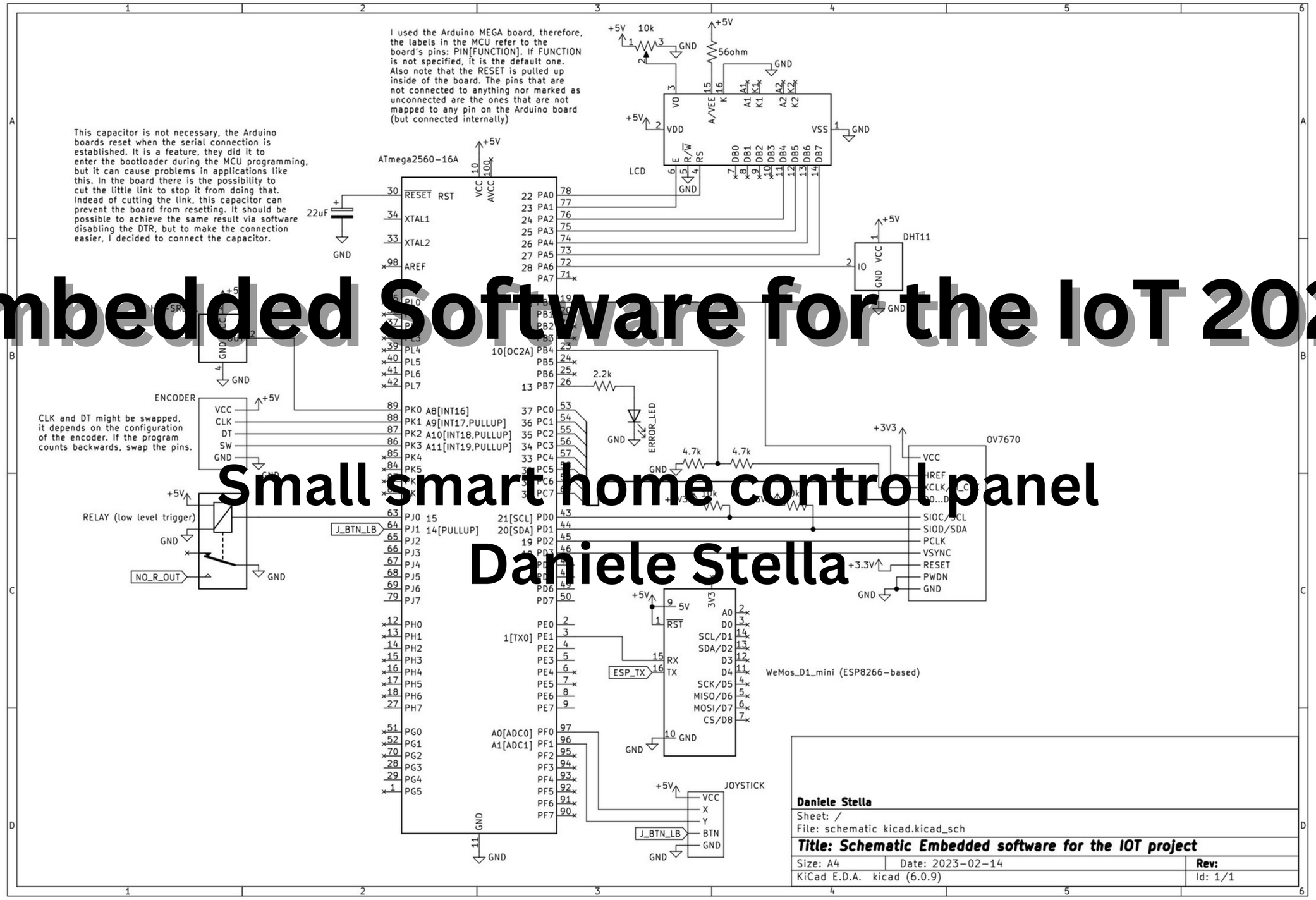


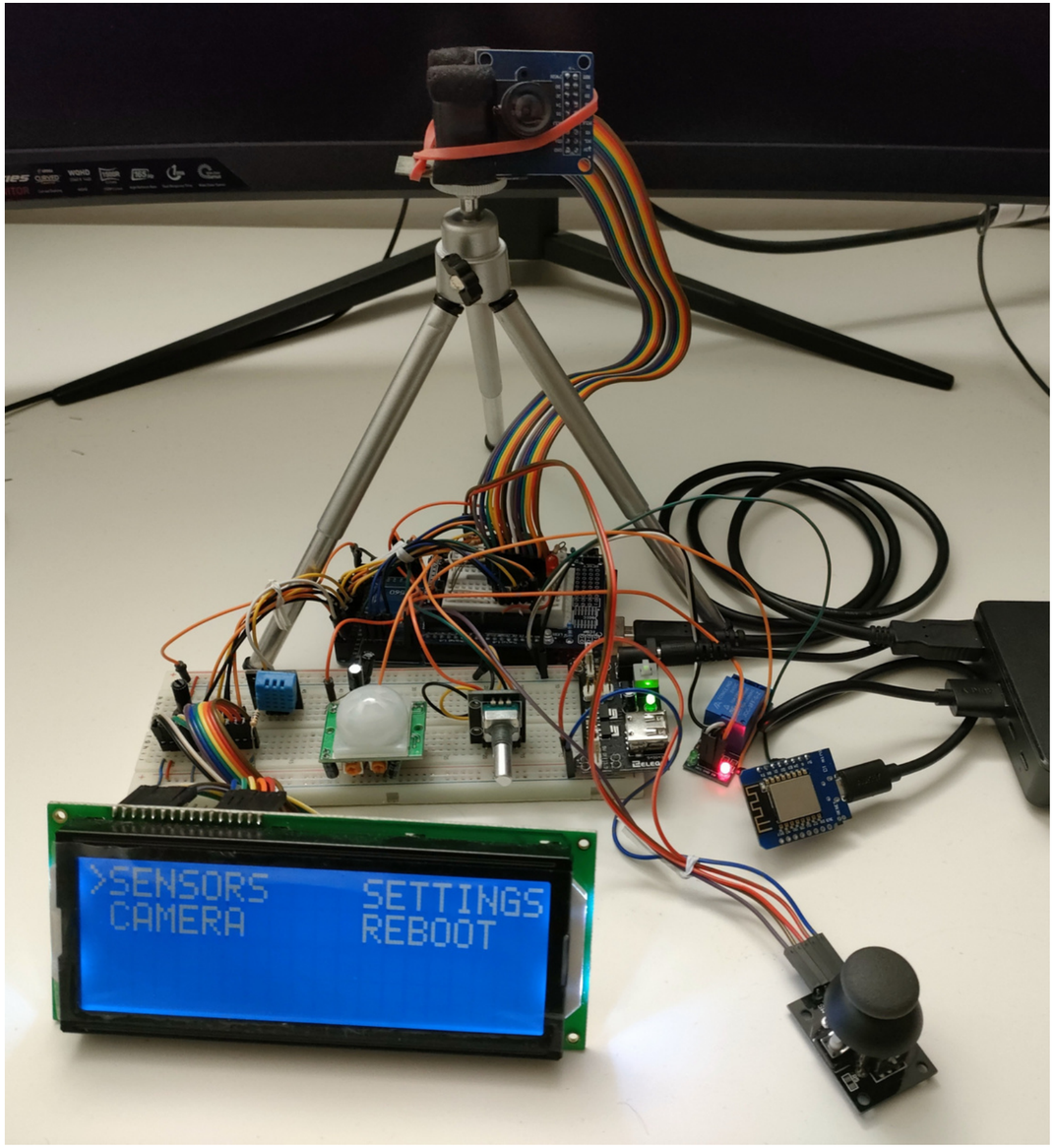
Embedded Software for the IoT 2022

Small smart home control panel

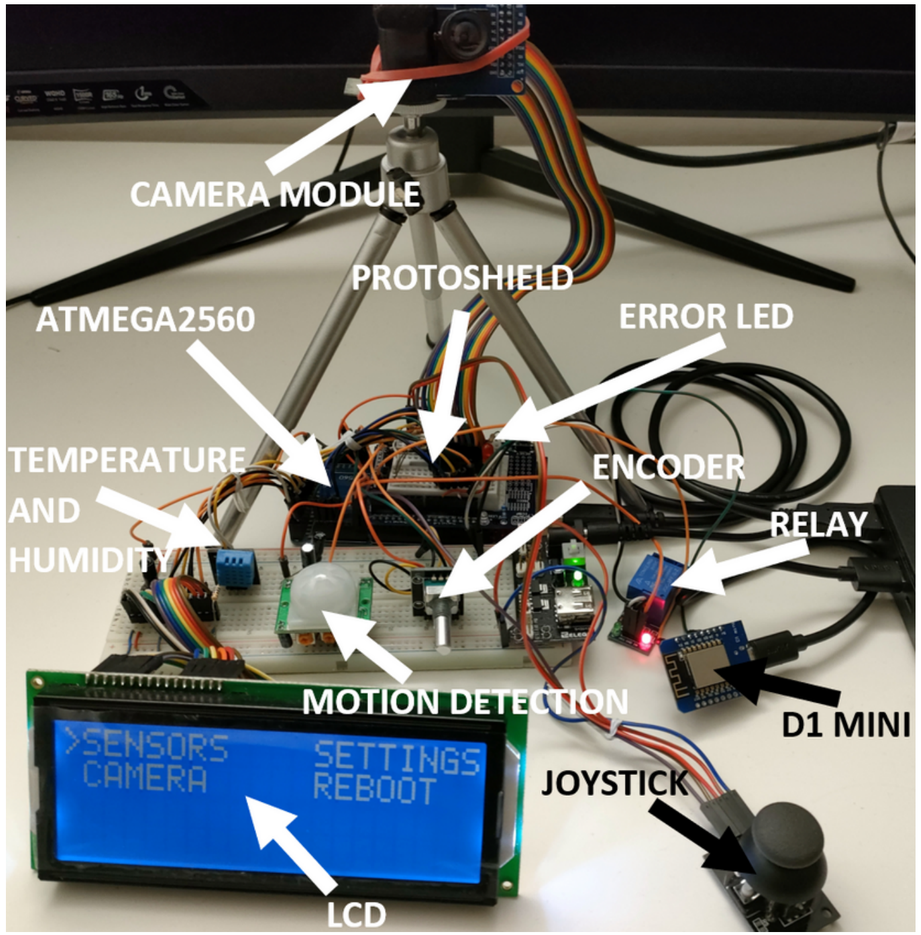
Daniele Stella

Problem statement

Small smart home control panel with sensors, a camera module, and network capabilities to transfer the images.

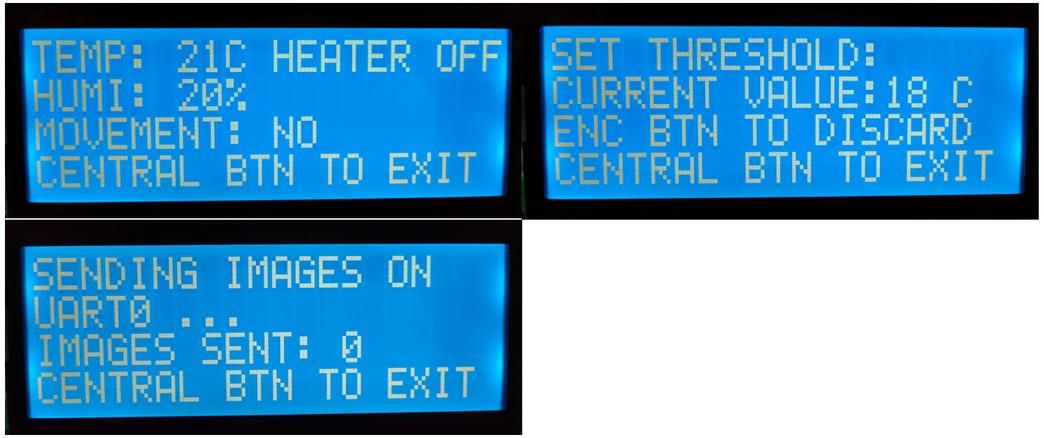


Basic Working Scheme



Workflow:

The user interacts with the system using the joystick and selects the menu.



In each one, the joystick button is used to exit. In the settings menu, the encoder is used to change the threshold, and its button is used to discard the changes.

The heater (relay) is turned on when the temperature is below the set threshold.

Temperature, humidity and motion are interrupt-controlled and constantly updated.

The raw images are sent on UART0 asynchronously.

The images are sent to the D1 mini. The D1 connects to WiFi and hosts a web server on port 80 and a WebSocket on port 81. It sends the image to the client along with some JavaScript code to render it.

Software Architecture - ATmega2560

This MCU was programmed in C without any library. Every hardware/software interaction can be seen easily. The Timer1 is working in normal mode, CTC, to call the interrupt every 1 ms. It is used to keep track of time, especially to respect the 1 s sample time of the DHT11.

```
ISR(TIMER1_COMPA_vect) {
    // called every 1ms
    ++curr_time_ms;
}
```

The camera communicates with the MCU via TWI.

```
TWSR &= ~((1 << TWPS0) | (1 << TWPS1));
TWBR = 72;  // set to 100khz
```

Encoder connected to external interrupt pins belonging to the PCINT2.
Fast debouncing (timer is interrupt based) in the ISR to avoid misreads.

```
ISR(PCINT2_vect) {
    int encoder_value = ENCODER_CLK_STATE | (ENCODER_DT_STATE << 1);
    if ((curr_time_ms - lastDebounceTime) > debounceDelay) {
        switch (encoder_value) {
            case 0b00:
                break;
            case 0b01:
                encoder_position++;
                encoder_changed = true;
                break;
            case 0b10:
                encoder_position--;
                encoder_changed = true;
                break;
            case 0b11:
                break;
        }
        lastDebounceTime = curr_time_ms;
    }
    if (!(PIN_ENCODER_SW & (1 << ENCODER_SW)))
        encoder_sw_pressed_interrupt = true;
    PCIFR |= (1 << PCIF2); // Reset interrupt flag
}
```

The ADC for the joystick is handled with busy waiting. This is because it is at the fastest operation setting possible, I do not need accuracy but using it without interrupts made everything really clear and easy to understand.

```
while (!(ADCSRA & (1 << ADIF)))
    ;
ADCSRA |= (1 << ADIF);
```

The joystick central button is connected to an external interrupt.

```
ISR(PCINT1_vect) {
    if (!(PINJ & (1 << PINJ1)))
        central_button_pressed_interrupt = true;
    PCIFR |= (1 << PCIF1);  // Reset interrupt flag
}
```

Everything used for the camera (Timer2 master clk, UART, TWI) is busy waiting based.

```
UDR0 = (PINC & ((1 << PINC0) | (1 << PINC1) |
                (1 << PINC2) | (1 << PINC3) |
                (1 << PINC4) | (1 << PINC5) |
                (1 << PINC6) | (1 << PINC7)));
WAIT_FOR_BYTE_TRANSMISSION_USART
WAIT_FOR_PIN_HIGH(PIND, PIND2)
WAIT_FOR_PIN_LOW(PIND, PIND2)
WAIT_FOR_PIN_HIGH(PIND, PIND2)
```

The data from the DHT11 is double-checked with the checksum provided by the sensor.

```
uint8_t check = bits2byte(data + 32);
uint8_t expect = humidity + humidity2 +
if (check != expect)
    return 105;
```

Software Architecture - D1 mini

The hardware interaction is hidden here because of the libraries and the firmware.

```
    find_new_image();
  } else {
    if (pixels_in_buffer == 0) {
      current_buffer = (use_buffer_1) ? brightness_buffer_1 : brightness_buffer_2;
      next_buffer = (use_buffer_1) ? brightness_buffer_2 : brightness_buffer_1;
    }
    current_buffer[pixels_in_buffer] = (uint8_t)Serial.read();
    ++pixels_in_buffer;
    ++pixels;
    if (pixels_in_buffer == bufferSize) {
      webSocket.broadcastBIN(current_buffer, bufferSize);
      use_buffer_1 = !use_buffer_1;
      pixels_in_buffer = 0;
      if (pixels == HEIGHT * WIDTH) {
        new_image = true;
        pixels = 0;
```

```
socket.onmessage = function (event) {
    const brightness = new Uint8Array(event.data);
    brightnessArray.set(brightness, i++ * width * 60);
    if (i * width * 60 == width * height) {
        i = 0;  renderImage(canvas_num++);
    }
}
```

It hosts a web server on port 80, which sends the HTML and JS to the client.
It also hosts a WebSocket on port 81 to send the image data.
It finds an image marker (*RDY*) and then forwards everything via the WebSocket with a binary broadcast.

Double buffering: two buffers of 320*60 bytes (1/4 of the image).

The image is rendered client-side

```
function renderImage(canvasId) {
    var canvas = document.createElement('canvas');
    canvas.id = 'canvas' + canvasId.toString();
    canvas.width = width;   canvas.height = height;
    document.body.appendChild(canvas);
    var ctx = canvas.getContext('2d');
    var imageData = ctx.createImageData(width, height);
    for (var i = 0; i < brightnessArray.length; i++) {
        var brightness = brightnessArray[i];
        var index = i * 4;
        imageData.data[index] = brightness;
        imageData.data[index + 1] = brightness;
        imageData.data[index + 2] = brightness;
        imageData.data[index + 3] = 255;
    }
    ctx.putImageData(imageData, 0, 0);
}
```

Progression with different data structures and algorithms:

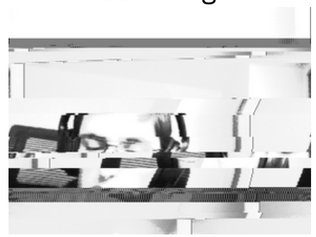
Webserver blocking I/O

Non-blocking I/O (other kind of buffers)

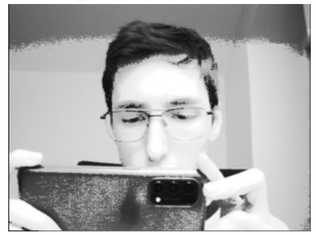
Double buffering non-blocking I/O

WebSocket

WebSocket double buffering



WebSocket with optimized double buffering and max RAM usage



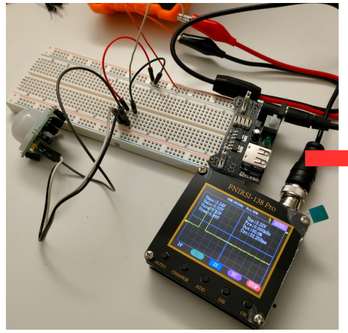
Testing

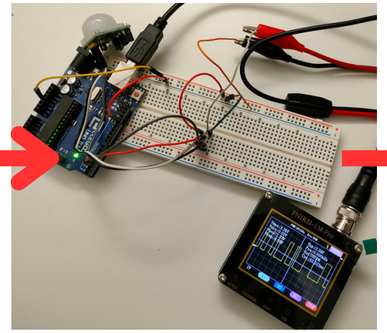
Component testing

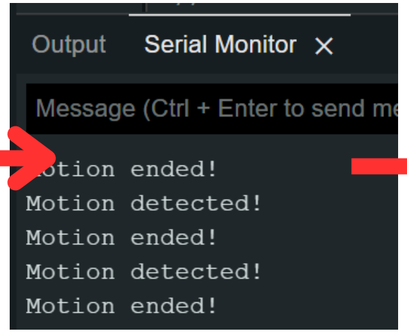
Integration testing

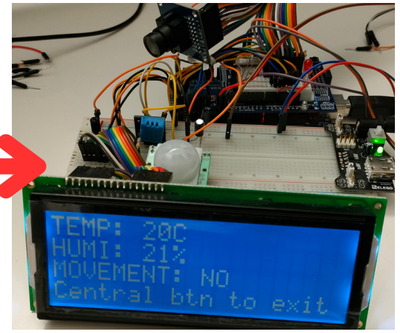
Fast prototyping with Arduino UNO and Arduino IDE

Debugging both via software and with multimeter/oscilloscope










```
G:.
    README
    test1_LCD.c
    test2_OC7670.c
    test3_OV7670_LCD.c
    test4_DHT11_LCD.c
    test5_joystick_menu_LCD.c
    test5_menu.c
    test5_menu.h
    test6_prototype.c

└───Rapid_prototyping
        test_encoder.ino
        test_PIR_sensor_motion.ino
```

Python script to render the image (emulating the D1)

```python
for y in range(HEIGHT):
    for x in range(WIDTH):
        tmp = int.from_bytes(ser.read(), byteorder="big")
        rgb[y][x] = tmp & 0xFF
# Draw the image
for y in range(HEIGHT):
    for x in range(WIDTH):
        r = g = b = rgb[y][x]
        pygame.draw.rect(window, (r, g, b), (x, y, 1, 1))
        pygame.event.clear()
```

C++ program to send an image to the D1 mini (emulating the ATmega2560)

```
    matrix.close();
    file.close();
    return 0;
}

uint8_t rgb[240][320] = {{232, 232, 235, 235, 234,
                          {235, 235, 235, 233, 234,
                          {235, 234, 235, 235, 233,
```

```
typedef enum error_type { GENERAL_ERROR,
                          CAMERA_TWI_ERROR,
                          CAMERA_ERROR,
                          ADC_MUX_SELECTION_ERROR,
                          DHT11_ERROR,
                          UNEXPECTED_STATE_ERROR,
                          UNEXPECTED_JOYSTICK_DIR_ERROR,
                          N_ERRORS } error_type;
```

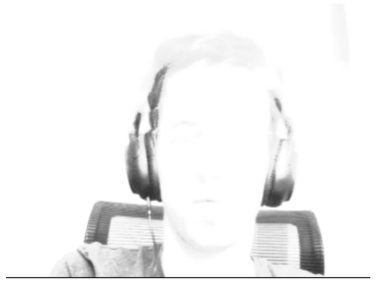
The D1 sends debugging information to UART: WiFi connection, IP address, web server information, WebSocket information, client information, etc.

Conclusions and Future Work

Highly formative experience. I have now a better understanding of how to: design software for MCUs, test and debug developed software, and interact with sensors. I also understood how difficult it is to figure out how to do complex things with such a performance constraint.

The project is not perfect, there are a lot of improvements that can be done.
The most important one is the camera module, which still has some problems:

- Sometimes the first image is overexposed
- Sometimes the data is bugged



Moreover, the D1 mini is not quite fast enough to always send the incoming data in time, sometimes this happens:

The camera software can be improved a lot, there are definitely some mistakes in the values I am pushing on its registers (the ones that control the internal parameters of the camera sensor).

It would be great to make everything interrupt-based.

The ATmega2560 still has a lot of unused pins, and it is not doing anything computationally difficult so it would be interesting to add more sensors.

Add a debugging state to see the information (discussed previously) sent by the D1 to the ATmega2560.
Extend the information the Atmega sends via UART so that it can be used without the LCD and with a web app or a simple app.