Smart home model

1 Abstract

This semester's project focused on designing and building a functional smart home model equipped with a centralized control panel for managing room-specific features such as lighting and PIR sensor. The model demonstrates how modern technology can enhance convenience, energy efficiency, and user interaction within residential spaces. Since this is a first experience with electronic product design for most of the members, we expect that this journey will teach us a lot about electronics and teamwork.

Table of Contents

Table of Figures

2 Introduction

This report encompasses various procedures involved in construction of a smart home model. One of the initial definitions of smart home is provided by Lutolf, defining it as “an integration of different services within a home by using a common communication system.” Moreover, it should assure „an economic, secure, and comfortable operation of the home and includes a high degree of intelligent functionality and flexibility.” (Lutolf, 1992) A fresher definition by Satpathy states, that “a home which is smart enough to assist the inhabitants to live independently and comfortably with the help of technology is termed as smart home. In a smart home, all the mechanical and digital devices are interconnected to form a network, which can communicate with each other and with the user to create an interactive space.” (Satpathy, 2006) We consider this definition more proper, as it emphasizes the role of network in smart home systems, in contrast to the vaguer term ‘common communication system’ used by Lutolf.

According to Zielonka, one of the major development trends related to smart homes are healthcare and quality of life, security in data management and optimal energy management and sustainability. Presumably, the trend of energy management and sustainability is intertwined with recently rising ecological trend (Zielonka, et al., 2021). However, the graph in Figure 1 shows that advanced lightning and intelligent energy management are not among the hottest trends. Nevertheless, we recognize the importance of innovation in this sector, as it is financially and ecologically beneficial.

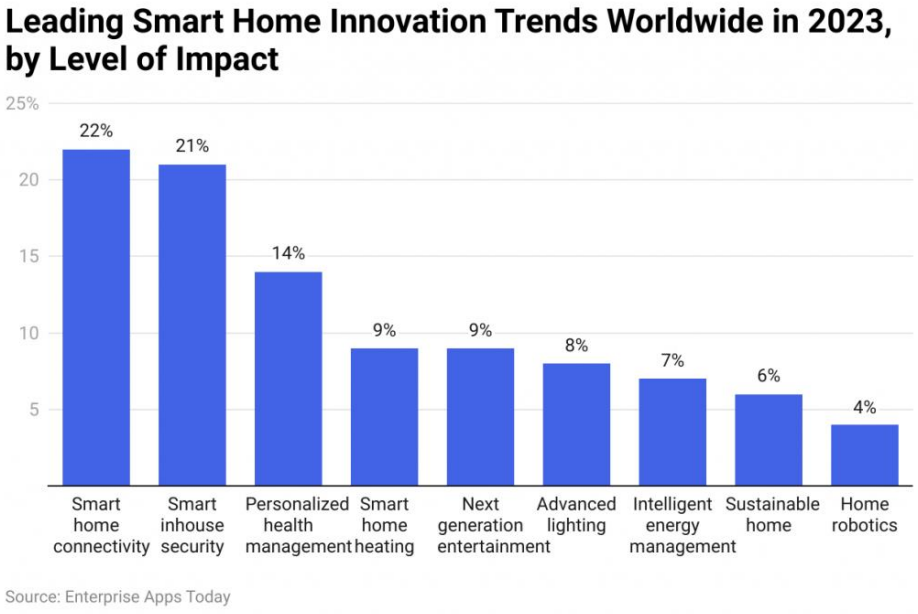


Figure 1 Graph of Leading Smart Home Innovation Trends Worldwide in 2023

Although lights with movement sensors are commonly used nowadays, they could benefit from integration into a centralized system, bringing everything ‘under one roof’. The main advantage of a system with a centralized unit is its flexibility, as users would be able to manipulate light status manually inside the house as well as through communication with the centralized unit, for instance via a control panel. New features could be implemented, for

example allowing the change of various thresholds according to user's needs easily and any time.

However, many challenges emerged during the design process. The most significant problems can be summarized in several 'how to' questions:

1. How to construct the house?
2. How to adjust intensity or color of the light?
3. How to assemble and integrate the circuits?
4. How to structure the control panel menu?

The following questions are answered in subsequent 8 sections. The third chapter includes detailed steps of the design process, where any decision is complemented by justification. The fourth section concerns the final product, possibly deviating from the planned design. The fifth section contains analysis of the results and its comparison with design. Moreover, the sixth section provides a concise conclusion on the whole project. References and appendices are placed in the last two sections.

3 Methods

3.1 Circuit design

3.1.1 Control panel PCB

We chose to have a Control panel, because a characteristic of a smart home is according to, (Alam, Reaz, & Ali, 2012) that it can be controlled remotely from outside the home. The control panel could be developed further to be wireless with Bluetooth and controlled via an app.

For the Control panel, we chose to use buttons to navigate in a menu displayed on the LCD.

A Potentiometer was planned to control the RGB Values and the brightness of the LEDs.

We incorporated 5 buttons UP, DOWN, OK, BACK and a Button for an extra function (sleep mode).

The 4k7 Ω Resistors function as Pull up Resistors for the I2C Module of the LCD.

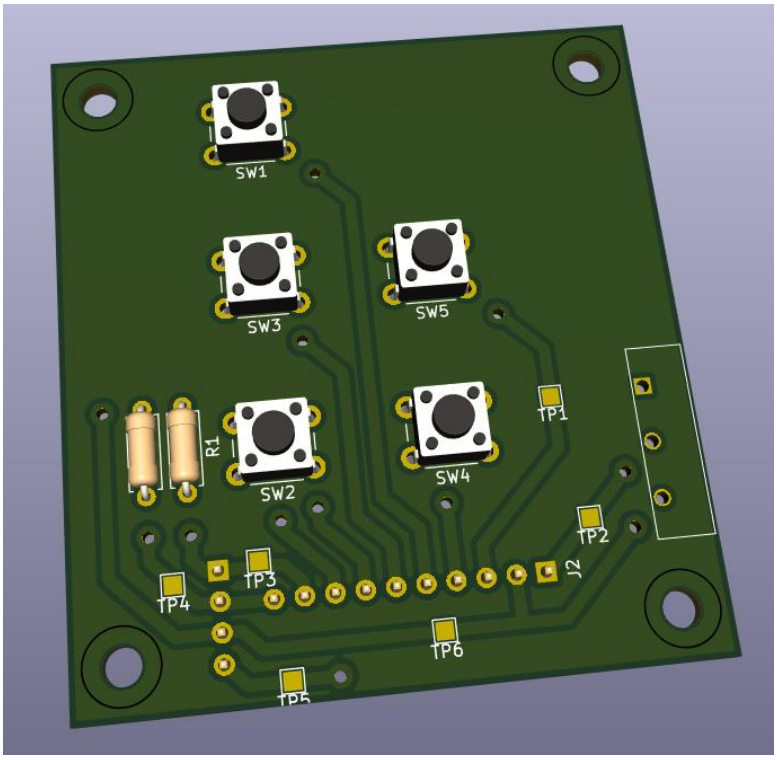


Figure 2 Control panel PCB 3D model

3.1.2 Main PCB

We decided to have the components in and on the house (switch-buttons, LED, Potentiometer, Sensors) connected to the PCB with pin header connectors. So, the PCB is detachable from the House. We Further divided the circuit in two PCB's. One for the Control panel, with the Buttons and Potentiometer mounted on the PCB via Through hole soldering and a pin header connection to the Liquid Crystal Display (LCD).

The other PCB has the transistor driver circuit for the LED's, circuits for the Sensors and the microcontroller on It.

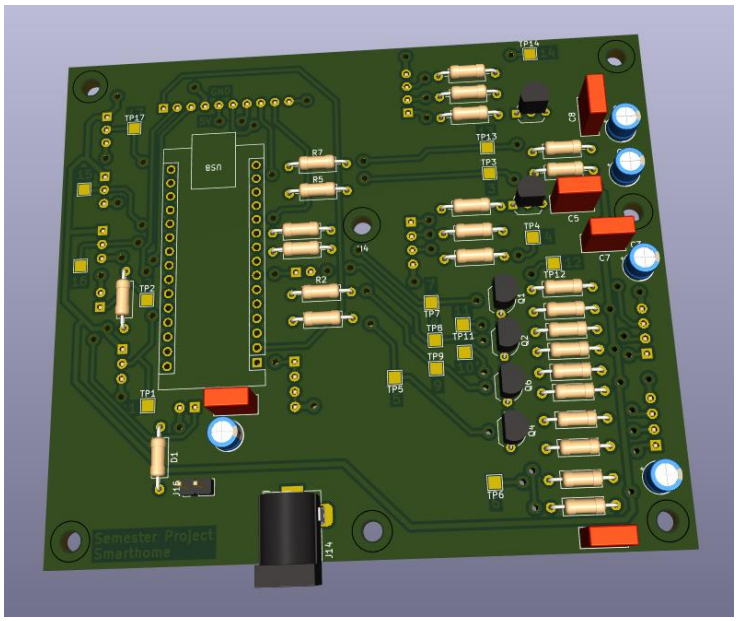


Figure 3 Main PCB 3D model

3.2 3D-Modelling

Design of the house can be divided into two categories, one is base and the other are walls and floor.

Base was designed in Fusion 360 for 3D printer, whole model consists of 230mm X 230mm square with three rooms. We also added mounting holes for walls and floor and space under the floor is empty for routing cables, for that reason we also added openings at the back near the control panel.

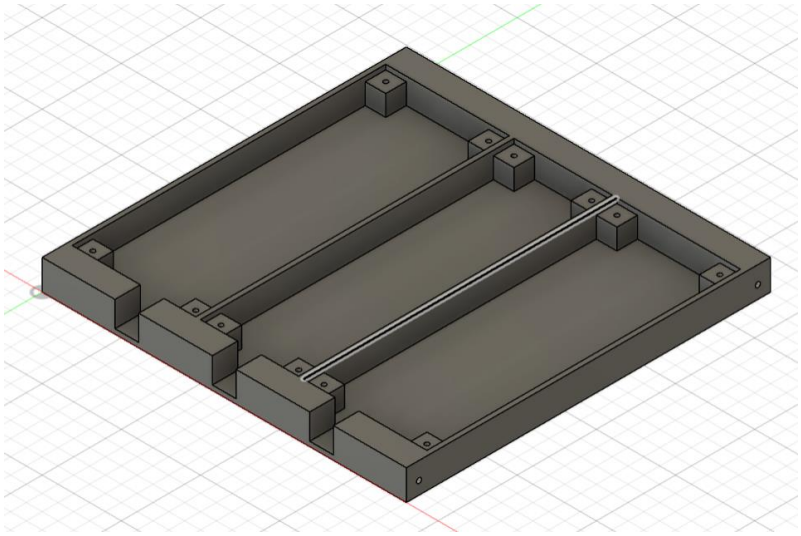


Figure 4 Base 3D model

Walls and floor were designed in Fusion 360 as well, but they were designed for a laser cutter. After laser cutting some parts had to be treated using a rasp so they fit together, because some connections are angled 45 degrees for stronger fit.

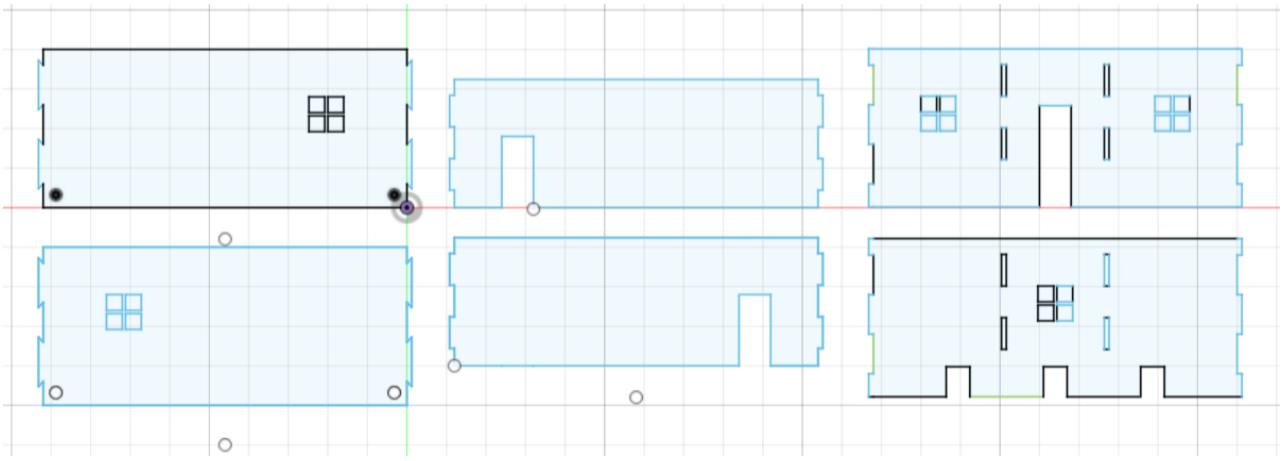


Figure 5 Walls and floor model

For slicing we used Bambu Studio and printed it as well with a Bambu 3D printer.

3.3 Assembly of the Model

After receiving the drilled PCBs, we checked the connections with a multimeter.

Because the PCBs had no connection errors, we continued with the next step: soldering the PCBs.

To have more space underneath the PCB when placing it on the table, we first soldered some of the pin headers to make it stand stable while soldering the vias.

Then we soldered the components starting with the flattest and ending with the tallest.

We reused the female-to-female jumper cables to create the wiring to the components in the house. So, the cables were cut in half and soldered to flex wire around 475 mm in length. Then the flex wires were soldered to the components according to the color-coded wiring table. We used shrinkable tubes to isolate the soldered wire to prevent short circuits. Now with the wiring the entire system and code could be tested.

The Control Panel PCB was mounted with four M3 x 40 mm screws and 20 mm long spacers on to the Mainboard.

Where the LEDs should go, 5mm holes got drilled in the walls for the Lens to go through. To mount the LDR sensor and the PIR sensor, holes were drilled in the outer walls as well.

To arrange the LEDs, buttons and sensors in the house, it was marked where the wires go through the floor. According to that, the floor plates got holes drilled through the wood.

With the components placed on the walls, the female connectors were thread through the holes in the floor, routed at the bottom and lead to the casing of the Control Panel.

The LCD was mounted to the lid of the control panel casing with four M3 x 10 mm screws and nuts.

With all the connections according to the wiring table, the casing of the control panel could be closed and the walls screwed to the bottom with four 3.5 x 25 mm spax screws. To fix the wires on the walls we used electrical tape.

3.4 Problems during assembly

On the Main PCB connection issues and other connection issues were discovered while testing the connections with a multimeter. We fixed the problem with resoldering the visual faulty solder joints.

We realized that D0 was just connected on top layer on the Main PCB Layout, a via was forgotten to be placed in front.

Fortunately, it was not relevant for the main functionality because it was code related:

The Wire for Room 1 Button was swapped with Room 1 Pot, because the IO Pin A6 formerly assigned to Room 1 Button can only be used analog.

The Potentiometer on the control panel was not implemented because it was more convenient to code the user interface with only buttons.

The Pin D1/RX was not able to be used because it interfered with serial communication (RX/TX), even without being connected to a PC. That was probably caused by the USB C connector circuit on the microcontroller.

D1 was originally used for button room 2 but then changed to A2.

Because J5 had just one pin connected to A2, the two wires of the buttons in parallel were soldered to one female connector.

3.5 Introduction to software

Our first idea was to create a library for every room, door and control panel. The library should implement *"Init()"* and *"Update()"* functions which will be called in main function. In addition to those them we also implemented some functions which control the state of the room directly, for example they turn the light on and off. This approach made code more maintainable and readable, therefore we used it in our final build. Also we could very easily divide individual code parts to team members, in our case everyone had to program one room.

```
int main(void)
{
  init_room_1();
  init_room_2();
  init_room_3();
  init_door();

  init_control_panel();

  while(1)
  {
    update_room_1();
    update_room_2();
    update_room_3();
    update_door();

    update_control_panel();
  }
}
```

Figure 6 Main program

We had to come up with a solution to how buttons should operate in rooms where there is the possibility to change brightness or color of the light. After a few discussions we stuck to idea last action takes effect. This means that if brightness was set to half of its potential by clicking the button the light is turned off (light is set to maximum brightness after clicking button, only when brightness is zero), even though the potentiometer's value stays unchanged. This solution turned out to be very handy when we started working on control panel.

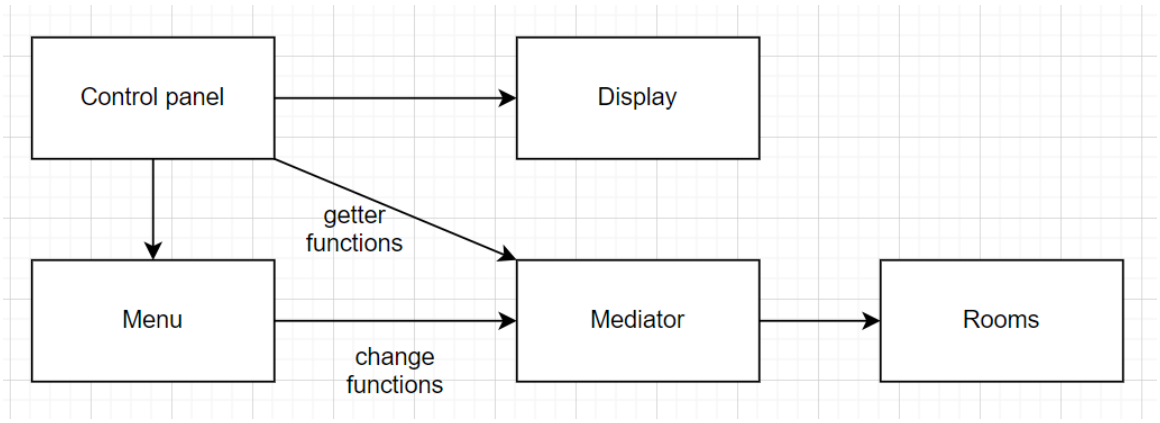


Figure 7 Software diagram

When a button is pressed the change from not being pressed and being pressed isn't immediate, there are always some fluctuations which are result of mechanical construction of a button. We solved this issue by adding delay every time a button is pressed, this resulted in functions for every button which check status of a button and if it is pressed they start 100 ms delay.

```
int check_btn_room_1()
{
    if ((PINC & (1 << BTN_PIN)) == 0)
    {
        _delay_ms(50);
        return 1;
    }
    return 0;
}
```

Figure 8 Check button function

3.6 Room 1 software

Room 1 has one button for turning the LED on and off and one potentiometer for controlling the brightness of the LED. We implemented functions for controlling and retrieving information about the LED's status and brightness.

```
#ifndef ROOM_1_H_INCLUDED
#define ROOM_1_H_INCLUDED

void init_room_1();
void update_room_1();

void turn_on_light_room_1();
void turn_off_light_room_1();
void switch_light_room_1();
void set_light_intensity_room_1(unsigned char intensity);
int get_light_intensity_room_1();

#endif
```

Figure 9 Room1.h

In *"Init()"* we initialized ADC and PWM using Alin's libraries, and also set button pin as input with pull up resistor, to do that we referred to AtMega datasheet (Atmel Corporation, 2015) and Arduino Nano pinout (Arduino cc, 2021). These documents we used for other rooms and control panel as well.

"Update()" function checks if potentiometer values changed since last measurement, if yes update LED's intensity accordingly. Then it checks if button is pressed and wasn't pressed previously, if this statement is true then the status of LED is inverted.

```
void update_room_1()
{
    int current_pot_value = read_pot_room_1();

    if (current_pot_value > previous_pot_value + 3 || current_pot_value < previous_pot_value - 3)
    {
        set_light_intensity_room_1(current_pot_value);
        previous_pot_value = current_pot_value;
    }

    if (check_btn_room_1())
    {
        if (!was_btn_pressed_previously)
        {
            was_btn_pressed_previously = 1;A
            switch_light_room_1();
        }
    }
    else
    {
        if (was_btn_pressed_previously)
        {
            was_btn_pressed_previously = 0;
        }
    }
}
```

Figure 10 Update room 1 function

3.7 Room 2 software

This is the simplest room it has only two buttons and one LED. Since we wired buttons in parallel, from the Arduino perspective they act as a one button. Again, we added functions for getting and setting status of the LED. Logic in *"Update()"* and *"Init()"* is very similar to Room 1, just without potentiometer and LED is controlled using registers not using the PWM library.

```
void update_room_2()
{
    int button_pressed = check_button_room_2(); // read button

    if (button_pressed && previous_button_state == 0)
    {
        switch_light_room_2();
    }

    previous_button_state = button_pressed;
}
```

Figure 11 Update room 2 function

3.8 Room 3 software

Room 3 contains one potentiometer for color changing, a button and one LED. We decided that potentiometer will affect only one property of the light for example red or green.

We were aware of the limitations of this approach and decided to implement configuration of a potentiometer's-controlled value into control panel, in final version it is possible to choose which part of the light's color is controlled by a potentiometer. We also added HSV (Atkins, 2010) color model because it is much easier to work with for people. So, you can configure red, green, blue, hue, saturation, value whilst red is default.

```
enum PotControlledValue
{
    R,
    G,
    B,
    H,
    S,
    V
} pot_1_controlled_val_room_3;

void init_room_3();
void update_room_3();

int check_button_room_3();
void turn_on_light_room_3();
void turn_off_light_room_3();
void switch_light_room_3();
int get_status_of_light_room_3();

ColorRGB get_color_of_light_RGB_room_3();
ColorHSV get_color_of_light_HSV_room_3();
void set_color_of_light_RGB_room_3(ColorRGB);
void set_color_of_light_HSV_room_3(ColorHSV);
```

Figure 12 Room3.h

Potentiometer is read the same way as in the room 2 and LED is controlled using PWM 3 from Alin's library, turning on and off using the switches between full white and completely off.

3.9 Door software

Door's LED is controlled by PIR sensor and LDR, therefore there is no button. In addition to *"Init()"* and *"Update()"* functions we implemented functions for detecting movement, getting current value of LDR and setting threshold for LDR.

In *"Init()"* function we just initialized ADC using Alin's and pin for LED as output and pin for PIR sensor as input using registers.

Every time the movement is detected and LDR has higher value than threshold LED is turned on and timer is reset. Timer is just one variable which is incremented by one every time the *"Update()"* function is called, when the LED is turned on the variable is set to zero, when this variable reaches particular value LED is turned off.

```
void update_door()
{
    light_timer++;
    int current_ldr = get_ldr_value_door();
    int movement = movement_detected_door();

    if (current_ldr > ldr_threshold)
    {
        if (movement)
        {
            turn_on_light_door();
        }
    }

    if (light_length < light_timer)
    {
        turn_off_light_door();
    }
}
```

Figure 13 Update door function

3.10 Control panel software

Even though it is not a room, it works very similarly to one, because buttons function the same as in the rooms and control panel and background logic is just a more complicated version of a LED.

```
void init_control_panel()
{
    init_display();
    init_menu();
    //adc_init();

    // set input and pull up for button up
    DDRC &= ~(1 << BTN_UP);
    PORTC |= (1 << BTN_UP);

    // set input and pull up for button down
    DDRC &= ~(1 << BTN_DOWN);
    PORTC |= (1 << BTN_DOWN);

    // set input and pull up for button select
    DDRC &= ~(1 << BTN_SELECT);
    PORTC |= (1 << BTN_SELECT);

    // set input and pull up for button back
    DDRC &= ~(1 << BTN_BACK);
    PORTC |= (1 << BTN_BACK);

    show_curr_elems_on_display();
}
```

Figure 14 Init control panel function

In addition to physical components, which make up control panel, we had to implement back-end logic for the menu responsible for storing and navigating in menu elements, we also implemented some type of a link or connection between control panel and rooms, we called it mediator.

```
void update_control_panel()

    Menu_element* curr_elems[4];
    int element_count;

    get_current_elms(curr_elems, &element_count);

    for (int i = 0; i < element_count; i++)
    {
        if (curr_elems[i]->type == MENU_FILE)
        {
            get_value_func get_value = get_getter_func(curr_elems[i]->element.file.ID);
            if (get_value != 0)
            {
                float new_value = get_value();

                if (new_value != curr_elems[i]->element.file.value)
                {
                    curr_elems[i]->element.file.value = new_value;
                    update_file_element(curr_elems[i]);
                }
            }
        }
    }
}
```

Figure 15 Part of the update control panel function

Mediator's responsibilities are to retrieve and change information for room associated with menu element for example for element showing status of LED in room 1, mediator will return status of led in room 1 and element will be updated, when value in menu element is changed mediator ensures that room is updated as well.

3.10.1 Menu implementation

When we were designing a menu, we took inspiration from files and folders in a computer, every element in a menu can be file or folder.

Folder contains more folders or files, for example at the top level of the menu there are four folders (room 1, room 2, room 3, door) after selecting one of the folders its content will be displayed on control panel. Returning back to previous folder is done by clicking “back” button.

Files hold values which can be linked to room or it can be some arbitrary value which affects other files for example file “Color format” can have value “RGB” or “HSV” (Atkins, 2010) and this value affect how is color of the light in room 3 represented on control panel. After selecting file its value can be changed by clicking “up” and “down” buttons, changes take effect instantaneously. File can be deselected by clicking “back” button.

Files and Folders are contained in “Menu_element” struct (W3Schools, C structs, n.d.) this struct has a variable of enumeration type (W3Schools, C enums, n.d.) to determine if current menu element is file or folder, after this variable is union (TutorialsPoint, n.d.) with file or folder struct in it. The only differences between folder and file structure are that file has also value, type of data stored in it and variable which determines if it is selected.

```
typedef struct {
    int ID;
    char* Name;
}  Folder;

typedef struct {
    int ID;
    char* Name;
    float value;
    type_of_file type;
    bool selected;
} File;

typedef struct {
    menu_element_type type;

    union {
        Folder folder;
        File file;
    } element;

} Menu_element;
```

Figure 16 Menu structs

Our first idea for storing menu elements was to use multidimensional array, but we quickly realized that it would be difficult to work with it, because we can’t determine the size of an array in C, and we risk going out of bounds of it. For that reason, we stored all the elements in one dimensional array and location of the element we encoded in its “ID”.

ID acts as a path to the element for example after clicking second folder on first level first element on second level will have ID equals to 12, path is inverted (first level is last digit and last level is first digit) so it is easier to search in array of elements.

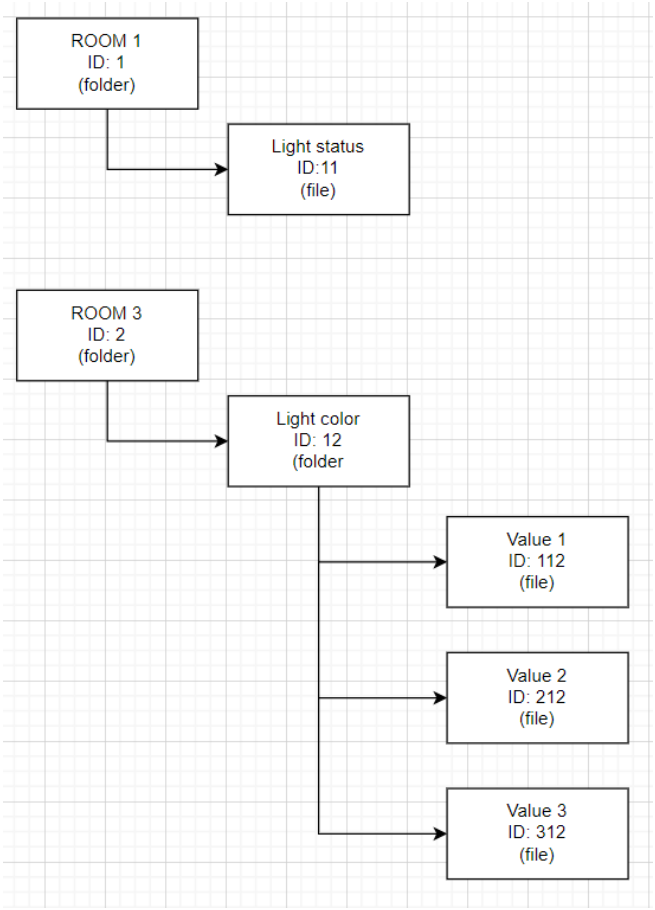


Figure 17 Menu diagram

3.10.2 Display implementation

To simplify the use of display we created helper library, which has functions for updating whole display with menu elements, which are passed to the function as a parameter, and for updating one element, so display is not refreshed every time a value in one element is changed only that element is refreshed.

```
void init_display();

void update_display(Menu_element** elems, int elements_count, int element_hovered, int element_selected);
//if no elment is hovered element_hovered=-1
//if no element is selected element_selected=-1
//one of variables(hovered,selected) will always be -1

void update_file_element(Menu_element* elem);
```

Figure 18 Display.h

I addition to displaying current menu elements, display also mark hovered element with "*" character and selected element with ">" character.

3.10.3 Mediator implementation

As it was mentioned in previous chapters to update rooms using control panel we are using Mediator. Mediator's API consists of two functions which return pointer to a function one returns a function for changing value and the second one returns a function for getting a value. Which function to return is determined by menu element ID, functions aren't stored in an array

pointer to a particular function is returned using switch statement (W3Schools, C switch, n.d.) which switches between different IDs.

```
typedef float (*get_value_func)(void);
typedef void (*change_value_by_func)(int); // change value by n points

// get pointers to functions using element id
get_value_func get_getter_func(int id);
change_value_by_func get_changer_func(int id);
```

Figure 19 Mediator.h

```
get_value_func get_getter_func(int id)
{
    switch (id)
    {
        case 11:
            return get_room1_led_brightness;
        case 21:
            return get_room1_led_status;
        case 12:
            return get_room2_led_status;
        case 13:
            return get_room3_led_status;
        case 14:
            return get_door_ldr_value;
        case 24:
            return get_door_movement;
        case 34:
            return get_ldr_threshold;
        case 44:
            return get_door_led_status;
        case 123:
            return get_room3_led_value1;
        case 223:
            return get_room3_led_value2;
        case 323:
            return get_room3_led_value3;
        case 423:
            return get_room3_color_format;
        case 133:
            return get_room3_pot1_controlled_val;
        default:
            return 0;
    }
}
```

Figure 20 get_getter_func function

3.11 Software problems

When debugging the whole assembly we encountered some issues, those problems were mostly logical and easily fixable. However, we weren't able to fix problems with altering the color of light in room 3 using control panel when color format was set to HSV (Atkins, 2010), therefore this feature is not available and can be changed using control panel only when color format is set to RGB.

Another inconvenience which made its way to the final product is unreliable control of the status of light in door, which means that door light can be controlled by control panel, but it does not work every time, we couldn’t figure out why it happens.

4 Results

In hindsight we can say that the project went well, even though we found ourselves in a hurry at the end, due to unexpected problems and difficulties.

Our main focus when designing smart home was on connectivity and ease of controlling individual rooms. We achieved this by incorporating a control panel whilst perceiving ability to control rooms in place. When working on the control panel we focused on making it as intuitive as possible, therefore we used the approach the last action takes effect. This way you don’t have to worry whether the light in a room is turned on or off because you can always set it in a room or in the control panel.

When designing the hardware part of the project we thought about how to route cables a lot, we wanted simple solution, which allows us to make mistakes when routing in terms of length of cables. By making the floor hollow we were able to route all the cables easily while having space for very long cables. Since all the cables are in the floor, we were able to use laser cutter to cut walls which was faster and easier way to manufacture walls.

When it came to software, we didn’t want to make it everything in one file and somehow make it work, because this way the code would become unreadable very quickly and possible problems would be very difficult to solve. Therefore, we created a library for every room and in the main function we just initialized and updated the room. In addition to room libraries, we created some helper libraries for the menu, display and mediator. As a result we had a very clean and maintainable code base, which made it much easier to find and fix bugs.

5 Discussion

This section contains explained differences between results and design, limitation of the project and suggestions for further development.

5.1 Design and results comparison

Overall, the design functioned as expected. Nevertheless, some features had to be abandoned, as their functionality was not required. Notably, the use of RX/TX or D0/D1 pins turned out to be complicated. Their behavior is different in comparison to standard digital pins, which was found out later in the project. Therefore, they are not used in the final version of the project. In addition, the button for additional features, initially intended as a switch to turn on or off sleep mechanism, could not have been implemented due to a design flaw. It is described in section 3.4 Problems during assembly.

What's more, it was decided that control panel potentiometer will not be used. It was intended as a tool to change brightness or color component directly on the control panel, however, such implementation proved to be more complex than the implementation of this function to the control panel buttons. Another neglected feature is the ability to change color components in HSV format, in Room 3 via the control panel. Presumably, the adjustment of the color values was faulty due to unknown causes likely stemming from coding errors. The displayed format of the color in Room 3 menu can still be changed, but the ability to adjust color values is solely available in RGB format.

5.2 Technical limitations

Moreover, some of our decisions were not optimal due to a few restrictions. One of the restrictions were mandatory requirements of the project, namely the use of Arduino Nano microcontroller and LCD display provided by the SDU. However, these requirements weren't and shouldn't be considered as true restrictions within the context of the project. The aim of this remark is to acknowledge these requirements, as they fundamentally influenced the development process. In addition, other confinements were introduced by electrical component provider and PCB manufacturer. The choice of electrical components was restricted by the limited variety of component models sold by the provider, while the design of PCBs had to align with available services of PCB manufacturer.

5.3 Retrospective

Furthermore, the dynamic of a newly formed project group affected the whole development process. The project plan design was unrealistic, as the capabilities of the members didn't match the task skill/knowledge requirements most of the time as well as the pace proposed by the plan was too high. However, due to the high frequency of group meetings, there was consistent work done each week and any problems could be resolved quicker. Arguably, a thorough plan was not required to efficiently finish the project due to a few factors. For example, members were new to the concept of working in group on such a technical project, members weren't familiar with all the aspects of the development process, but most importantly, cumulative acquisition of knowledge didn't allow smoother completion of tasks during the whole semester. Nevertheless, a group would benefit from more frequent assignment of deadlines to the tasks emerging during the meetings. Notably, this section is not objective and

is written from the writer's perspective. The opinions of members on this topic might significantly differ.

6 Conclusion

To conclude, a couple of insignificant features included in the design weren't implemented due to various reasons. Various sources of technical limitations were mentioned. The group dynamic is brought up as a factor influencing the development process, while the requirement of a strict project plan is challenged.

On the other hand, after successfully completing our smart home model project we know a lot about team work, development timeline and task division, we also learned how to be responsible and complete tasks, which were assigned to us, because this wasn't personal project and our discipline could decide between success or a failure. In addition to soft skills we also learned how to navigate complex world of product design and technical documentation. This knowledge will prove useful in future projects.

7 References

Alam, M. R., Reaz, M. B., & Ali, M. A. (2012). *A Review of Smart Homes—Past, Present, and Future.* IEEE.

Arduino cc. (2021). *Arduino Nano pinout.* Arduino cc.

Atkins, R. (2010, December). *RGBConverter*. Retrieved from GitHub: https://github.com/ratkins/RGBConverter

Atmel Corporation. (2015). *ATmega328P datasheet.* San Jose: Atmel Corporation.

Lutolf, R. (1992). Smart Home concept and the integration of energy meters into a home based system. *Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply 1992* (pp. 277-278). Glasgow: IET.

Satpathy, L. (2006, 5 8). *Smart Housing: Technology to Aid Aging in Place - New Opportunities and Challenges.* Retrieved from Thesis and Dissertations: https://scholarsjunction.msstate.edu/td/3967/

Schratz, M., Christine, G., Struhs, T. J., & Gray, K. (2016). A New Way to See the Light: Improving Light Quality with Cost-Effective LED Technology. *IEEE Industry Applications Magazine*, 55-62.

TutorialsPoint. (n.d.). *C unions*. Retrieved from TutorialsPoint: https://www.tutorialspoint.com/cprogramming/c_unions.htm

W3Schools. (n.d.). *C enums*. Retrieved from W3Schools: https://www.w3schools.com/c/c_enums.php

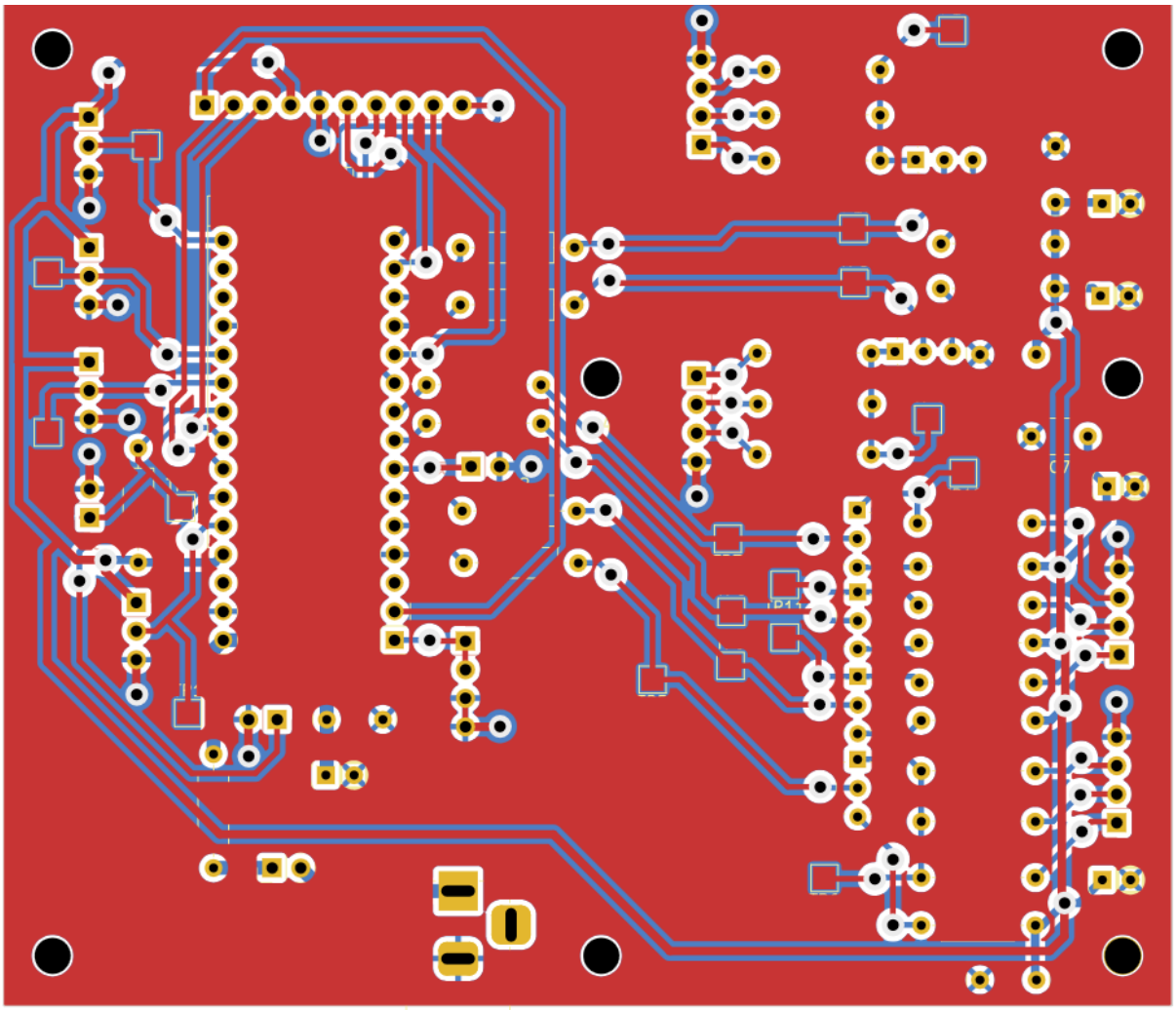
W3Schools. (n.d.). *C structs*. Retrieved from W3Schools: https://www.w3schools.com/c/c_structs.php

W3Schools. (n.d.). *C switch*. Retrieved from W3Schools: https://www.w3schools.com/c/c_switch.php

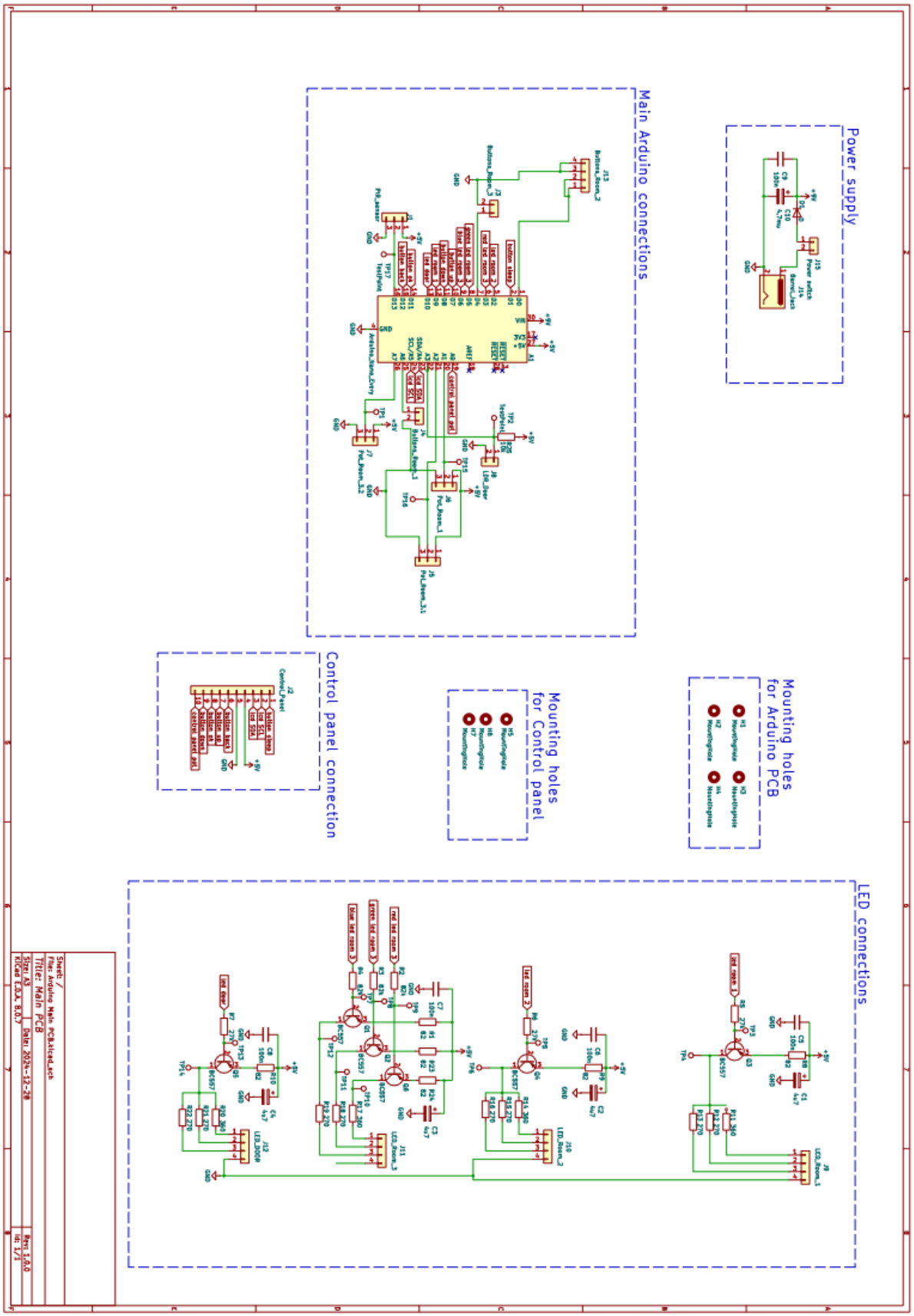
Zielonka, A., Woźniak, M., Garg, S., Kaddoum, G., Piran, J., & Ghulam, M. (2021). Smart Homes: How Much Will They Support Us? A Research on Recent Trends and Advances. *IEEE Access*, 26388-26419.

8 Appendices

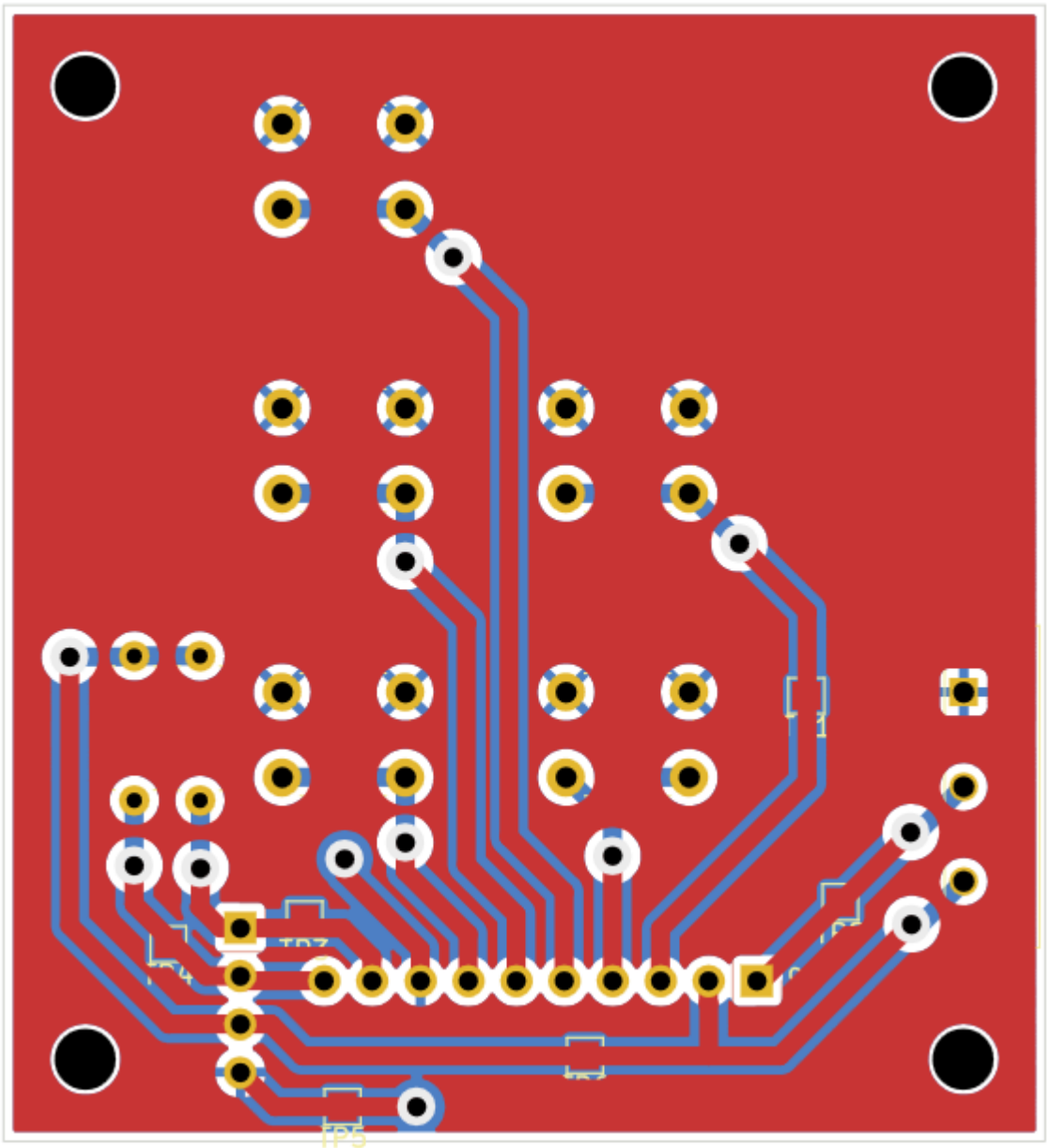
8.1 Main PCB



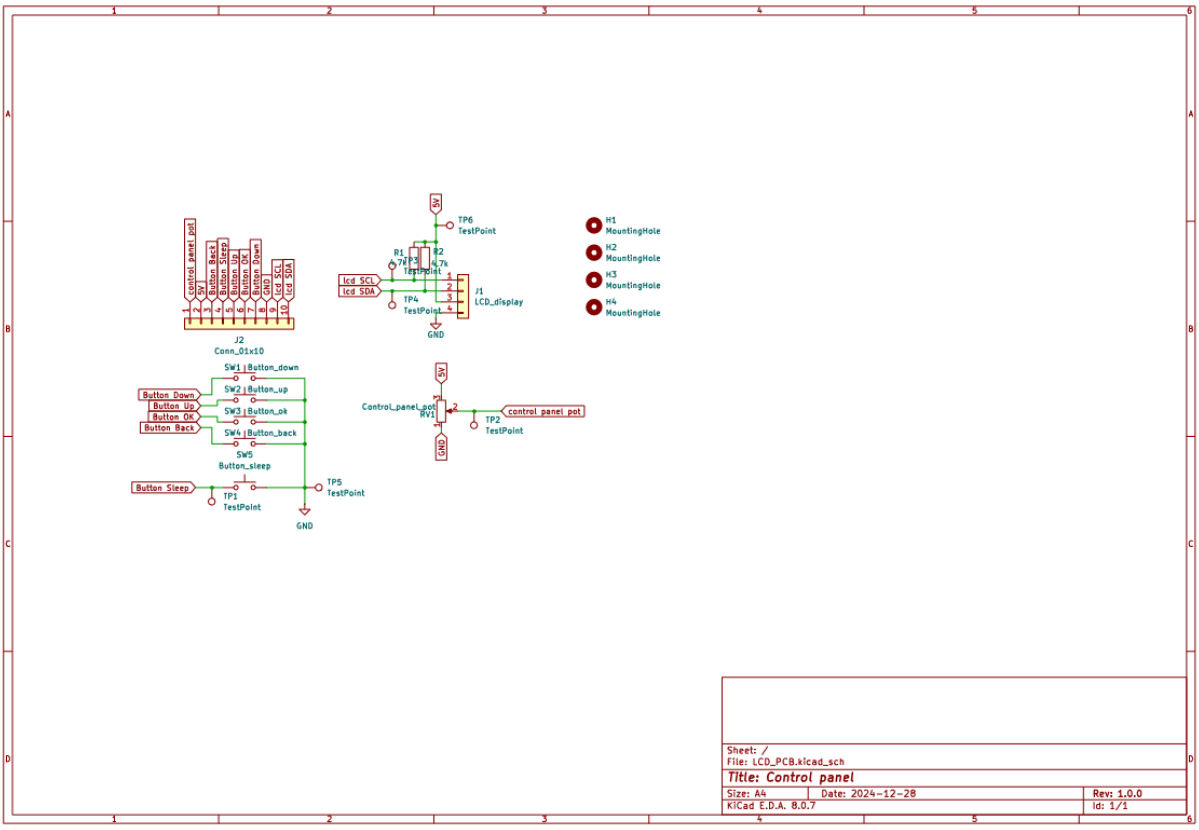
8.2 Main board schematic



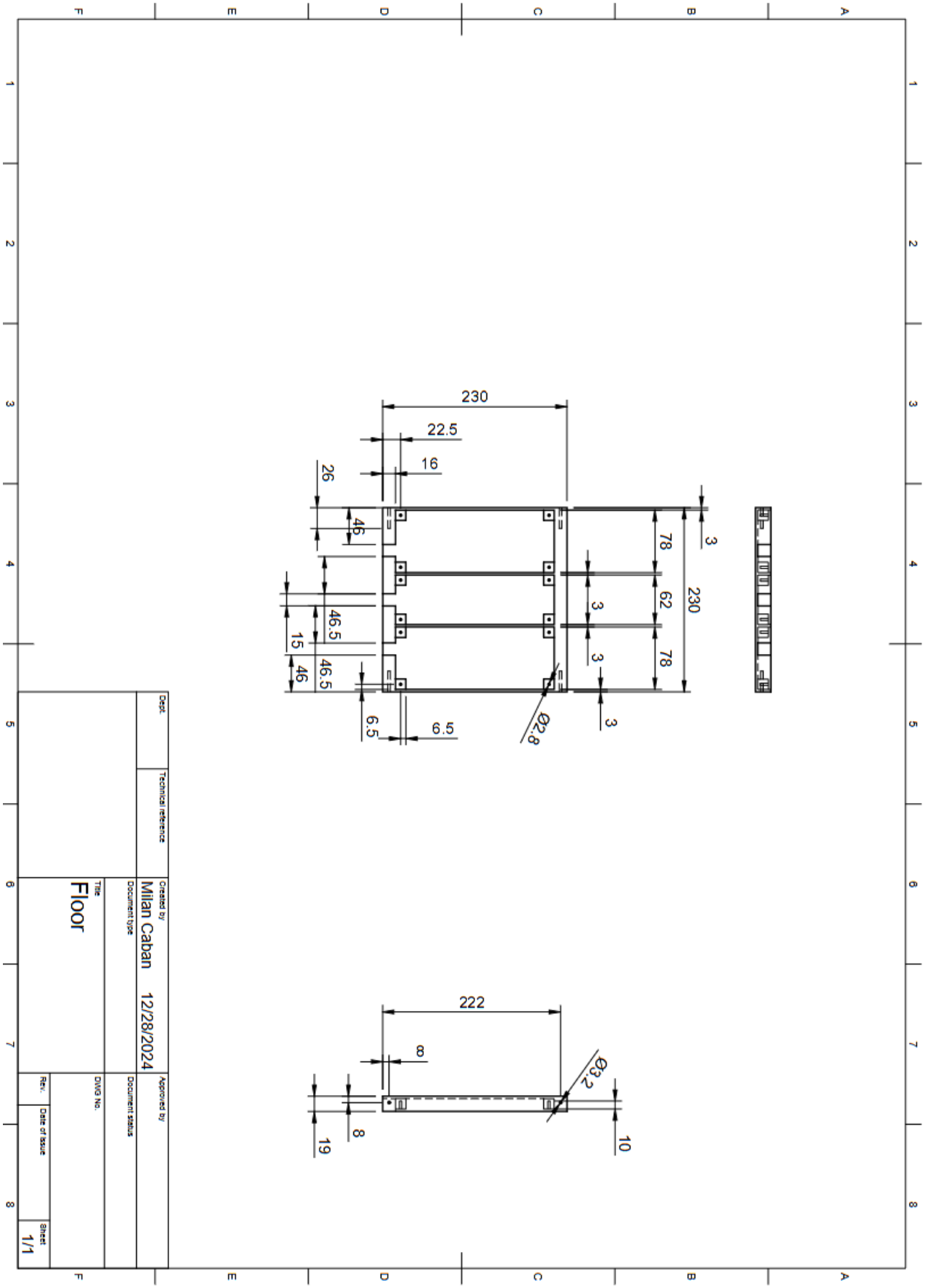
8.3 Control panel PCB



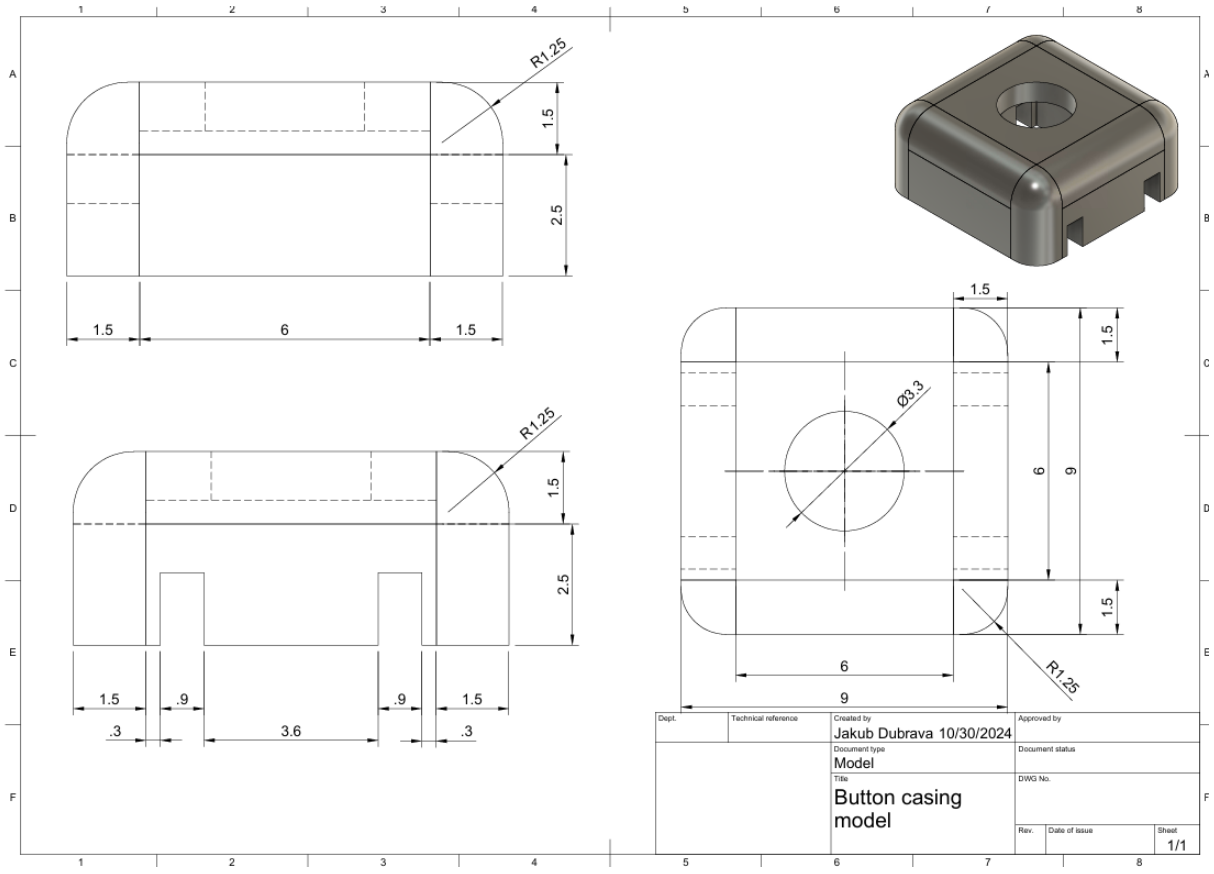
8.4 Control panel board schematic



8.5 Base drawing



8.6 Button casing drawing



8.7 Calculations

$$I_c = -10\ mA$$

$$v_{CE-sat} = -75\ mV$$

$$h_{FE-min} = 180$$

$$I_B = \frac{I_C}{h_{FE-min}} = -\frac{0.01}{180} = -55\ \mu A$$

$$R_{bb} = \frac{v_{cc} - v_{BE}}{I_B} = \frac{-5 - (-0.7)}{-5.5 \cdot 10^{-5}} = 78.2\ k\Omega$$

$$v_{Resistor} = v_{cc} - v_{CE-sat} - v_{LED}$$

$$v_{Red\ LED} = -1.85\ V$$

$$v_{Green\ LED} = -3\ V$$

$$v_{Blue\ LED} = -3\ V$$

$$v_{Red} = v_{cc} - v_{CE-sat} - v_{Red\ LED} = -5 - (-0.075) - (-1.85) = -3.075\ V$$

$$v_{Green} = v_{cc} - v_{CE-sat} - v_{Green\ LED} = -5 - (-0.075) - (-3) = -1.925\ V$$

$$v_{Blue} = v_{cc} - v_{CE-sat} - v_{Blue\ LED} = -5 - (-0.075) - (-3) = -1.925\ V$$

$If\ R_{bb} = 82\ k\Omega$:

$$I_B = \frac{v_{cc} - v_{BE}}{R_{bb}} = \frac{-5 - (-0.7)}{82000} = -52.4\ \mu A$$

$$I_C = I_B h_{FE-min} = -5.24 \cdot 10^{-5} \cdot 180 = -9.44\ mA$$

$$R_{LED} = \frac{v_{Resistor}}{I_C}$$

$$R_{Red} = \frac{v_{Red}}{I_C} = \frac{-3.075}{-9.44 \cdot 10^{-3}} = 326\ \Omega$$

$$R_{Green} = \frac{v_{Green}}{I_C} = \frac{-1.925}{-9.44 \cdot 10^{-3}} = 204\ \Omega$$

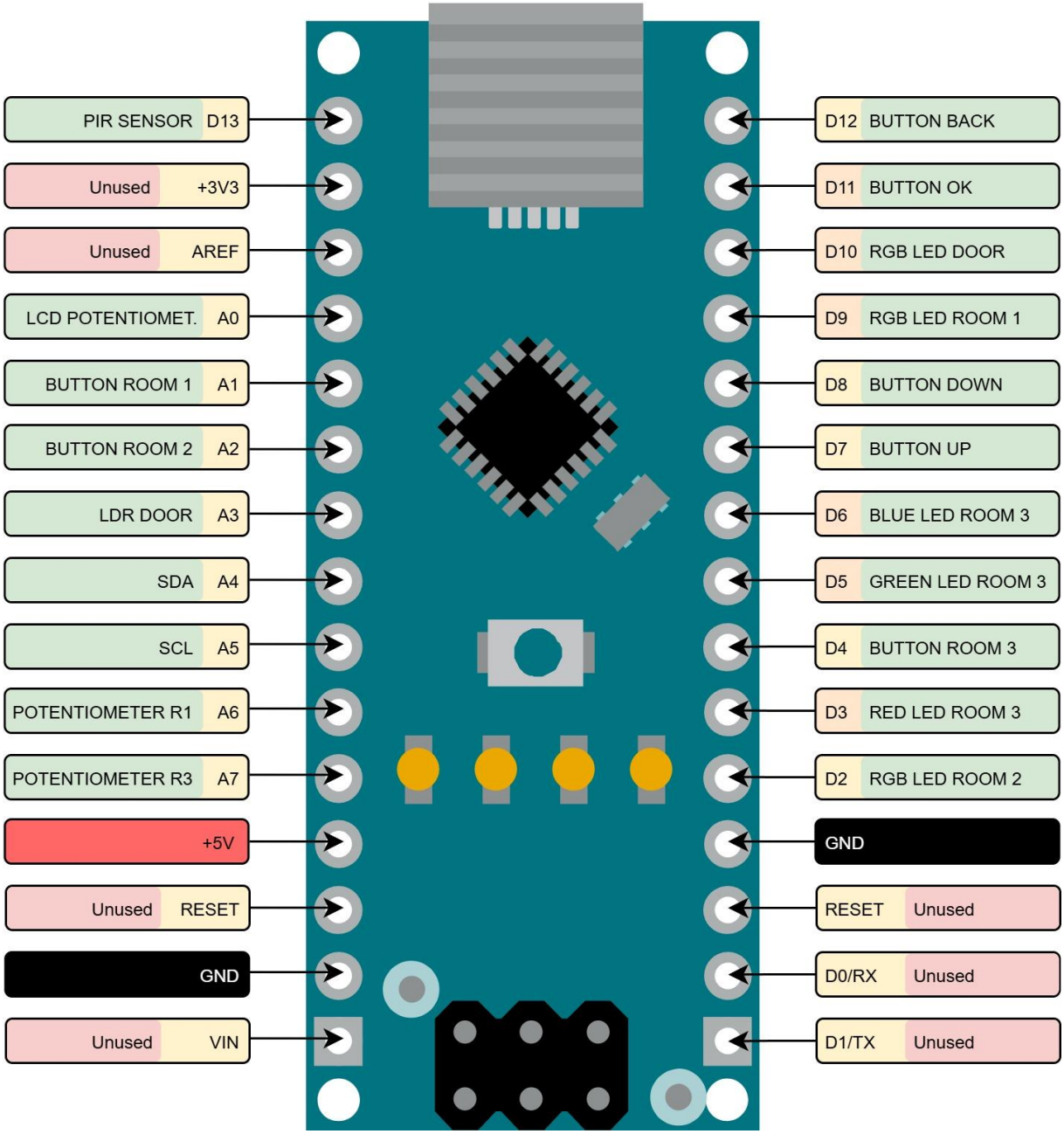
$$R_{Blue} = \frac{v_{Blue}}{I_C} = \frac{-1.925}{-9.44 \cdot 10^{-3}} = 204\ \Omega$$

$If\ I_C = 30\ mA$:

$$I_B = \frac{I_C}{h_{FE-min}} = -\frac{0.03}{180} = -\frac{1}{6000} A = 166.67\ \mu A$$

$$R_{bb} = \frac{v_{cc} - v_{BE}}{I_B} = \frac{-5 - (-0.7)}{-\frac{1}{6000}} = 25.8\ k\Omega$$

8.8 Pinout


PIR SENSOR D13
Unused +3V3
Unused AREF
LCD POTENTIOMET. A0
BUTTON ROOM 1 A1
BUTTON ROOM 2 A2
LDR DOOR A3
SDA A4
SCL A5
POTENTIOMETER R1 A6
POTENTIOMETER R3 A7
+5V
Unused RESET
GND
Unused VIN
D12 BUTTON BACK
D11 BUTTON OK
D10 RGB LED DOOR
D9 RGB LED ROOM 1
D8 BUTTON DOWN
D7 BUTTON UP
D6 BLUE LED ROOM 3
D5 GREEN LED ROOM 3
D4 BUTTON ROOM 3
D3 RED LED ROOM 3
D2 RGB LED ROOM 2
GND
RESET Unused
D0/RX Unused
D1/TX Unused

8.9 Wiring table for pin header connectors

Wire Color	Component / Pin	Microcontroller pin	Pin Header Connector
Blue	LED Room 1 / GND	GND	J9 / 4.
Red	LED Room 1 / Red	D9	J9 / 1.
Blue	LED Room 2 / GND	GND	J10 / 4.
Red	LED Room 2 / Red	D2	J10 / 1.
Blue	LED Room 3 / GND	GND	J11 / 4.
Red	LED Room 3 / Red	D3	J11 / 1.
Green	LED Room 3 / Green	D5	J11 / 2.
Orange	LED Room 3 / Blue	D6	J11 / 3.
Yellow	Button Room 2 / 1.	A2	J5 / 2.
Yellow	Button Room 2 / 1. GND	GND	J13 / 3.
Black	Button Room 2 / 2. GND	GND	J13 / 4.
Brown	Button Room 3	D4	J3 / 1. + 2.
Gray	LDR	A3	J8 / 1. + 2.
Yellow	Pot Room 1 / 2.	A6	J4 / 1.
Orange	Button Room 1 / 1.	A1	J6 / 2.
Orange	Button Room 1 / GND	GND	J4 / 2.
Yellow	Pot Room 3 / 2.	A7	J7 / 2.
Blue	Pot Room 3 / GND	GND	J7 / 3.
Blue	LED Door / GND	GND	J12 / 4.
Red	LED Door / Red	D10	J12 / 1.
Blue	PIR / GND	GND	J1 / 3.
Green	PIR / 5V	5V	J1 / 1.

8.10 Wiring table for connecting Control panel to main PCB

Function	Wire Color	Connector on Control Panel PCB	Connector on Main PCB	Microcontroller Pin
Control panel pot	White	1	10	Not implemented
Button Down	Gray	7	9	D8
Button ok	Green	6	8	D11
Button up	Purple	5	7	D7
Button back	Black	3	6	D12
GND	Blue	8	5	GND
5V	Red	2	4	5V
LCD SDA	Brown	10	3	A4
LCD SCL	Orange	9	2	A5
Button sleep		4	1	D0/RX (Not connected or used)