



Aalto-yliopisto
Sähkötekniikan
korkeakoulu

ELEC-C5070 – Elektroniikkapaja

Loppuraportti IoT-sarastusvalo

Syksy 2021 - Ryhmä numero 16

Justus Ojala
(770084, justus.ojala@aalto.fi, puhelin 044-973 1700)

1 JOHDANTO

Työn tavoitteena oli rakentaa puhelimeen liittyvä sarastusvalo, käytännössä siis herätskello, johon liitettävä valo kirkastuu hiljalleen ennen herätystä. Taustalla tähän on työn tekijän torkutustaipumus.

Ajatus on, että puhelimen herätskello voi halutessa lähetä ennen herätystä valolle signaalin, joka saa sen kirkastumaan toivotusti. Valon tulisi toimia samalla tavallisena kirkkaussäädetävänä valona.

2 TOTEUTETTU LAITTEISTO

Valo on rakennettu tavallisesta pöytälampusta korvaamalla sen teholelektronikka itse tehdyllä ESP32-ohjatulla. Prototyppissä teholelektronikka on itse valaisimen ulkopuolella, ja sisäinen elektronikka on ohitettu. Valaisinelementti on kytketty suoraan tehonsyöttöön, ja uusi elektronikka tulee teholähteestä ja lampun syötön väliin.

Elektroniikan prototyppi on rakennettu protolevyyn kytketyn ESP32-kehitysalustan ympärille. Se ottaa syötön lampun teholähteeltä ja jatkaa sen eteenpäin vaihdettavan esivastuksen ja valaisinelementin maapuolelle kytketyn hakkurina toimivan MOSFETin läpi. ESP32 saa käyttöjännitteensä lineaariregulaattorilta, joka puodottaa jännitteen 11 V -> 5 V.

ESP32 on kytketty mikrokytkimeen, potentiometrin keskikytkentään ja MOSFETin hilaan. Se lukee mikrokytkintä digitaalisesti sisäisen pulldown-vastuksen avulla (kytkin kytkee anturipinnin +5V) ja potentiometriä analogisesti. Transistorille se syöttää PWM-signaalia.

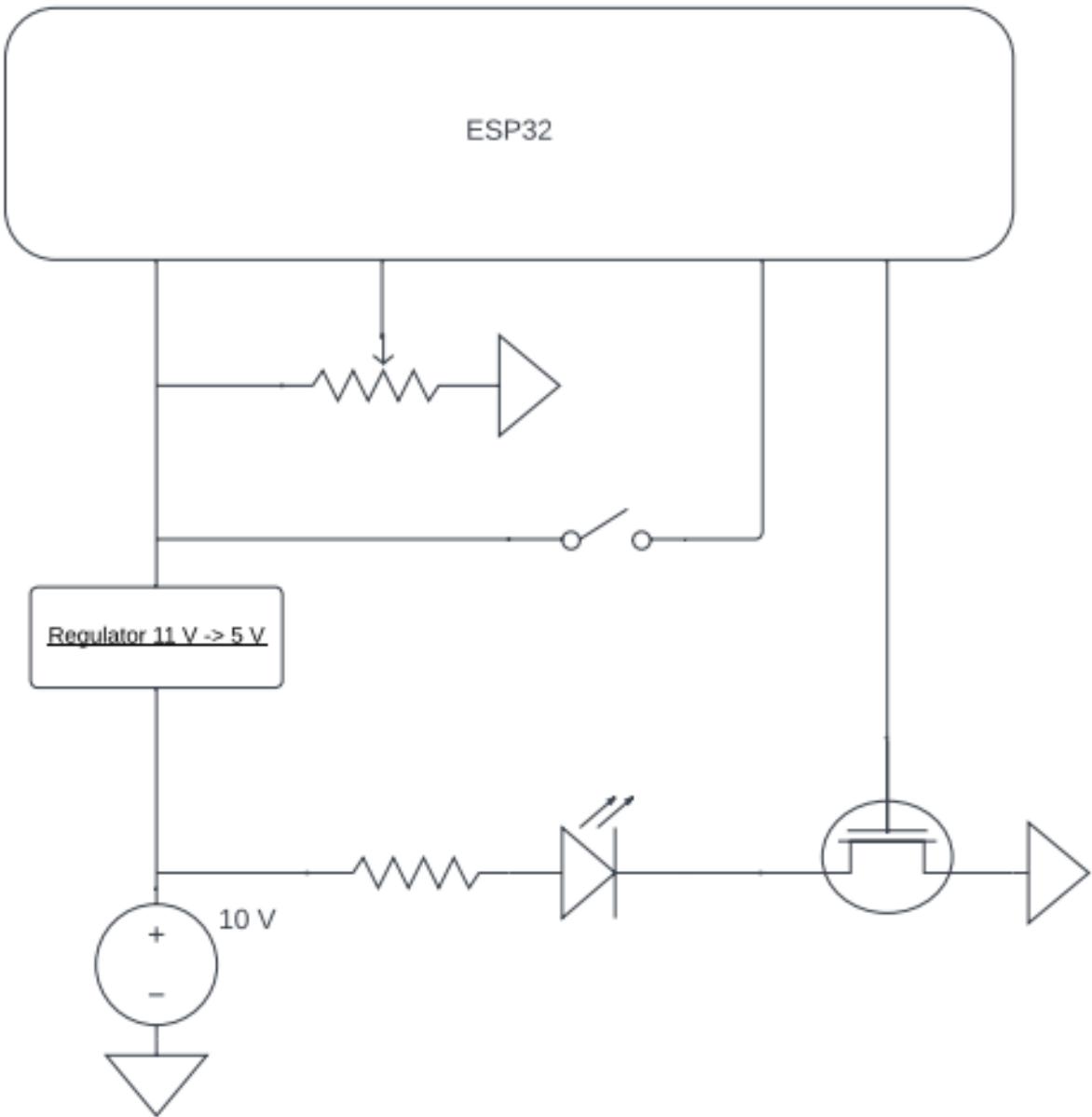


Figure 1: Laitteen kytkentäkaavio

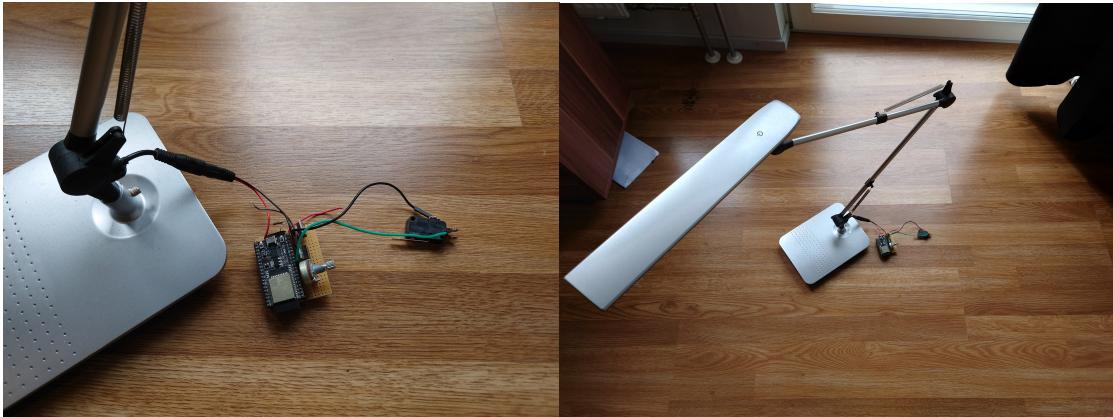
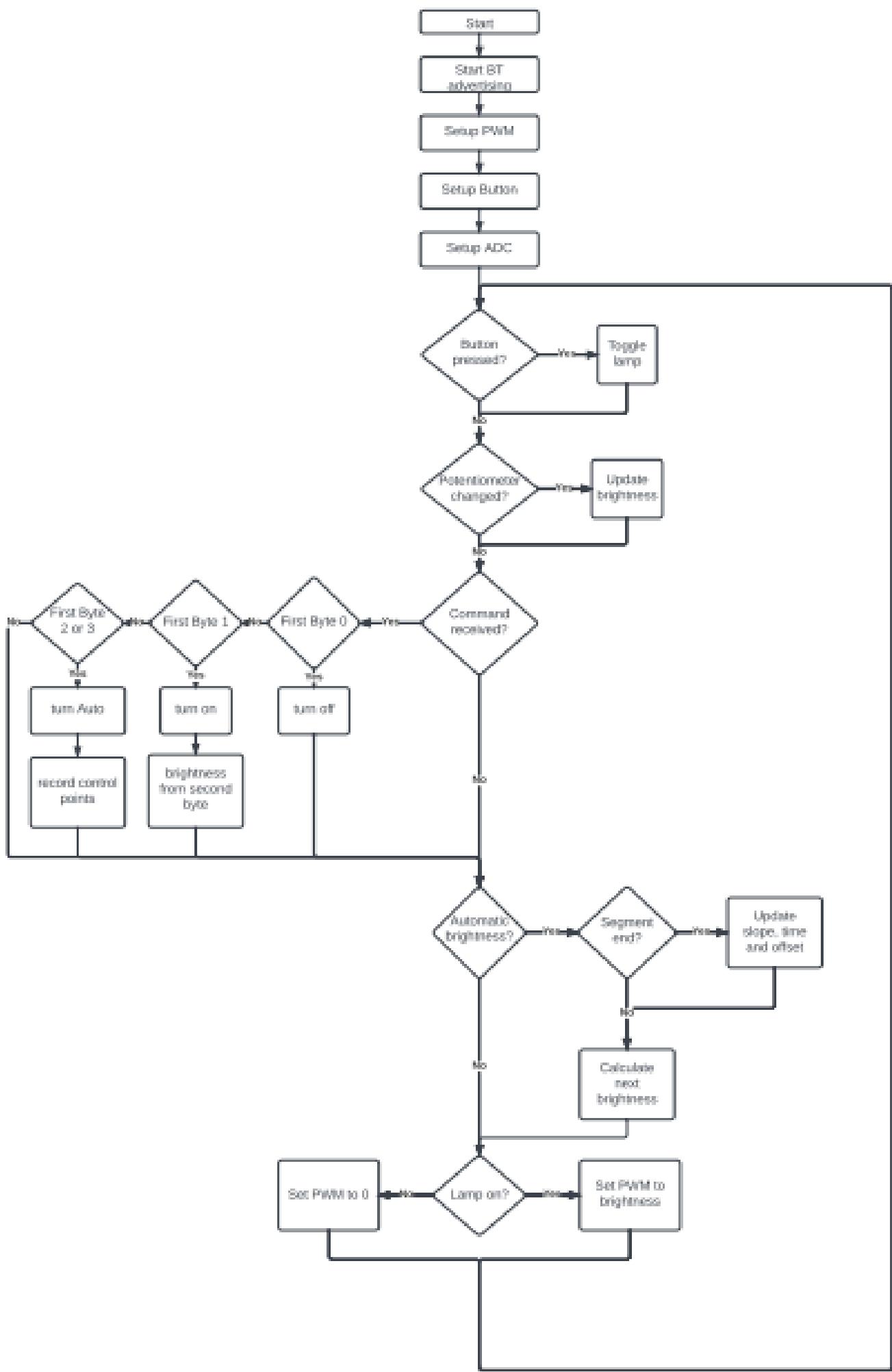


Figure 2: Kuva laitteen elektroniikasta Figure 3: Kuva koko laitteesta

3 LAITTEEN OHJELMISTO

ESP32 ohjaa MOSFETia ja käyttää sitä PWM-hakkurina. ESP32 itse saa ohjauksensa joko Bluetoothin yli tai potentiometriltä ja mikrokytkimeltä, jotka ajavat virtakytkimen ja kirkkaudensäädön virkoja. Viimeisenä säädetty ohjaus on etusijalla, ja mekaaniset ohjaimet ohittavat ohjelmoidun kirkastumisen.

Laite vastaanottaa Bluetoothin kautta yksinkertaisia signaaleja, mahdolliset komennot ovat pois, päälle <kirkkaus> ja ohjelmoitu kirkastuminen <hallintapisteet>. Laite lukee jatkuvasti kytkimen ja potentiometrin asentoa, ja päivittää PWM-aliohjelman ohjausarvon niiden mukaisesti. Jos laite on vastaanottanut viestin, se asettaa kirkkauden sen mukaisesti tai alkaa interpoloimaan ohjauspisteiden välilä. Interpoloatio keskeytyy, jos laite saa muita ohjauksia.



4 OHJAINSOVELLUS

Laitteelle tehtiin oma ohjaussovellus Android studiolla. Kyseessä on yksinkertainen yksinäyttöinen sovellus, joka osaa lähetä laitteelle edellä mainitut peruskomennot sekä asettamaan herätyksen, jolloin se lähetää laitteelle ohjelmoidun kirkastumisen ohjeen kahdella pisteellä alkaen aloitusajasta ja -kirkkaudesta ja päättyen lopetusaikeaan ja -kirkkauteen. Lopetusajan olisi ajatus täsmätä perinteiseen herätykseen.

Sovelluksessa on vain yksi näyttö, jossa on kaikki tällä hetkellä käytettävissä olevat ohjaimet. Suurinta osaa vastaa jokin MainActivityn jäsenfunktio. Kirkkaussäädin säättää ensin kirkkautta, ja vapauttaessa lähetää on-kutsun asetetulla kirkkaudella. On lähetää vain on-kutsun.

Herärys asetetaan asettamalla toivottu kirkkaus herätyksen alussa ja lopussa välillä 0-255, aloitus- sekä lopetusajat ja painamalla Set. Ajan säädöt päivittävät ohjelmaan tallennetut ajat, ja Set lukee kirkkauden tekstikentät sekä siirtää arvot Androidin hälytysjärjestelmään.

Androidin hälytysjärjestelmä lähetää hälytyksen osuessa sovellukselle kutsun, joka saa sen lähetämään ohjelmoidun kirkastumisen ohjeet itse laitteelle.



Figure 5:
*Ohjainsovelliuksen
näkymä*

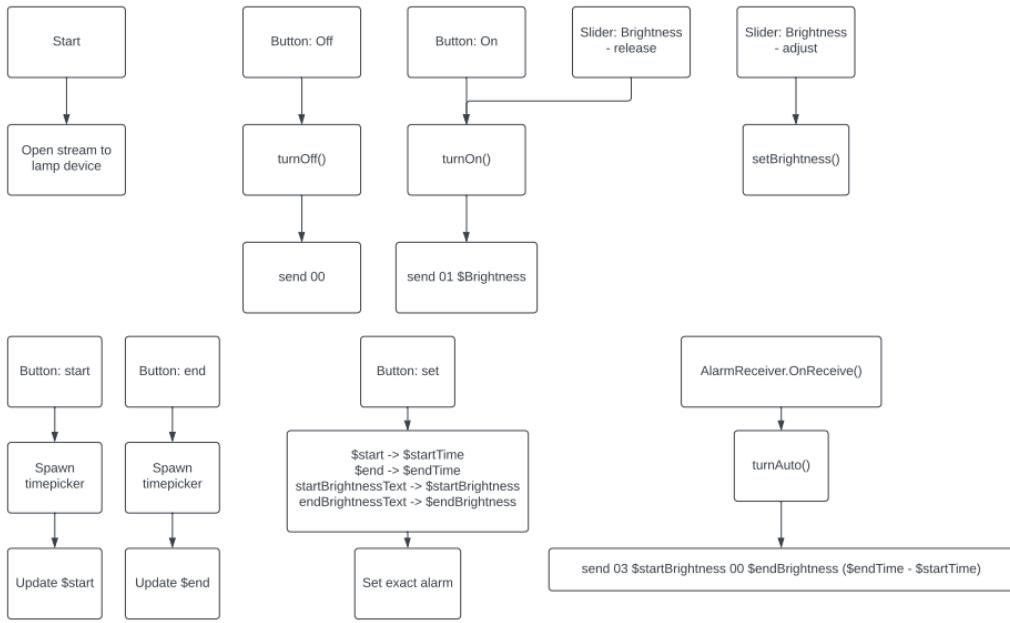


Figure 6: Ohjainsovelluksen tila- ja tapahtumakaavio

5 MAHDOLLISTA JATKOKEHITYSTÄ

- Laitteeseen
 - Kunnollinen piirilevy, joka käyttäisi ESP32-pakettia sellaisenaan kehitysalustan sijasta. Piirilevyn olisi tarkoitus olla riittävän pieni, että sen saa asennettua alkuperäisen paikalle lampun sisään.
 - Kunnollinen koteloointi. Vaatii joko uuden alustan ja elektroniikan kytkemisen tehonsyöttöön tai mielummin edellisen kohdan toteutumisen, jolloin lampun oman kotelon muokkaaminen riittäisi.
 - Yhteysongelmien korjaaminen. Nykyisellään paritettua laitetta ei voi yhdistää uudestaan puhelimeen yhteyden katkettua, vaan se pitää unohtaa ja parittaa uudelleen. Syy tuntematon.
- Sovellukseen

- Yleistä koodin laadun parantamista. Nykyisellään sovelluksen koodi on sekavaa ja tekee kaiken MainActivityn eli sovelluksen ensisijaisen graafisen komponentin kautta. Esimerkiksi kommunikaatio ja herätykset pitäisi erottaa omaan osioonsa.
 - Vakaus- ja yhteysongelmien korjaaminen. Sovellus odottaa juuri tiettyä laitetta, prototyypissä käyttämääni ESP32:hta, kovakoodatulla MAC-osoitteella. Jos sitä ei ole yhdistetty sovelluksen käynnistyessä, se kaatuu. Niin myös jos yhteys on katkennut lähetystä yrittääessä. Sovelluksen olisi hyvä osata yhdistää muihinkin laitteisiin, yrittää yhteyttä lähetäessä ja hallita poikkeustilanteita.
 - Kyky herättää levosta. Nykyisellään herätyksellä vain, jos sovellus on käynnissä sen alkaessa. Käytetyn järjestelmän sinänsä pitäisi pystyä herättämään laite lepotilasta ja käyttämään sovelluksen rutiineja. Puute saattaa johtua aiemmin mainituista, riippuvuus graafisesta käyttöliittymästä ja yhteysongelmat voivat estää heräystä toimimasta oikein.
 - Kunnollinen herätskäyttöliittymä. Nykyisellään sovellus osaa ajastaa yhden herätyksen, jota ei voi kumota. Sillä olisi hyvä pystyä ajastamaan ja kumoamaan useita herätyksiä, asettaa toistuvia herätyksiä jne.. Yleisesti sen olisi hyvä toteuttaa modernin herätyssovelluksen perustoiminnot. Myös kyky laukaista kuultava herätsääni olisi hyvä lisä.
 - Integraatio sovelluksen ja Androidin oman kellosovelliuksen välillä. Tämä voisi korvata edellisen kohdan, mutta ei ole välttämättä mahdollista; toisen sovelluksen asettamiin herätyksiin ei pääse käsiksi.
 - Kyky lähetä monimutkaisempia kirkastumisohjelmia. Nykyisellään sovellus lähetää vain kaksi pistettä, joiden kirkkaus on vapaavalintainen, mutta joista toinen on aina ajalla 0 ja toinen n minuutin päästä. Laite osaa käsittää mielivaltaisen määrän hallintapisteitä mielivaltaisilla kirkauksilla ja lähes mielivaltaisilla ajoilla (minuutin tarkkuudella, jatkuvasti kasvava)
- Molempien
 - Kaksisuuntainen liikenne. Nykyisellään sovellus ei tiedä laitteen tilaa, vaan osaa lähetä sille vain pyytöjä. Laite voisi lähetä tiedon sovellukselle, mikä olisi hyödyllistä erityisesti sen saadessa ohjauksia jostain muualta kuin ohjaimelta.
 - Siirtyminen Matter-verkkoon. Kyseessä on uusi IoT-laitteille tarkoitettu protokolla, joka toimii eri rajapintojen yli, jota ei ollut olemassa projektin alussa. Se tekisi sekä lampusta että sovelluksesta monipuolisempia, ja voisi tehdä laitekohtaisen sovelluksen tarpeettomaksi. Vaatii suuria muutoksia sekä laitteeseen että sovellukseen.
 - Bezier-interpoloatio nykyisen lineaarisen lisäksi. Alkuun neljällä pisteellä ja myöhemin jatketulla Bezier-käyrällä. Tämä pitäisi toteuttaa erikseen sekä sovelluksessa että laitteessa. Tämä tulisi tehtyä lähinnä huvin vuoksi, ominaisuudesta ei ole merkittävää hyötyä.

6 MITTAUKSET JA TESTAUS

Laitteen testaamiseksi on kokeiltu kirkkauen muuttamista ja virran kytkemistä sekä etänä puhelimen BT-sarjaterminaalista ja sovelluksesta että mekaanisista ohjaimista. Lisäksi on asetettu herätyksiä sovelluksesta ja katsottu laitteen reagoivan niihin odotetusti.

Kirkkautta on arvioitu silmämääräisesti ja PWM-signaalia tutkittu oskilloskoopilla. Laitteen toimintaa on tarkasteltu myös seuraamalla sen tuottamia lokitulosteita tietokoneella.

PWM-signaalin pulssinleveys muuttuu odotetusti lineaarisesti kirkkausarvon mukana. Havaittu kirkkaus vaikuttaa muuttuvan nopeammin matalilla kirkkauksilla.

Mekaaninen säätö on hieman vajavainen, koska kirkkauen saa nostettua vain ~0.75 asti potentiometrinä. Tämä vaatii kalibointia.

Etäohjattuna lampun kirkkaus muuttuu odotetusti koko välillä 0-1. Ohjelmoitua kirkastumista valo seuraa pääosin odotetusti, mutta se lopettaa säädön ennen kuin viimeisen hallintapisteen kirkkaus on saavutettu. Tämän seurauksena kirkkaus jäää hieman ohi tavoitearvosta.

7 JOHTOPÄÄTÖKSET

Laite toimii nykyisellään kirkkaussäädetävänä valaisimena. Sekä paikallinen ohjaus että etäohjaus toimivat odotetusti. Herätyks toimii teknisesti ottaen, sillä laite lähtee kirkastumaan odotettua tahtia odotettuun kellonaikaan. Ilman kykyä herättää ilman sovelluksen aukipitämistä se ei kuitenkaan ole käytännöllinen herätyskello, ja herätyks pitää vanhanaikaisesti asettaa joka kerta uudestaan. Laitteen käytännöllisyttä valaisimena rajoittaa myös koteloinnin puute ja sen ulkopuolella roikkuva suojaamaton elektroniikka.

Merkittäväät toiminnalliset puutteet, tärkeimpänä toimimattomuus käytännöllisenä herätyskellona, ovat kiinni vain puhelinsovelluksesta, laite itse on niiden osalta valmis. Laitteen omat puutteet liittyvät lähinnä viimestelyyn ja kotelointiin. Muuten laite vastaa vaatimusmäärittelyä.

LIITE 1: TYÖNJAKO JA MITÄ OPITTIIN

Kyseessä oli yksilötyö, joten työnjakoa ei ollut. Opin ESP32-ohjelmoinnista ja ESP-IDF ympäristön käytöstä, ESP32 kommunikaatiostackista, Android-ohjelmoinnista ja jonkin verran Bluetooth-stackista.

LIITE 2: ESP32-KOODI

```
#include <stdio.h>
#include <string.h>
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "driver/ledc.h"
#include "driver/adc.h"
#include "esp_timer.h"
#include "esp_bt.h"
#include "esp_spp_api.h"
#include "esp_bt_device.h"
#include "esp_bt_main.h"
#include "esp_gap_bt_api.h"
#include "esp_log.h"
#include "nvs.h"
#include "nvs_flash.h"

#define mos_pin 14
#define button 26
#define NUM_READINGS 100

#define SPP_TAG "IOT_LAMP"

int received = 0;
uint8_t* rdata = NULL;

//Struct for control points to control the brightness of the lamp during
wakeup
typedef struct {
    float brightness;    //Brightness in range [0, 1]
    int time;           //Time from start in minutes
}cpoint;

//SPP callback function for BT communication
static void esp_spp_cb(esp_spp_cb_event_t event, esp_spp_cb_param_t *param)
{
    switch (event) {
    case ESP_SPP_INIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_INIT_EVT");
        esp_spp_start_srv(ESP_SPP_SEC_NONE, ESP_SPP_ROLE_SLAVE, 0,
"SPP_SERVER");
    }
```

```

        break;
    case ESP_SPP_DISCOVERY_COMP_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_DISCOVERY_COMP_EVT");
        break;
    case ESP_SPP_OPEN_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_OPEN_EVT");
        break;
    case ESP_SPP_CLOSE_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CLOSE_EVT");
        break;
    case ESP_SPP_START_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_START_EVT");
        esp_bt_dev_set_device_name("ESP_LAMP");
        esp_bt_gap_set_scan_mode(ESP_BT_CONNECTABLE,
ESP_BT_GENERAL_DISCOVERABLE);
        break;
    case ESP_SPP_CL_INIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CL_INIT_EVT");
        break;
    case ESP_SPP_DATA_IND_EVT:
        rdata = (uint8_t*) malloc(param->data_ind.len);
        memcpy(rdata, param->data_ind.data, param->data_ind.len);
        received = param->data_ind.len;
        break;
    case ESP_SPP_CONG_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_CONG_EVT");
        break;
    case ESP_SPP_WRITE_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_WRITE_EVT");
        break;
    case ESP_SPP_SRV_OPEN_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_SRV_OPEN_EVT");
        break;
    case ESP_SPP_SRV_STOP_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_SRV_STOP_EVT");
        break;
    case ESP_SPP_UNINIT_EVT:
        ESP_LOGI(SPP_TAG, "ESP_SPP_UNINIT_EVT");
        break;
    default:
        break;
    }
}

void esp_bt_gap_cb(esp_bt_gap_cb_event_t event, esp_bt_gap_cb_param_t *param)
{
    switch (event) {
    case ESP_BT_GAP_AUTH_CMPL_EVT:
        if (param->auth_cmpl.stat == ESP_BT_STATUS_SUCCESS) {
            ESP_LOGI(SPP_TAG, "authentication success: %s", param-
>auth_cmpl.device_name);

```

```

        esp_log_buffer_hex(SPP_TAG, param->auth_cmpl.bda,
ESP_BD_ADDR_LEN);
    } else {
        ESP_LOGE(SPP_TAG, "authentication failed, status:%d", param-
>auth_cmpl.stat);
    }
    break;
}
case ESP_BT_GAP_PIN_REQ_EVT:{
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_PIN_REQ_EVT min_16_digit:%d", param-
>pin_req.min_16_digit);
    if (param->pin_req.min_16_digit) {
        ESP_LOGI(SPP_TAG, "Input pin code: 0000 0000 0000 0000");
        esp_bt_pin_code_t pin_code = {0};
        esp_bt_gap_pin_reply(param->pin_req.bda, true, 16, pin_code);
    } else {
        ESP_LOGI(SPP_TAG, "Input pin code: 1234");
        esp_bt_pin_code_t pin_code;
        pin_code[0] = '1';
        pin_code[1] = '2';
        pin_code[2] = '3';
        pin_code[3] = '4';
        esp_bt_gap_pin_reply(param->pin_req.bda, true, 4, pin_code);
    }
    break;
}
#endif (CONFIG_BT_SSP_ENABLED == true)
case ESP_BT_GAP_CFM_REQ_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_CFM_REQ_EVT Please compare the numeric
value: %d", (int) param->cfm_req.num_val);
    esp_bt_gap_ssp_confirm_reply(param->cfm_req.bda, true);
    break;
case ESP_BT_GAP_KEY_NOTIF_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_KEY_NOTIF_EVT passkey:%d", (int) param-
>key_notif.passkey);
    break;
case ESP_BT_GAP_KEY_REQ_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_KEY_REQ_EVT Please enter passkey!");
    break;
#endif
case ESP_BT_GAP_MODE_CHG_EVT:
    ESP_LOGI(SPP_TAG, "ESP_BT_GAP_MODE_CHG_EVT mode:%d", (int) param-
>mode_chg.mode);
    break;
default: {
    ESP_LOGI(SPP_TAG, "event: %d", (int) event);
    break;
}
}

```

```

    return;
}

void app_main(void){
    ESP_LOGI(SPP_TAG, "Main entry");
    //Initialize bluetooth
    esp_err_t ret = nvs_flash_init();
    ESP_LOGI(SPP_TAG, "Nvs Init comp");
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK( ret );

    ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_BLE));

    esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
    if ((ret = esp_bt_controller_init(&bt_cfg)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_controller_enable(ESP_BT_MODE_CLASSIC_BT)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable controller failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluetooth_init()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s initialize bluetooth failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bluetooth_enable()) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s enable bluetooth failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_bt_gap_register_callback(esp_bt_gap_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s gap register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }

    if ((ret = esp_spp_register_callback(esp_spp_cb)) != ESP_OK) {
        ESP_LOGE(SPP_TAG, "%s spp register failed: %s\n", __func__,
esp_err_to_name(ret));
        return;
    }
}

```

```

}

if ((ret = esp_spp_init(ESP_SPP_MODE_CB)) != ESP_OK) {
    ESP_LOGE(SPP_TAG, "%s spp init failed: %s\n", __func__,
esp_err_to_name(ret));
    return;
}

#endif

/*
 * Set default parameters for Legacy Pairing
 * Use variable pin, input pin code when pairing
 */
esp_bt_pin_type_t pin_type = ESP_BT_PIN_TYPE_VARIABLE;
esp_bt_pin_code_t pin_code;
esp_bt_gap_set_pin(pin_type, 0, pin_code);

//Initialize LED PWM timer
ledc_timer_config_t mos_timer = {
    .speed_mode      = LEDC_LOW_SPEED_MODE,
    .timer_num       = LEDC_TIMER_0,
    .duty_resolution = LEDC_TIMER_13_BIT,
    .freq_hz         = 5000,
    .clk_cfg         = LEDC_AUTO_CLK
};
ESP_ERROR_CHECK(ledc_timer_config(&mos_timer));

//Initialize LED PWM channel
ledc_channel_config_t mos_channel = {
    .speed_mode      = LEDC_LOW_SPEED_MODE,
    .channel         = LEDC_CHANNEL_0,
    .timer_sel       = LEDC_TIMER_0,
    .intr_type       = LEDC_INTR_DISABLE,
    .gpio_num        = mos_pin,
    .duty            = 0, // Set duty to 0%
    .hpoint          = 0
};
ESP_ERROR_CHECK(ledc_channel_config(&mos_channel));

//configure button pin
gpio_reset_pin(button);
gpio_set_direction(button, GPIO_MODE_INPUT);
gpio_set_pull_mode(button, GPIO_PULLDOWN_ONLY);

//Configure ADC
adc1_config_channel_atten(ADC1_CHANNEL_4, 3); //Potentiometer on pin 32

```

```

//Current and previous potentiometer reading
int pots[NUM_READINGS] = {0}; uint poti = 0;
int avg_pot = 0; int prev_avg = 0;

//Button values
int button_read = 0; uint64_t bstates = 0; unsigned int state = 0;

//Current brightness and whether or not the lamp is on
float brightness = 0.0f; int on = 0;

//Values for wakeup brightness control
int bezier = 0; //Whether or not the brightness curve is a bezier curve
int autom = 0; //Whether or not the brightness is currently determined
automatically
cpoint* control_points = NULL; //The brightness curve control points

//Values for linear interpolation
float k = 0; //The slope of the current piece
float b0 = 0; //The vertical offset of the piece
float t0 = 0; //The horizontal offset of the piece
uint pt = 0; //Current point
uint points = 0; //Number of points
float t1 = 0; //The stop time of the current piece

float tStart = 0; //Start time

while(1) {
    //Take button reading with debouncing
    if(gpio_get_level(button)){
        bstates |= 1 << state;
    }else bstates &= ~(1 << state);
    state = (state + 1) % 64;

    //Check the button reading and toggle lamp/override automatic control
    if high
        if(!button_read && !(~bstates)){
            button_read = 1;
            if(autom == false) on = !on;
            else autom = false;
            ESP_LOGI(SPP_TAG, "Button toggle");
        }else if(button_read && !bstates){
            button_read = 0;
        }

    //Check the potentiometer value with smoothing, update brightness and
    override automatic
    //control if altered sufficiently
    int pot = adc1_get_raw(ADC1_CHANNEL_4);
    avg_pot += pot / NUM_READINGS - pots[poti] / NUM_READINGS;
    pots[poti] = pot;
    poti = (poti + 1) % NUM_READINGS;
}

```

```

//ESP_LOGI(SPP_TAG, "Pot %i", pot);
if(abs(avg_pot - prev_avg) > 100){
    prev_avg = avg_pot;
    brightness = (float) avg_pot / 4095.0f;
    if(brightness < 0.0) brightness = 0.0;
    if(brightness > 1.0) brightness = 1.0;
    autom = false;
    ESP_LOGI(SPP_TAG, "Pot %i", pot);
    ESP_LOGI(SPP_TAG, "Manual adjust, average %i, brightness %f",
avg_pot, brightness);
}

if(received){
    ESP_LOGI(SPP_TAG, "Command received (%i bytes)", (int) received);
    switch(rdata[0]){
        case 0: //Turn off
            on = 0;
            autom = 0;
            ESP_LOGI(SPP_TAG, "Turning off");
            break;
        case 1: //Turn on
            on = 1;
            brightness = (float) rdata[1] / 255.0;
            autom = 0;
            ESP_LOGI(SPP_TAG, "Turning on; brightness %f",
brightness);
            break;
        case 2: //Set to brighten automatically (linear)
            tStart = esp_timer_get_time() / 1000000.0f;
            bezier = 0;
            on = 1;
            autom = 1;
            pt = b0 = k = t0 = t1 = 0;
            points = (received - 1) / 2;
            ESP_LOGI(SPP_TAG, "Setting up linear interpolation with %i
points", (int) points);
            if(control_points) free(control_points);
            control_points = (cpoint*) malloc(points *
sizeof(cpoint));
            if(!control_points){
                ESP_LOGI(SPP_TAG, "Memory allocation for cpoints
failed.");
                break;
            }
            for(uint i = 0; i < points; i++){
                cpoint point = {(float) rdata[i*2 + 1] / 255.0,
rdata[i*2 + 2]};
                control_points[i] = point;
                ESP_LOGI(SPP_TAG, "Point %i: brightness %f, time %i",
i, point.brightness, point.time);
            }
            break;
    }
}

```

```

        case 3: //Set to brighten automatically (Bezier - for now
interpreted as linear)
            bezier = 1;
            on = 1;
            autom = 1;
            pt = 0;
            points = (received - 1) / 2;
            ESP_LOGI(SPP_TAG, "Setting up Bezier interpolation with %i
points", (int) points);
            if(control_points) free(control_points);
            control_points = (cpoint*) malloc(points *
sizeof(cpoint));
            if(!control_points){
                ESP_LOGI(SPP_TAG, "Memory allocation for cpoints
failed.");
                break;
            }
            for(uint i = 0; i < points; i++){
                cpoint point = {(float) rdata[i*2 + 1] / 255.0,
rdata[i*2 + 2]};
                control_points[i] = point;
                ESP_LOGI(SPP_TAG, "Point %i: brightness %f, time %i",
i, point.brightness, point.time);
            }
            break;
        default: break;
    }
    received = 0;
    free(rdata);
    rdata = NULL;
}

//Automatic brightness control for wakeup
if(autom){
    float t = (float) esp_timer_get_time() / 1000000.0f - tStart;
    if(0 /*bezier*/){
        //Only linear interpolation for now
    }else{
        if(t >= t1){
            if(pt < points){
                t0 = 60*control_points[pt].time;
                t1 = 60*control_points[pt + 1].time;
                b0 = control_points[pt].brightness;
                if(t1 > t0){

                    k = (control_points[pt + 1].brightness - b0)/(t1 -
t0);
                }else if(t1 == t0){
                    brightness = control_points[pt + 1].brightness;
                }else{
                    free(control_points);
                    control_points = NULL;
                }
            }
        }
    }
}

```

```

        autom = false;
    }
}else{
    free(control_points);
    control_points = NULL;
    autom = false;
}
pt++;
}while(brightness < b1 || brightness > b2);
brightness = b0 + (t - t0)*k;
}
//ESP_LOGI(SPP_TAG, "Automatic adjust, brightness %f",
brightness);
}

//Set the PWM value based on the brightness and whether the lamp is on
if(on){
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0,
brightness * 8191.0f));
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_LOW_SPEED_MODE,
LEDC_CHANNEL_0));
}else{
    ESP_ERROR_CHECK(ledc_set_duty(LEDC_LOW_SPEED_MODE, LEDC_CHANNEL_0,
0));
    ESP_ERROR_CHECK(ledc_update_duty(LEDC_LOW_SPEED_MODE,
LEDC_CHANNEL_0));
}
}
}

```

LIITE 3: ANDROID-KOODI (OHJELMAKOODI, EI LAYOUT)

```
package com.example.controller

import android.app.AlarmManager
import android.app.AlarmManager.RTC_WAKEUP
import android.app.Dialog
import android.app.PendingIntent
import android.app.TimePickerDialog
import android.bluetooth.*
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager.PERMISSION_GRANTED
import android.icu.util.Calendar
import android.os.Bundle
import android.text.format.DateFormat
import android.text.format.DateFormat.is24HourFormat
import android.widget.Button
import android.widget.SeekBar
import android.widget.SeekBar.OnSeekBarChangeListener
```

```

import android.widget.TextView
import android.widget.TimePicker
import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.DialogFragment
import java.io.OutputStream
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale

class StartTimePickerFragment : DialogFragment(),
TimePickerDialog.OnTimeSetListener {

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        // Use the current time as the default values for the picker.
        val c = Calendar.getInstance()
        val hour = c.get(Calendar.HOUR_OF_DAY)
        val minute = c.get(Calendar.MINUTE)

        // Create a new instance of TimePickerDialog and return it.
        return TimePickerDialog(activity, this, hour, minute,
DateFormat.is24HourFormat(activity))
    }

    override fun onTimeSet(view: TimePicker, hourOfDay: Int, minute: Int) {
        val c = Calendar.getInstance()
        val cHour = c.get(Calendar.HOUR_OF_DAY)
        val cMinute = c.get(Calendar.MINUTE)

        if(hourOfDay*60 + minute > cHour*60 + cMinute){
            start = c.timeInMillis - c.get(Calendar.MILLISECONDS_IN_DAY) +
hourOfDay*3600000 + minute*60000
        }else{
            start = c.timeInMillis - c.get(Calendar.MILLISECONDS_IN_DAY) +
(24+hourOfDay)*3600000 + minute*60000
        }

        ma?.findViewById<Button>(R.id.startTime)?.text = "$hourOfDay:$minute"
    }
}

class EndTimePickerFragment : DialogFragment(),
TimePickerDialog.OnTimeSetListener {

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        // Use the current time as the default values for the picker.
        val c = Calendar.getInstance()
        val hour = c.get(Calendar.HOUR_OF_DAY)
        val minute = c.get(Calendar.MINUTE)

        // Create a new instance of TimePickerDialog and return it.
        return TimePickerDialog(activity, this, hour, minute,
DateFormat.is24HourFormat(activity))
}

```

```

    }

    override fun onTimeSet(view: TimePicker, hourOfDay: Int, minute: Int) {
        val c = Calendar.getInstance()
        val cHour = c.get(Calendar.HOUR_OF_DAY)
        val cMinute = c.get(Calendar.MINUTE)

        if(hourOfDay*60 + minute > cHour*60 + cMinute){
            end = c.timeInMillis - c.get(Calendar.MILLISECONDS_IN_DAY) +
            hourOfDay*3600000 + minute*60000
        }else{
            end = c.timeInMillis - c.get(Calendar.MILLISECONDS_IN_DAY) +
            (24+hourOfDay)*3600000 + minute*60000
        }

        ma?.findViewById<Button>(R.id.endTime)?.text = "$hourOfDay:$minute"
    }
}

class AlarmReceiver : BroadcastReceiver(){
    override fun onReceive(context: Context, intent: Intent) {
        ma?.turnAuto()
    }
}

var start: Long = 0
var end: Long = 0
var ma: MainActivity? = null

class MainActivity : AppCompatActivity() {

    private var lamps: List<BluetoothDevice>? = emptyList()
    private var oStream: OutputStream? = null

    private var brightness: Int = 255

    private var alarmDesc: String = "No alarm set"

    private var startTime: Long = 0 //Brightening start time in millisecond
    Unix time
    private var endTime: Long = 0
    private var startBrightness: Int = 0
    private var endBrightness: Int = 0

    private var alarmManager: AlarmManager? = null

    private val seekBarListener: OnSeekBarChangeListener = (object :
    OnSeekBarChangeListener{
        override fun onProgressChanged(seek: SeekBar,
                                       progress: Int, fromUser: Boolean) {
            setBrightness(progress)
        }
    })
}

```

```

        }

        override fun onStartTrackingTouch(seek: SeekBar) {

        }

        override fun onStopTrackingTouch(seek: SeekBar) {
            turnOn()
        }
    })

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    ma = this

    setContentView(R.layout.activity_main)

    findViewById<Button>(R.id.button).setOnClickListener{turnOff()}
    findViewById<Button>(R.id.button2).setOnClickListener{turnOn()}
    findViewById<Button>(R.id.setButton).setOnClickListener{setSlope()}

    findViewById<Button>(R.id.startTime).setOnClickListener {
        StartTimePickerFragment().show(supportFragmentManager,
        "timePicker")
    }
    findViewById<Button>(R.id.endTime).setOnClickListener {
        EndTimePickerFragment().show(supportFragmentManager, "timePicker")
    }
    findViewById<Button>(R.id.startTime).text = "??:??"
    findViewById<Button>(R.id.endTime).text = "??:??"

    findViewById<TextView>(R.id.description).text = alarmDesc

    findViewById<SeekBar>(R.id.seekBar).setOnSeekBarChangeListener(seekBarListener
    )

    alarmManager = getSystemService(Context.ALARM_SERVICE) as AlarmManager

    val blM: BluetoothManager =
    getSystemService(BluetoothManager::class.java)
    val blA: BluetoothAdapter? = blM.adapter

    if(checkSelfPermission("android.permission.BLUETOOTH_SCAN") != PERMISSION_GRANTED) return
    if(checkSelfPermission("android.permission.BLUETOOTH_CONNECT") != PERMISSION_GRANTED) return

    val pairedDevices: Set<BluetoothDevice>? = blA?.bondedDevices
    lamps = pairedDevices?.filter{it.name == "ESP_LAMP" || it.address == "A8:03:2A:6A:2A:52"}
    if(lamps == null || lamps?.isEmpty() == true){

```

```

        return
    }

    val socket: BluetoothSocket =
lamps?.first()!!.createInsecureRfcommSocketToServiceRecord(lamps!! .first().uui
ds.first().uuid)
        blA?.cancelDiscovery()
        val a = socket.connectionType
        val b = lamps?.first()?.bondState
        val c = socket.connectionType
        val d = socket.isConnected
        val uuid = lamps!! .first().uuids
        socket.connect()
        oStream = socket.outputStream
    }

private fun turnOn(){
    val message: ByteArray = ByteArray(2)
    message[0] = 1
    message[1] = brightness.toByte()
    oStream?.write(message) ?: return
}

private fun turnOff(){
    oStream?.write(0) ?: return
}

public fun turnAuto(){
    if(startTime > endTime) return

    val message: ByteArray = ByteArray(5)
    message[0] = 2

    //First control point; start at T+0 min
    message[1] = startBrightness.toByte()
    message[2] = 0

    //Second control point
    message[3] = endBrightness.toByte()
    message[4] = ((endTime - startTime)/60000).toByte()

    oStream?.write(message) ?: return
}

private fun setBrightness(b: Int){
    brightness = b
}

private fun setSlope(){
    startTime = start
    endTime = end
}

```

```
        startBrightness =
findViewById<TextView>(R.id.startBright).text.toString().toInt()
        endBrightness =
findViewById<TextView>(R.id.endBright).text.toString().toInt()
        findViewById<TextView>(R.id.description).text = alarmDesc
        if(startTime > endTime) return

        val startDate = Date(startTime)
        val endDate = Date(endTime)

        alarmDesc = "Alarm set from $startBrightness at $startDate to
$endBrightness at $endDate"
        findViewById<TextView>(R.id.description).text = alarmDesc

        val alarmIntent: Intent = Intent(this, AlarmReceiver::class.java)
        val pi = PendingIntent.getBroadcast(this, 0, alarmIntent,
PendingIntent.FLAG_IMMUTABLE)
        alarmManager?.setExact(RTC_WAKEUP, startTime, pi)
    }
}
```