UNIVERSIDADE DE BRASÍLIA – UNB
FACULDADE GAMA

ELETRÔNICA EMBARCADA 120961- TURMA A 2019.2

PONTO DE CONTROLE IV

PROJETO: SISTEMA DE DESPERTADOR CONFIGURÁVEL COM CONTROLE DE DISPOSITIVOS EXTERNOS.

Alexandre Torres Kryonidis
Matrícula: 13/0099767

Engenharia de Software

Faculdade Gama - Universidade de Brasília
Área Especial de Indústria Projeção A Brasília,
CEP: 72.444-240

email: alexandrekry@gmail.com

RESUMO

Este relatório tem como objetivo apresentar a proposta do nosso projeto da disciplina de Sistemas Operacionais Embarcados, um dispositivo despertador ajustável com sistema embarcado que permite o controle de dispositivos externos por meio de relés. A implementação do projeto permitirá o controle de motores para levantar/abaixar persianas, o acionamento de lâmpadas, ou o acionamento de outros tipos de equipamentos que podem ser acionados via relês. Para tanto usaremos como elemento principal do produto a RASPBERRY PI 3 MODEL B, da *Fundação Raspberry Pi*, uma série de computadores de placa única do tamanho reduzido. As funcionalidades do sistema trazem facilidade e conforto ao usuário e o garantem um sono com mais qualidade e um despertar mais suave.

1. JUSTIFICATIVA

Cada vez mais destaca-se a importância de um sono de qualidade para o ser humano, seja para uma vida saudável, melhor memória ou até mesmo desempenho profissional. Enquanto a qualidade do sono está diretamente ligada à qualidade de vida do ser humano, por outro lado as consequências de um sono de má qualidade vão de estresse e ansiedade, a curto prazo, a complicações cardiovasculares após alguns anos [2].

Ao analisar as características de um sono de qualidade, percebe-se que muitos fatores dependem de ações e escolhas do indivíduo, como ir dormir cedo, se alimentar bem, não interromper ciclos de sono, etc [3]. Portanto, uma maneira de melhorar a qualidade de vida da pessoa nesse sentido é, contando que o sejam seguidas as boas práticas para o bom sono, melhorar o processo de despertar da mesma. Com isso, é possível notar que o despertar de uma pessoa pode muitas vezes ser estressante e por muitas vezes assustador, pois em sua maioria ocorre de forma abrupta.

O processo de relógio biológico do sono de um ser humano é afetado diretamente pela luz natural, este regulando aquele [4], de forma que o hormônio responsável pela indução do sono, a melatonina, cresce proporcionalmente a ausência de luz, o que torna o processo de iluminação natural um grande auxiliador no despertar tranquilo de uma pessoa [6].

Daí surge a necessidade de apresentar um dispositivo que auxilie no processo de despertar, de forma que a pessoa desperte mais tranquilamente no horário desejado, com seus níveis de melatonina

reduzidos, e não de maneira repentina que interrompe ciclos de sono como convencional.

O dispositivo proposto permite que em um tempo estabelecido antes do horário desejado de despertar, sejam executadas ações externas como o controle de motores para a abertura de uma cortina, permitindo que a luz natural entre no quarto gradativamente, outra opção é ligar lâmpadas. Quando o corpo percebe a claridade, o processo de despertar se dá de maneira mais natural. Isso ocorre porque o nível de melatonina (hormônio responsável pela indução ao sono) cai devido à luz. Além desse artifício, o dispositivo também emite sons a escolha do usuário no horário desejado para o ajudar a acordar sem susto.

2. OBJETIVOS

A. Objetivos gerais

Projetar um dispositivo embarcado em uma Raspberry Pi [1], que realize a função de despertar o usuário com o toque e/ou a abertura de cortina, em uma hora específica, sendo tudo programável conforme necessidade do usuário.

Deseja-se construir um sistema desacoplado do sistema principal (plataforma web de interação usuário-despertador), de tal forma que nesta interface o usuário possa definir as configurações e preferências suas no despertador.

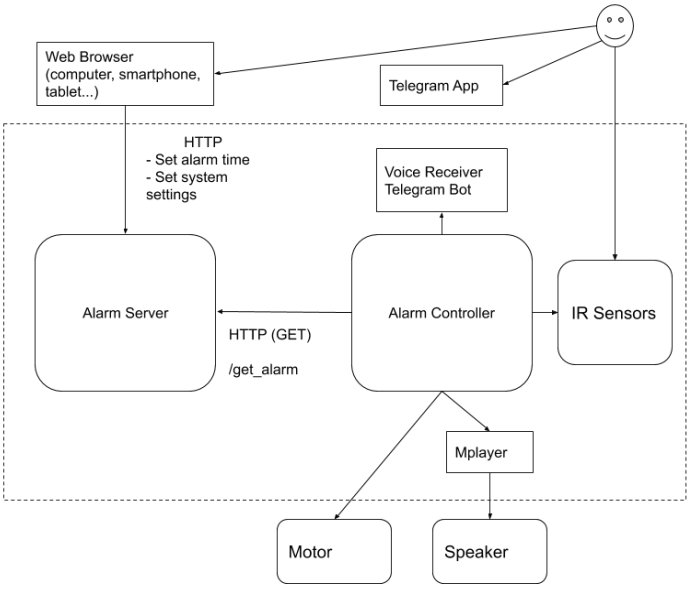
B. Objetivos específicos

Neste sistema auxiliar o usuário deve ser apto a:

- Fazer upload e troca de música ou toque escolhido para despertar;
- Definir horário do alarme de despertar;
- Selecionar se, e em quantos minutos antes do alarme abrir a persiana;
- Ajustar configurações básicas do despertador tais como: "somente abrir a persiana", "somente tocar música", "abrir a persiana e tocar música", etc;
- Ser um sistema de fácil utilização e intuitivo, em que o usuário gaste pouco tempo para ativar e desativar um alarme;
- Ter um dispositivo de baixo custo e confiável.

3. VISÃO GERAL DO PROJETO

O sistema geral pode ser decomposto em subsistemas. A partir da imagem a seguir é possível observar os principais subsistemas que compõem o projeto: *Alarm Controller, Alarm Server* e *Voice Receiver Telegram Bot*.



C. Subsistemas de software

- **Alarm Server**

Este é o principal sistema que possibilita a interação do usuário com o sistema, é uma aplicação web que permite com que o usuário configure as informações do alarme. Nesta aplicação o usuário seleciona o horário de despertar, a música que será tocada, se deseja somente abrir a persiana, se deseja somente tocar a música, ou se deseja realizar ambas

as operações e quanto tempo deve se esperar para realizar a próxima operação.

Este sistema é um servidor que roda na raspberry e é visualizado pelo usuário por meio do navegador.

- **Alarm Client**

Este é o software que tem como objetivo comunicar com os diversos componentes eletrônicos do projeto, como a execução do áudio, obter informação dos sensores e controlar o motor. Para realizar ações opcionais o usuário pode interagir com este sistema por meio do sensor infravermelho. O usuário pode aumentar e reduzir o volume, movimentar o motor e cancelar o alarme.

- **Web Browser**

O navegador é externo ao projeto, porém é a interface utilizada pelo usuário para configurar o funcionamento do restante do sistema.

- **Receiver Telegram Bot**

Caso o usuário deseje utilizar o sistema para lembrá-lo de algo pode gravar um áudio no telegram. Desta forma, quando ele setar o alarme como lembrete na aplicação web, o que ele disse por último no telegram será reproduzido ao invés do som do alarme.

D. Subsistemas de eletrônica

- **Speaker**

Para tocar músicas, a comunicação será feita usando a entrada *audio jack* já presente na raspberry pi.

- **Sensors**

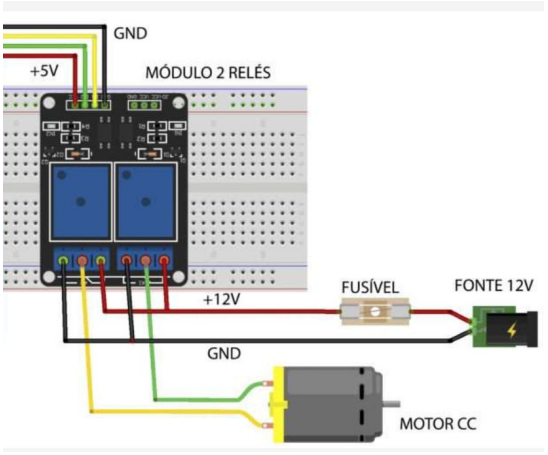
Os sensores utilizados são:

- Sensor infravermelho para controle do motor e do som.

- **Acionamento de relés**

Serão utilizados dois relés para possibilitar o controle o motores nas duas direções. Possibilita também o acionamento de até 2 lâmpadas.
Além disso, os relés utilizados são 5v, portanto, foram utilizados transistores para possibilitar o acionamento utilizando os pinos de 3.3v do GPIO da RaspberryPi.

Na imagem abaixo está o circuito a ser utilizado, para controlar o motor, a raspberry altera o nível lógico dos fios amarelo e verde que controlam o relé.



4. REQUISITOS DO PROJETO

As especificações do projeto a ser desenvolvido serão descritas neste tópico, serão apresentados os requisitos que ele deverá atender.

A. Requisitos globais

- O usuário deve poder selecionar horário de despertar.

- O usuário deve poder fazer upload de música e sons a serem tocados.

- O sistema deve facilitar o upload e troca de músicas e sons.

- O usuário deve poder escolher o intervalo antes do alarme para acionar os relês.

- O usuário deve ser capaz de selecionar entre opções de: "somente acionar relé", "somente tocar música" e "acionar relé e tocar música".

- O usuário deve ser capaz de refazer uma configuração feita incorretamente.

- O sistema auxiliar de configurações do despertador deve ser desacoplado do dispositivo físico.

- O usuário deve apertar um botão para desligar o despertador.

- O usuário deve selecionar entre lembrete e alarme.

B. **Requisitos do subsistemas**

- **Alarm Server**
- Deve gerenciar Alarmes:
    - Criar
    - Visualizar
    - Deletar

- Deve permitir visualização e alterações das configurações.

- Deve possuir endpoints para:
    - Verificar um se alarme deve ser tocado neste instance. E retornar as informações sobre o alarme.

- Deve armazenar os dados em um Banco de Dados para garantir a persistência e integridade dos mesmos.

- Os alarmes semanais devem repetir todo dia da semana em que foi programado.

- Os alarmes não semanais devem ser deletados após serem executados.

- Deve permitir a escolha entre alarme e lembrete.

- **Alarm Client**
- Deve realizar consultas via HTTP para o *endpoint* /get_alarm do *Alarm Server.*

- Deve acionar os relés a partir do servidor.

- Deve acionar os relés a partir de infravermelho

- Deve receber sinais infravermelhos para cancelar alarmes.

- Deve receber sinais infravermelhos para aumentar e diminuir o volume da RaspberryPi.

- Deve utilizar a saída de áudio da raspberry para tocar o som.

- Speaker
- Deve tocar o áudio por meio de um alto-falante externo.

- Relés
- Deve ser bem isolado do restante do circuito da RaspberryPi, de tal forma, que contenha circuito de proteção que evite curtos.

5. LISTA DE MATERIAIS

- 1x Raspberry PI 3 Model B+
- 1x Carregador Raspberry
- 2x Módulos Relés
- 2x 2N2222 Transistor
- 1x Controle Infravermelho
- 1x Receptor Infravermelho
- 1x Caixa de Som / Fone de ouvido
- 1x Motor DC
- 1x Fonte para o Motor
- 1x Persiana Pequena
- Resistores

- Jumpers

6. DETALHES DE IMPLEMENTAÇÃO

A. Subsistemas de software

- **Alarm Server**

O servidor foi desenvolvido utilizando o Framework Ruby on Rails. Essa escolha foi feita por que um dos integrantes do grupo, que nunca havia trabalhado com programação web, aprenderia esta tecnologia em uma outra disciplina ao longo do semestre. Desta forma, a escolha foi estratégica com o objetivo de facilitar a contribuição dos dois membros ao projeto.

A arquitetura do padrão do framework é MVC (Model View Controller). Portanto, seguimos essa arquitetura padrão para o desenvolvimento da aplicação web.

A Model é responsável por manter às relações entre os Objetos e o Banco de dados. Também trata de validações, associações e transações.

A View está relacionada a como os dados serão apresentados ao usuário. Apesar disso, o Controller é que toma a decisão de qual será o formato de apresentação desses dados (ex: HTML, JSON,XML).

Controller redireciona o tráfego, faz consultas a models por dados específicos, organiza e processa esses dados para os enviá-los para uma determinada View.

Foi utilizada a estrutura de pastas padrão do Framework conforme pode ser visto abaixo.

Boa parte do código desenvolvido se encontra na pasta "app". A pasta "config" tem contém arquivos de configuração, como, por exemplo, configurações do banco de dados, web server e até mesmo as rotas utilizadas. O upload das músicas é feito na pasta “public/system/settings/mp3s”

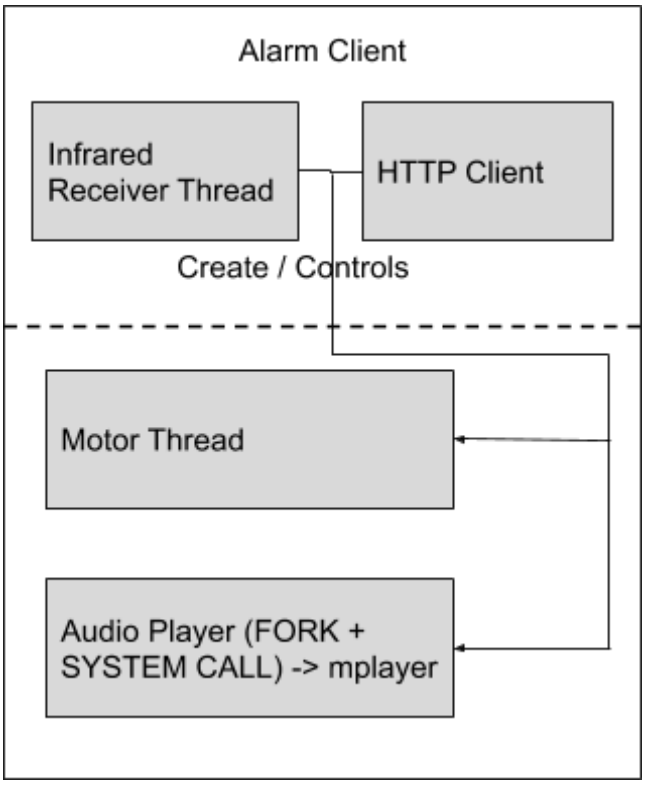
O webserver desenvolvido possui vários endpoints, porém somente 3 são utilizados pelos usuários.

- /
- /get_alarm
- /settings/1/edit

Para rodar o servidor foi utilizado docker. Portanto, basta instalá-lo e executar o comando "sudo docker-compose up".

- **Alarm Client**

O diagrama abaixo demonstra o que compõem o Alarm Client.



O Cliente HTTP e o Receptor de Infravermelho rodam infinitamente. Eles criam novas threads e processos que rodam temporariamente para controle do motor e o *streaming* de áudio. Além disso, eles têm controle sobre essas threads e processos (no caso do processo, ele apenas envia um sinal de SIGTERM para matar o processo responsável pelo streaming se o usuário desejar).

O Receptor infravermelho recebe os seguintes comandos:

- Motor para a direita (representado pelo botão >>, byte associado 0x44)
- Motor para a esquerda (representado pelo botão <<, byte associado 0x40)
- Parar motor (representado pelo botão >||, byte associado 0x43)
- Parar música (representado pelo botão CH+, byte associado 0x47)
- Aumentar Volume (byte associado 0x07).
- Diminuir Volume (byte associado 0x15).

O Cliente HTTP faz requisições para a url /get_alarm, e o protocolo de dados utilizado será descrito no próximo tópico.

- **Telegram Bot**

O bot simplesmente realiza o download do último áudio enviado pelo telegram. O áudio, posteriormente, é executado pelo sistema "Alarm Client" quando um alarme é disparado e o usuário seleciona a opção de "Lembrete".

B. Comunicação entre os Subsistemas de Software

O protocolo de comunicação utilizado é HTTP. Somente um *endpoint* implementado no servidor será utilizado pelo *Alarm Controller*. O nome dado a este *endpoint* foi /get_alarm. Ao realizar um *request* do tipo *GET*, o retorno é uma string que segue o seguinte padrão:

"segundos_para_abrir_a_cortina,segundos_para_tocar_o_som,url_do_som_a_ser_tocado".

É portanto uma string com três informações separadas por virgulas. Portanto, as informações enviadas consistem de:

- segundos_para_abrir_a_cortina: número inteiro, caso seja (-1), não deve abrir a cortina. Caso contrário, o valor representa em quanto tempo deve-se executar a funcionalidade.

- segundos_para_tocar_o_som: número inteiro, caso seja (-1), não deve tocar o som. Caso contrário, o valor representa em quanto tempo deve-se executar a funcionalidade.

- url_do_som_a_ser_tocado: string, indica onde está o som a ser tocado.

C. Comunicação entre os Subsistemas de Software e eletrônica.

Para tocar músicas, a comunicação será feita usando a entrada *audio jack* já presente na raspberry pi. O software utilizado para tocar a música será o *mplayer*.

Toda a comunicação com o motor e com os sensores é feita via escrita e leitura dos pinos digitais da raspberry.

7. RESULTADOS

A implementação de software e eletrônica do projeto funcionou conforme o desejado. Cada uma das partes do sistema (tanto a eletrônica como os subsistemas de software) foram testadas individualmente e, posteriormente, foi realizada a integração. Após cada integração verificou-se novamente o funcionamento do conjunto.

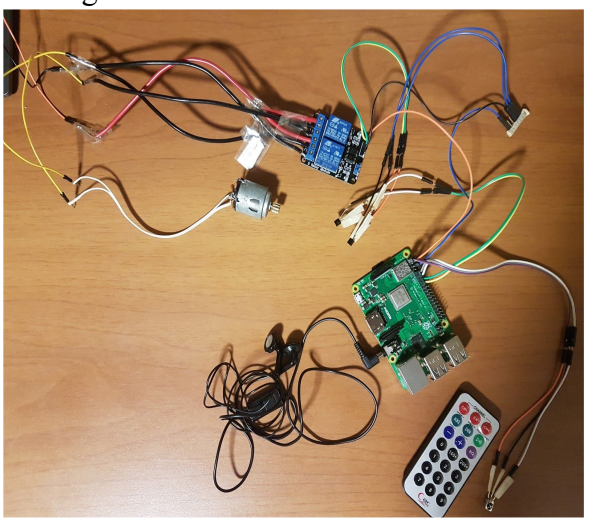
Os requisitos de software e eletrônica definidos foram alcançados e foram desenvolvidas mais funcionalidades que as estipuladas inicialmente. O maior problema encontrado neste projeto foi a implementação mecânica de movimentação da persiana. Deixou-se, então, o objetivo do projeto de abrir a persiana e passou-se a ter como objetivo o controle de motores e outros equipamentos como lâmpadas por meio do relé, o que foi alcançado.

Com o projeto atual é possível ligar e desligar 1 ou 2 lâmpadas e mover um motor em duas direções por meio do sistema de alarme.

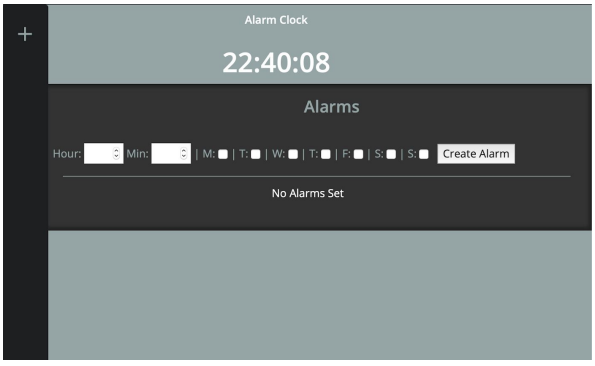
Ao finalizar o projeto, percebeu-se que deveriam ser utilizadas menos interfaces com o usuário, apesar de apenas uma ser a essencial e as demais opcionais. Este é o principal ponto de possível melhoria encontrado.

As imagens abaixo demonstram algumas partes do sistema.

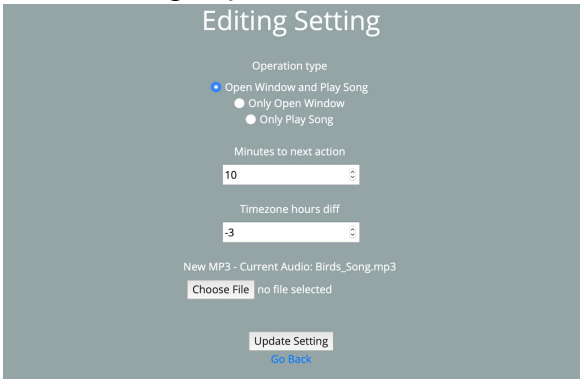
Visão geral do circuito:



Tela inicial do *Alarm Server* sem alarmes:

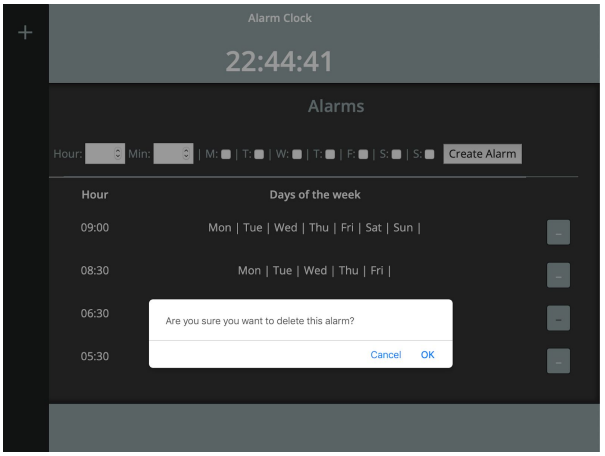


Tela de configurações do *Alarm Server*:

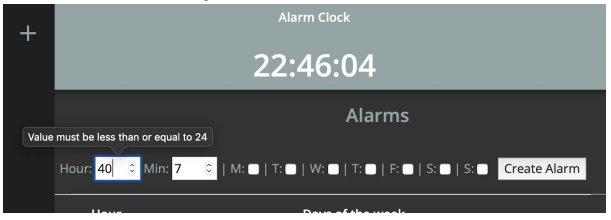


Tela inicial do *Alarm Server* após adicionar alarmes:

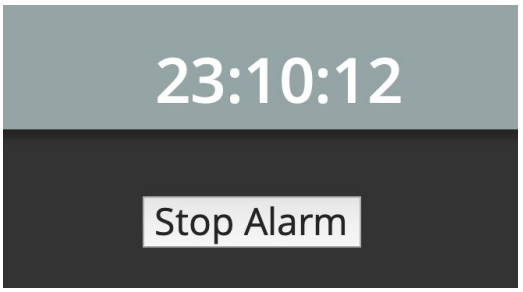
Tela para confirmar se deseja excluir o alarme :



Verificação se a hora/minuto é válido:



Foi adicionado um botão para parar a execução do áudio:



Streaming do alarme:

```
Playing http://localhost:3000/system/settings/mp3s/000/000/001/original/Birds_Song.mp3?1572226953.
Resolving localhost for AF_INET6...
Connecting to server localhost[::1]: 3000...

Cache size set to 320 KBytes
Cache fill:  0.00% (0 bytes)

Audio only file format detected.
==========================================================================
Opening audio decoder: [mpg123] MPEG 1.0/2.0/2.5 layers I, II, III
AUDIO: 48000 Hz, 2 ch, s16le, 320.0 kbit/20.83% (ratio: 40000->192000)
Selected audio codec: [mpg123] afm: mpg123 (MPEG 1.0/2.0/2.5 layers I, II, III)
==========================================================================
AO: [pulse] Init failed: Connection refused
Failed to initialize audio driver 'pulse'
AO: [alsa] 48000Hz 2ch s16le (2 bytes per sample)
Video: no video
Starting playback...
A:  31.7 (31.7) of 62.0 (01:02.0)  2.8% 41%
```

Valor hexadecimal referente ao infravermelho. Usado somente para debug.

```
44
40
43
47
```

8. CONCLUSÃO

A forma como um ser humano acorda pode influenciar bastante o restante do seu dia. A maior parte dos despertadores atuais possuem pouca customização, e não não possuem interfaces com dispositivos externos.

Através do uso do dispositivo proposto e de suas ferramentas dispostas, o usuário terá a possibilidade de um despertar mais tranquilo.

Além dos benefícios na saúde, o usuário também irá dispor da comodidade oferecida pela interface web, onde de qualquer lugar ele poderá fazer as modificações conforme desejar. Junto a isso, tem

as permissões de ajustes preferenciais de músicas e de opções do processo de despertar, visando a melhor adaptação do usuário.

Outra funcionalidade inovadora é a possibilidade do usuário substituir o som de um alarme por um lembrete por meio de uma gravação da sua própria voz.

O principal ponto de possível melhoria encontrado após finalizar o projeto é diminuir a quantidade de interfaces com o usuário e focar em apenas uma ou no máximo duas.

Outro problema encontrado no último ponto de controle foi a troca do horário interno da raspberry, porém isso pode ser consertado facilmente se ela possuir acesso à internet. Basta rodar um simples script (já desenvolvido) logo após a sua inicialização (no script de inicialização também já desenvolvido).

Os resultados esperados foram alcançados, o sistema se comportou conforme o desejado. Não além das melhorias citadas acima não foi encontrado nenhum outro problema com o projeto.

Por fim, o projeto presente mostra-se uma solução inovadora e moderna, que apesar de simples tem grande impacto na vida do usuário, que já pode começar o dia sem o estresse matinal de acordar no susto.

9. REFERÊNCIAS

1.RASPBERRYPI.ORG. **What is a Raspberry Pi?** Disponível em: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>. Acesso em: 28 ago. 2019.

2.OLIVEIRA, João Vitor. **A importância de dormir bem.** Revista Espaço Aberto, USP. 2012. Disponível em: <http://www.usp.br/espacoaberto/?materia=a-importancia-de-dormir-bem>. Acesso em: 19 set. 2019.

3.HARTMANN, Marcel. **Você dorme bem? Veja o que é considerado o 'sono de qualidade'.** O Estado de S. Paulo. 2017. Disponível em: <https://emais.estadao.com.br/noticias/bem-estar,voce-dorme-bem-veja-o-que-e-considerado-o-sono-de-qualidade,70001641118>. Acesso em: 20 set. 2019.

4.DINIZ, Lucilia. **O sono pela janela.** D Lucilia Diniz. Disponível em: <http://luciliadiniz.com/o-sono-pela-janela/>. Acesso em: 20 set. 2019.

5.FIGUEIRO, Mariana G. **The impact of daytime light exposures on sleep and mood in office workers.** SLEEP HEALTH. 2017. Disponível em: <https://www.sleephealthjournal.org/article/S2352-7218(17)30041-4/fulltext>. Acesso em: 20 set. 2019.

6.GALLAGHER, James. **Como luz natural regula relógio biológico e pode combater insônia.** BBC News Brasil. 2017. Disponível em: <https://www.bbc.com/portuguese/geral-38885424>. Acesso em: 20 set. 2019.

6.FREECODECAMP, **Understanding the basics of Ruby on Rails: HTTP, MVC, and Routes**. Disponível em: <https://www.freecodecamp.org/news/understanding-the-basics-of-ruby-on-rails-http-mvc-and-routes-359b8d809c7a/>. Acesso em: 01 nov. 2019.

10. APÊNDICE (PARTE DO CÓDIGO)

O código abaixo refere-se somente ao subsistema ALARM CLIENT. Os códigos dos demais subsistemas podem ser encontrados no github https://github.com/AlexandreTK/Sistemas_Embarcados/tree/master/3_Trabalho.

```
#include <stdio.h>
#include <unistd.h>
#include <curl/curl.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>
#include <wiringPi.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>
```

```
//************* SERVER DEFINITION ***************
#define SECONDS_NEXT_REQUEST 2
//static char GET_ALARM_INFO_URL[] =
"http://raspberrypi.local:3000/get_alarm";
static char GET_ALARM_INFO_URL[] =
"http://localhost:3000/get_alarm";

static const char MUSIC_PLAYER[] = "mplayer";
#ifdef __APPLE__
static const char MUSIC_PLAYER_PATH[] =
"/usr/local/bin/mplayer";
#else
static const char MUSIC_PLAYER_PATH[] = "/usr/bin/mplayer";
#endif

static char REMINDER_PATH[] = "./telegram_bot/voice.ogg";

pid_t fork_song_pid = 0;
//*****************************************
#define CANCEL_SONG -2
#define PLAY_REMINDER -10
#define NEXT_ALARM_DELAY 20


//************* MOTOR DEFINITION ***************
#define RELAY_1 2
#define RELAY_2 3
#define RELAY_ENABLE  HIGH
#define RELAY_DISABLE LOW
//*****************************************


//************* IR DEFINITION ***************
#define ERROR 0xFE
#define PIN_IR 0 //GPIO17 WPI_PIN0

#define FULL_CYCLE_USEC 110000
#define SLEEP_USEC 10
//*****************************************



void  SIGTERM_fork_handler(int sig) {
    //system("ps -a");
    //system("echo ------ ");
    fprintf(stderr, "Killing mplayer\n");
    system("killall mplayer");
    sleep(2);
    //system("ps -a");
    fprintf(stderr, "(FORK) Exit from SIGTERM\n");
    exit(0);
}


//************* SERVER CODE ***************
struct url_data {
  size_t size;
  char* data;
};

size_t write_data(void *ptr, size_t size, size_t nmemb, struct url_data
*data) {
  size_t index = data->size;
  size_t n = (size * nmemb);
  char* tmp;

  data->size += (size * nmemb);

#ifdef DEBUG
      fprintf(stderr, "data at %p size=%ld nmemb=%ld\n", ptr, size,
nmemb);
#endif
  tmp = realloc(data->data, data->size + 1);

  if(tmp) {
    data->data = tmp;
  } else {
    if(data->data) {
      free(data->data);
    }
    fprintf(stderr, "Failed to allocate memory.\n");
    return 0;
  }

  memcpy((data->data + index), ptr, n);
  data->data[data->size] = '\0';

  return size * nmemb;
}

char *handle_url(char* url) {
  CURL *curl;

  struct url_data data;
  data.size = 0;
  data.data = malloc(4096); /* reasonable size initial buffer */
  if(NULL == data.data) {
```

```
        fprintf(stderr, "Failed to allocate memory.\n");
        return NULL;
    }

    data.data[0] = '\0';

    CURLcode res;

    curl = curl_easy_init();
    if (curl) {
        curl_easy_setopt(curl, CURLOPT_URL, url);
                    curl_easy_setopt(curl,   CURLOPT_WRITEFUNCTION,
write_data);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &data);
        res = curl_easy_perform(curl);
        if(res != CURLE_OK) {
                fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));
        }

        curl_easy_cleanup(curl);

    }
    return data.data;
}
//*****************************************



//************* MOTOR CODE ***************
void * run_motor(void *voidData) {
    digitalWrite(RELAY_1, RELAY_ENABLE);
    digitalWrite(RELAY_2, RELAY_DISABLE);
    sleep(10);
    digitalWrite(RELAY_1, RELAY_DISABLE);
    digitalWrite(RELAY_1, RELAY_DISABLE);

    return NULL;
}
//*****************************************



//************* IR CODE ***************
long getMicrotime(){
    struct timeval currentTime;
    gettimeofday(&currentTime, NULL);
    return currentTime.tv_sec * (int)1e6 + currentTime.tv_usec;
}

bool IRStart() {
    long timeRisingEdge = 0;
    long timeFallingEdge[2] = {0, 0};
    long timeSpan[2] = {0, 0};
    // Start signal is composed with a 9 ms leading space and a 4.5 ms
pulse.

    while(digitalRead(PIN_IR) == 1) {
            usleep(SLEEP_USEC);
    }
    timeFallingEdge[0] = getMicrotime();
    while(digitalRead(PIN_IR) == 0) {
            usleep(SLEEP_USEC);
    }
    timeRisingEdge = getMicrotime();
    while(digitalRead(PIN_IR) == 1) {
            usleep(SLEEP_USEC);
    }
    timeFallingEdge[1] = getMicrotime();

    timeSpan[0] =  timeRisingEdge - timeFallingEdge[0];
    timeSpan[1] =  timeFallingEdge[1] - timeRisingEdge;

    if ((timeSpan[0] > 8500) && (timeSpan[0] < 9500) &&
(timeSpan[1] > 4000) && (timeSpan[1] < 5000) ) {
            return true;
    } else {
            return false;
    }

    //return false;
}

char getByte() {
    char byte = 0;
    long timeRisingEdge = 0;
    long timeFallingEdge = 0;
    long timeSpan = 0;
    // Logic '0' == 0.56 ms LOW and 0.56 ms HIGH
    // Logic '1' == 0.56 ms LOW and 0.169 ms HIGH
    int i;
    for(i=0; i<8; i++) {

            while(digitalRead(PIN_IR) == 0) {
                    usleep(SLEEP_USEC);
            }
            timeRisingEdge = getMicrotime();
```

```
        while(digitalRead(PIN_IR) == 1) {
                usleep(SLEEP_USEC);
        }
        timeFallingEdge = getMicrotime();

        timeSpan = timeFallingEdge - timeRisingEdge;

        if ((timeSpan > 1600) && (timeSpan < 1800)) {
                byte |= 1 << i;
        }
    }
    return byte;
}

char getKey() {
    char bytes[4]= {0, 0, 0, 0};
    // bytes[0] -> ADDRESS
    // bytes[1] -> LOGICAL INVERSE OF ADDRESS
    // bytes[2] -> COMMAND
    // bytes[3] -> LOGICAL INVERSE OF COMMAND
    if (IRStart() == false) {
        usleep(FULL_CYCLE_USEC); // One message frame lasts 108 ms.
        return ERROR;
    } else {
        int i;
        for(i=0; i<4; i++) {
                bytes[i] = getByte();
        }
        if ((bytes[0]+bytes[1]==0xFF) && (bytes[2]+bytes[3]==0xFF)) {
                return bytes[2];
        } else {
                return ERROR;
        }
    }
}

void * run_infrared(void *voidData) {
    char key;
    while(1) {
        key = getKey();
        //printf("Raw %x\n", key);

        if(key != ERROR) {
                printf("%x\n", key);
        }
        if(key == 0x44) {
                digitalWrite(RELAY_2, RELAY_DISABLE);
                digitalWrite(RELAY_1, RELAY_ENABLE);
        }
        if(key == 0x40) {
                digitalWrite(RELAY_1, RELAY_DISABLE);
                digitalWrite(RELAY_2, RELAY_ENABLE);
        }
        if(key == 0x43) {
                digitalWrite(RELAY_1, RELAY_DISABLE);
                digitalWrite(RELAY_2, RELAY_DISABLE);
        }
        if(key == 0x47) {
                if (fork_song_pid!=0) {
                        fprintf(stderr, "Stopping alarm pid %d from IR\n", fork_song_pid);
                        //printf("Stopping alarm\n");
                        if (kill(fork_song_pid,0) == 0) {
                                while( kill(fork_song_pid, SIGTERM) != 0 ) {
                                        fprintf(stderr, "error to kill\n");
                                        sleep(2);
                                }
                                //sleep(2);
                                //system("ps -a");
                                fork_song_pid = 0;
                        } else {
                                fprintf(stderr, "pid do not exists");
                        }
                }
        }
        if (key == 0x07) {
                system("amixer set PCM,0 $(expr $(amixer get PCM | grep -o [0-9]*% | sed 's/%//') - 5)%");
                system("( speaker-test -t sine -c 2 -s 2 -f 800 & TASK_PID=$! ; sleep 0.09 ; kill -s 2 $TASK_PID ) > /dev/null");
        }

        if (key == 0x15) {
                system("amixer set PCM,0 $(expr $(amixer get PCM | grep -o [0-9]*% | sed 's/%//') + 5)%");
                system("( speaker-test -t sine -c 2 -s 2 -f 800 & TASK_PID=$! ; sleep 0.09 ; kill -s 2 $TASK_PID ) > /dev/null");
        }
    }
    return NULL;
}
//***********************************
```

```
int main(int argc, char* argv[]) {
    // SERVER
    //pid_t fork_song_pid = 0;
    char* data = NULL;
    int play_song_secs;
    int move_motor_secs;
    char song_to_play_path[200];
    // MOTOR
    pthread_t thread_motor;
    // INFRARED
    pthread_t thread_infrared;
    //char key;



    wiringPiSetup();
    // MOTOR
    pinMode(RELAY_1, OUTPUT);
    pinMode(RELAY_2, OUTPUT);
    digitalWrite(RELAY_1, RELAY_DISABLE);
    digitalWrite(RELAY_2, RELAY_DISABLE);
    // INFRARED
    pinMode(PIN_IR, INPUT);
    pinMode(PIN_IR, PUD_UP);


    if (pthread_create(&thread_infrared, NULL, run_infrared, NULL))
{
        fprintf(stderr, "An error occured while creating new thread");
        return 1;
    }

    /*
    if (signal(SIGINT, SIGINT_handler) == SIG_ERR) {
      printf("SIGINT install error\n");
      exit(1);
    }*/

    while(1) {

        data = handle_url(GET_ALARM_INFO_URL);

        if (!data[0]) {
            fprintf(stderr, "ERROR - no data received\n");
            sleep(SECONDS_NEXT_REQUEST);
            continue;
        }

        sscanf(data, "%d,%d,%s", &move_motor_secs,
&play_song_secs, song_to_play_path);
        free(data);

        //printf("%d,", play_song_secs);
        //printf("%d,", move_motor_secs);
        //printf("%s\n", song_to_play_path);

        if (move_motor_secs>=0) {
            if (pthread_create(&thread_motor, NULL,
run_motor, NULL)) {
                fprintf(stderr, "An error occured while
creating new thread");
                return 1;
            }
        }




        if (play_song_secs >= 0) {

            // KILL OLD PROCESS
            if (fork_song_pid!=0) {
                fprintf(stderr, "Stopping alarm pid %d \n",
fork_song_pid);
                if (kill(fork_song_pid,0) == 0) {
                    while( kill(fork_song_pid,
SIGTERM) != 0 ) {
                        fprintf(stderr, "error to
kill\n");
                        sleep(2);
                    }
                    //sleep(2);
                    //system("ps -a");
                    fork_song_pid = 0;
                } else {
                    fprintf(stderr, "pid do not
exists");
                }
            }


        fork_song_pid = fork();
        if (fork_song_pid == 0) {
            sleep(play_song_secs);
            //execl(MUSIC_PLAYER_PATH,
MUSIC_PLAYER, song_to_play_path, NULL);
            if (signal(SIGTERM,
SIGTERM_fork_handler) == SIG_ERR) {
```

```
                                        fprintf(stderr,          "SIGTERM
install error\n");
                                        exit(1);
                                }

                                char song_to_play_command[200];
                                sprintf(song_to_play_command, "%s %s",
MUSIC_PLAYER, song_to_play_path);
                                while(true) {

system(song_to_play_command);


sleep(NEXT_ALARM_DELAY);
                                }

                                return 0;
                        }

                }

                if (play_song_secs==PLAY_REMINDER) {

                        // KILL OLD PROCESS
                        if (fork_song_pid!=0) {
                                fprintf(stderr, "Stopping alarm pid %d \n",
fork_song_pid);
                                if (kill(fork_song_pid,0) == 0) {
                                        while(          kill(fork_song_pid,
SIGTERM) != 0 ) {
                                                fprintf(stderr, "error to
kill\n");
                                                sleep(2);
                                        }
                                        //sleep(2);
                                        //system("ps -a");
                                        fork_song_pid = 0;
                                } else {
                                        fprintf(stderr,    "pid    do    not
exists");
                                }
                        }


                        fork_song_pid = fork();
                        if (fork_song_pid == 0) {
                                //execl(MUSIC_PLAYER_PATH,
MUSIC_PLAYER, REMINDER_PATH, NULL);
                                if                      (signal(SIGTERM,
SIGTERM_fork_handler) == SIG_ERR) {
```

```
                                        fprintf(stderr,          "SIGTERM
install error\n");
                                        exit(1);
                                }

                                char song_to_play_command[200];
                                sprintf(song_to_play_command, "%s %s",
MUSIC_PLAYER, REMINDER_PATH);
                                while(true) {


system(song_to_play_command);


sleep(NEXT_ALARM_DELAY);
                                }




                                return 0;
                        }

                }

                if (play_song_secs==CANCEL_SONG) {
                        fprintf(stderr,    "Stopping    alarm    pid    %d    from
SERVER\n", fork_song_pid);

                        if (kill(fork_song_pid,0) == 0) {
                                while( kill(fork_song_pid, SIGTERM) !=
0 ) {
                                        fprintf(stderr, "error to
kill\n");
                                        sleep(2);
                                }
                                //sleep(2);
                                //system("ps -a");
                                fork_song_pid = 0;
                        } else {
                                fprintf(stderr, "pid do not exists");
                        }
                }




                sleep(SECONDS_NEXT_REQUEST);
        }

                //pthread_join(thread_motor, NULL);
```

```
        return 0;
}
```