Automação residencial: Comandando lâmpadas pelo *Telegram*

Parley Martins - 11/0038096 Tatielen Pereira - 12/0136074

Resumo—Este trabalho propõe a utilização de automação residencial para possibilitar ao usuário acender e apagar lâmpadas remotamente, utilizando integração com aplicativo no celular.

Index Terms—Automação Residencial, Telegram, bot, *smart house*

1 Introdução

Automação residencial é resultado da combinação de espaços residenciais, como sala, banheiro, quarto com tecnologias, para maior conforto, segurança, ou menos contato humano [1]. Estas tecnologias e ideias eram, até recentemente, consideradas sonhos de um futuro distante [2], sem uso prático, exceto no entretenimento.

No entanto com um mundo conectado pela internet, que mudou o jeito que as pessoas se comunicam e se relacionam, é normal que este conceito esteja cada vez mais próximo da realidade das pessoas. Para ter mais conforto, já é possível controlar pelo celular o volume das televisões (e outros aparelhos de som), o canal em que se está, a intensidade com que aparelhos devem funcionar, entre outras comodidades. Para ter mais segurança, é possível controlar luzes, sistema de alarmes, de detecção de movimentos, etc. Existem diversas empresas que fornecem esse tipo de serviço, mas eles ainda podem ter um custo muito elevado.

2 Solução

Para facilitar e desmistificar o acesso à automação residencial, a proposta deste projeto é implementar um sistema que possa controlar remotamente as lâmpadas de uma casa. O usuário, após instalação do sistema físico, poderá utilizar seu *smartphone* para ligar e desligar as lâmpadas.

A interação com o usuário se dará através de um bot no aplicativo *Telegram*. Deve-se iniciar uma 'conversa' com o bot, e mandar o comando desejado (ligar ou desligar, por exemplo). Este irá mandar para o módulo wifi do sistema, que fará a comunicação com o MSP, desligando ou ligando a lâmpada selecionada.

Para fins deste trabalho, uma lâmpada e uma fonte de energia externas, controladas pela protoboard, serão utilizadas para facilitar a instalação e testes.

O *hardware* será composto, inicialmente, pelos seguintes items:

- protoboard, para execução do sistema;
- microcontrolador MSP430, irá executar o controle da energia na lâmpada;
- módulo esp8266, proverá o acesso à rede wifi;
- lâmpada, para testes;
- fonte de energia, tanto para o microcontrolador quanto para a lâmpada.

O *software* embarcado no microcontrolador será escrito nas linguagens C e Assembly, enquanto o código do *bot* será desenvolvido utilizando Python 3. Os serviços serão conectados através do IFTTT, que conecta servidores de terceiros a outros serviços [3].

2.1 Requisitos

O *software* do microcontrolador deve corretamente identicar os comandos e apagar ou acender a lâmpada, conforme instrução recebida.

O *bot*, *software* que responde a comandos pré definidos automaticamente, deve ser integrado ao aplicativo *Telegram* e deve mandar instruções de ligar e de desligar a lâmpada.

O sistema completo, tanto hardware quanto *software*, deve ter acesso à internet para o funcionamento correto.

2.2 Benefícios

Este projeto tem como principal beneficiário o cidadão comum que quer ter um pouco do conforto que a automação residencial traz a sua casa. Além disso, ajudará na economia de energia, já que a pessoa pode mandar um comando de apagar determinada luz, mesmo a distância.

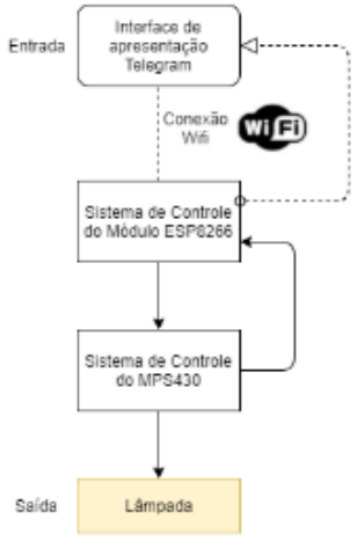

Figura 1. Diagrama de blocos do projeto

3 Desenvolvimento

Esta seção apresenta um detalhamento do sistema proposto no projeto. Para isto, será apresentado a descrição do hardware e software.

3.1 Hardware

O Hardware utilizado pelo grupo consiste nos items descritos na Tabela 1

Tabela 1
Tabela de componentes do projeto

Componentes	Quantidade
MSP430G2553	1
Módulo Wireless ESP8266	1
Protoboard	1
Relé	1
Diodo	1
Lâmpada Led 12w	1
Jumpers	Indefinido

O diagrama de blocos tem como objetivo apresentar o princípio de funcionamento do sistema. Os blocos apresentados de acordo com a Figura 1 são:

- Interface de apresentação será o meio de comunicação entre o usuário e o sistema. Com o auxílio do aplicativo Telegram será possível escolher se o usuário acenderá, apagará ou verificar o estado em que a lâmpada se encontra.
- Sistema de controle do módulo ESP8266 receberá a decisão enviada via aplicativo por conexão wifi para maior mobilidade do usuário. Como o usuário possui a possibilidade de verificar o estado da lâmpada o sistema de controle do módulo Wifi também envia um sinal de retorno para o app Telegram.
- Sistema de controle do MSP430 será ligado junto ao módulo Wifi, onde será recebida e executada a decisão do usuário. Como o usuário possui a possibilidade de verificar o estado da lâmpada sistema de controle do MSP430 também envia um sinal de retorno para o módulo Wifi.
- Lâmpada será acesa dependendo da decisão do usuário.

Para conectar o módulo Wifi ESP8266 ao MSP430 foram conectados os seguintes pinos, de acordo com a Tabela 2. A lâmpada até o momento só foi testada com o LED 1 do próprio MSP.

O ESP foi configurado para se conectar à internet, mas não foi possível fazê-lo mandar

Tabela 2
Tabela de conexoes MSP-ESP

Módulo Wifi ESP8266	MSP430
TX	P1.4
CH_PD	VCC
RST	
VCC	VCC
GND	GND
GPIO2	
GPIO0	
RX	P1.3

dados para o MSP.

3.2 Software

O MSP foi configurado para receber um caractere na conexão UART que troca ou lê o estado da lâmpada. Os caracteres aceitos são `n`, para ligar; `f`, para desligar; e `s` para ler e retornar o estado atual da lâmpada. Qualquer outro input resultará em nenhum retorno ou ação do MSP. O código 1 do Anexo A inicializa o modo de comunicação UART com *baud rate* 9600 e *clock* em 1MHz. Caso algum dado seja recebido, uma interrupção do RX do UART será acionada. Esta interrupção lerá o valor recebido e tomará uma ação de acordo com o explicado acima. Para substituir a lâmpada, por enquanto, o LED conectado ao pino P1.6 do MSP está sendo utilizado.

O *bot* do Telegram foi feito para responder à comandos pré-definidos. Após o uso de um deles, o *bot* responde o usuário com uma mensagem de texto ou emoji, para demonstrar que o MSP recebeu e executou o comando, como demonstrado na Figura 2.

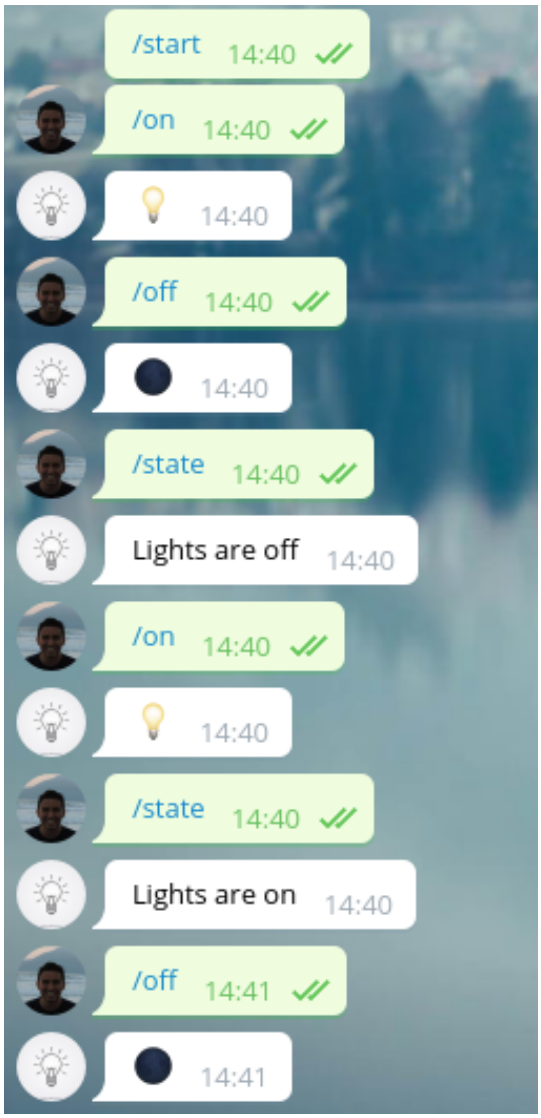

Figura 2. Comandos do Telegram

- `/on`, vai ligar a lâmpada, retornando um emoji de lâmpada acesa.
- `/off` desliga a lâmpada e retorna uma lua nova (pra demonstrar que a lâmpada foi apagada).
- `/state` retorna o estado do lâmpada, com a mensagem "Light is on/off".

Para o Código 2 do *bot*, a biblioteca `python-telegram-bot`, que é um *wrapper* da API do Telegram para fazer bots escrito em Python, foi utilizada. Primeiramente alguns *imports* são feitos e o log é definido. A função `send_message` recebe um caractere que será enviado ao MSP. A conexão serial é aberta, o caractere é enviado e a conexão é fechada. A função `read_return_message` faz o mesmo da anterior, mas lê o retorno da mensagem mandada. As três próximas funções, `turn_on`, `turn_off` e `check_state` são para gerenciar os comandos do *bot*. Elas mandam uma mensagem para o MSP e depois uma para o usuário com o retorno da comunicação serial ou uma mensagem indicando que a ação foi executada. Os comandos são adicionados ao *dispatcher* do *bot*.

4 Resultados

Para realizar a comunicação entre a internet e o MSP foi escolhido o modulo ESP 8266, no entanto, até este ponto do trabalho, a conexão entre eles não está totalmente pronta, pois comandos realizados em relação ao módulo wifi não estão gerando respostas satisfatórias. O ESP parece se conecta a internet, mas não devolve informações simples como o IP.

5 Conclusão

Anexo A - Código MSP

Listing 1. `lamp.c`

```
#include <msp430g2553.h>
#include <legacymsp430.h>

#define RX BIT1
#define TX BIT2
#define LAMP BIT6

void send_data(unsigned char c);
void init_uart();

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;

  P1OUT &= ~LAMP;
  P1DIR |= LAMP;

  init_uart();
  _BIS_SR(GIE);
  return 0;
}

void send_data(unsigned char c)
{
  while((IFG2&UCA0TXIFG)==0);
  UCA0TXBUF = c;
}


void init_uart()
{
  P1SEL2 = P1SEL = RX + TX;
  UCA0CTL0 = 0; //UART, 8bits, no parity,
  UCA0CTL1 = UCSSEL_2; // SMCLK
  UCA0BR0 = 104; //Baud rate: 9600
  UCA0BR1 = 0;
  UCA0MCTL = UCBRF_0 + UCBRS_1; //Baud rate:
    9600
  IE2 |= UCA0RXIE; // Set interrruption by
    UART data arrival
}

void send_state(char state[]){
  int i = 0;

  for(i = 0; state[i] != '\0'; i++){
    send_data(state[i]);
  }
  send_data('\n');
}

interrupt(USCIAB0RX_VECTOR) set_lamp_state(
    void){
  unsigned char state = UCA0RXBUF;
  if(state == 'n'){ // turn the light on
    P1OUT |= LAMP;
  } else if(state == 'f') { // turn the light
    off
    P1OUT &= ~LAMP;
  } else if(state == 's') { // check light
    state
    if((P1OUT&LAMP)==0){
      send_state("off");
    } else{
      send_state("on");
    }
  }
}
```

Anexo B - Código Telegram

Listing 2. `bot.py`

```
from telegram.ext import Updater,
    CommandHandler
from telegram import KeyboardButton,
    ReplyKeyboardMarkup, ReplyKeyboardRemove
from telegram.ext.filters import Filters
from emoji import emojize


import logging
import json
import serial

import serial

configs = {}

with open('.conf', 'r') as f:
  configs = json.loads(f.read())

logging.basicConfig(level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(
    levelname)s - %(message)s')

def send_message(msg):
  ser = serial.Serial(configs['port'])  # open
    serial port
  ser.write(str.encode(msg))
  ser.close()

def read_return_message(msg):
  ser = serial.Serial(configs['port'])  # open
    serial port
  ser.write(str.encode(msg))
  ser.flush()
  ret = ser.readline()
  ser.close()
  return ret.decode()
```

```
32
34 def turn_on(bot, update):
    send_message('n')
36  bot.send_message(text=emojize(':bulb:',
      use_aliases=True), chat_id=update.message.
      chat_id)

38 def turn_off(bot, update):
    send_message('f')
40  bot.send_message(text=emojize(':new_moon:',
      use_aliases=True), chat_id=update.message.
      chat_id)

42 def check_state(bot, update):
    state = read_return_message('s')
44  bot.send_message(text='Lights are {}'.format
      (state), chat_id=update.message.chat_id)


46
 updater = Updater(token=configs['token'])
48
 # updater.bot.send_message
50 updater.dispatcher.add_handler(CommandHandler(
     'on', callback=turn_on))
 updater.dispatcher.add_handler(CommandHandler(
     'off', callback=turn_off))
52 updater.dispatcher.add_handler(CommandHandler(
     'state', callback=check_state))


54
 updater.start_polling()
56 updater.idle()
```

REFERÊNCIAS

[1] F. Moraes, A. Amory, N. Calazans, E. Bezerra, and J. Petrini, "Using the can protocol and reconfigurable computing technology for web-based smart house automation," in *Integrated Circuits and Systems Design, 2001, 14th Symposium on.* IEEE, 2001, pp. 38–43.

[2] A. GhaffarianHoseini, N. D. Dahlan, U. Berardi, A. GhaffarianHoseini, and N. Makaremi, "The essence of future smart houses: From embedding ict to adapting to sustainability principles," *Renewable and Sustainable Energy Reviews*, vol. 24, pp. 593 – 607, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032113001342

[3] S. Ovadia, "Automate the internet with "if this then that"(ifttt)," *Behavioral & social sciences librarian*, vol. 33, no. 4, pp. 208–211, 2014.