ID: Daniel Costa 112185

Projeto 01 de Sistemas Embarcados: simulação de automação residencial

São José dos Campos - Brasil

2019

ID: Daniel Costa 112185

Projeto 01 de Sistemas Embarcados: simulação de automação residencial

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Sistemas Embarcados.

Docente: Prof. Dr. Sérgio Ronaldo Barros dos Santos

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

2019

Resumo

O projeto em questão teve como finalidade implementar, de modo à simular, uma automação residencial através de uma placa Arduino Mega junto a interface de desenvolvimento provida pelo Arduino.

Palavras-chaves: Arduíno. Embarcados. Automação. Residência.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com a evolução da tecnologia, muitas tarefas são deixadas de serem realizadas por humanos, sendo passadas para uma tecnologia capaz de automatizar processos ou, ao menos, facilitar esse. A residência não foge desse padrão, considerando tempos prévios aos anos 2000, pode-se dizer que, para a época, a aparição do controle remoto em televisões tenha seguido a linha de uma automação.

A automação residencial pode oferecer diversos benefícios desde que aplicada de maneira correta. Diversos dispositivos podem ser empregados para reduzir consumo de energia, baseado na luminosidade por exemplo, melhorar a segurança, como sensores de movimento, e, talvez a maior motivação para a procura de uma automação residencial, o menor gasto e preocupação com diversas atividades rotineiras, agora feitas por dispositivos.

2 Objetivos

2.1 Geral

O objetivo desse projeto visa a simulação, utilizando uma placa Arduíno, composta por um microcontrolador ATmega 2560, junto à sua *integrated development environment* (IDE), de uma automação residencial. Não haverá portanto a implementação real em uma rede residencial.

2.2 Específico

- Compreender os dispositivos à serem usados e definir a simulação a ser realizada em cada dispositivo.
- Implementar o algoritmo.
- Simular o funcionamento.

3 Fundamentação Teórica

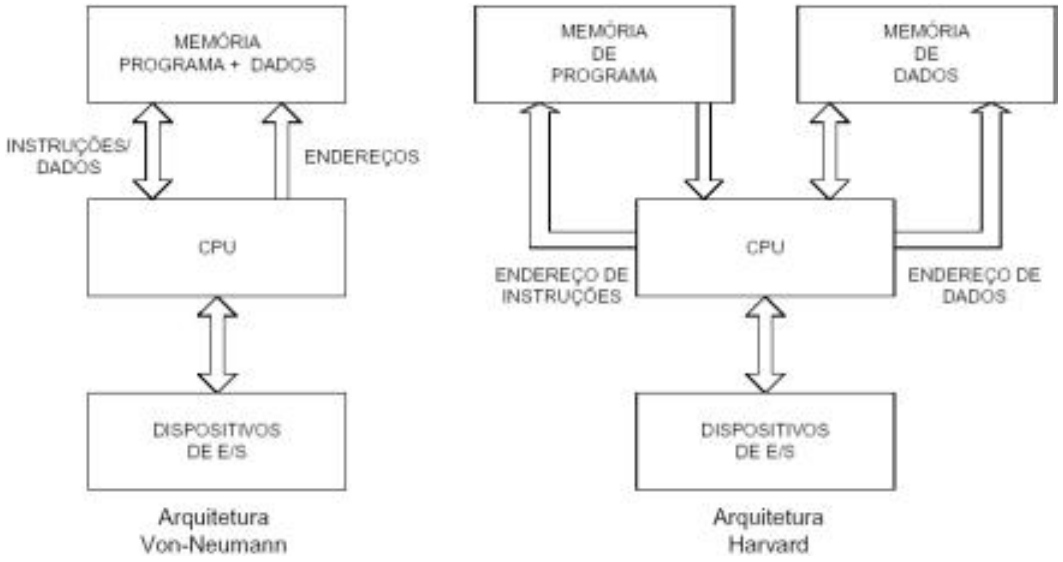
Alguns dispositivos, mais comuns, não serão relatados, como resistores, leds, botões, cabos, entre outros.

3.1 Modelos de arquitetura

Como principais modelos de estudo, temos as arquiteturas de Von Neumann e de Harvard. A primeira trata-se de uma arquitetura baseada em uma organização com uma única unidade de memória, tanto para dados quanto para instruções, além de possuir a unidade lógica e aritmética e uma unidade de controle. Já a arquitetura de Harvard diferencia-se da de Von Neumann na questão da memória, já que divide a memória em duas, uma para dados e outra para instruções.

A unidade de controle tem como função controlar os outros componentes, por meio de sinais que geram bloqueios ou não a um determinado sinal. A unidade lógica e aritmética é responsável por realizar as operações de lógica e aritmética, normalmente controlada por meio de sinais vindos da unidade de controle. Tem-se ainda os módulos de entrada e saída.

Figura 1 – Arquiteturas Von Neumann e Harvard



Fonte: Os poderosos microcontroladores AVR (IFSC)

Há ainda um outro modelo de arquitetura, conhecido como arquitetura Harvard modificada, essa arquitetura apresenta-se assim como uma Harvard, porém, sua memória

programa, ou de instrução, pode ser utilizada como uma memória de dados caso necessário. Tal arquitetura é a utilizada nas placas Arduíno.

3.2 Placa Arduíno composta por ATmega2560

3.2.1 Microcontrolador e pinos

As placas Arduíno apresentam microcontroladores da família AVR, mais especificamente, megaAVR, apresentando como características uma memória flash de 8kB à 256kB, 4kB de EEPROM e 8kB de SRAM, entre 23 e 86 pinos de E/S, comparadores e canais PWM, ADC, I2C, USART e SPI.

Os Arduínos que apresentam o microcontrolador ATmega2560, possuem 54 pinos de entradas e saídas digitais (DPIO), sendo que 15 podem operar como pinos de modulação de largura de pulso (PWM), 16 pinos analógicos, além de pinos para comunicações Serial, *Serial peripheral interface protocol* (SPI) e *I2C*. Os pinos para realizar comunicação podem ser emulados em pinos não definidos para esse popósito com bibliotecas. Apresenta ainda pinos para alimentação.

3.2.2 Comunicação Serial

A comunicação Serial é utilizada para comunicação entre a placa Arduino e outros dispositivos, como o computador, os pinos são descritos como RX e TX na placa. A comunicação Serial feita pelo Arduino Mega é do tipo USART (Universal Synchronous/Asynchronous Receiver/Transmitter). A transmissão dos dados ocorre com a entrada de dados paralelos, os quais são convertidos em bits através de uma onda quadrada, tendo seu limite marcado por bits de inicialização e término de dados.

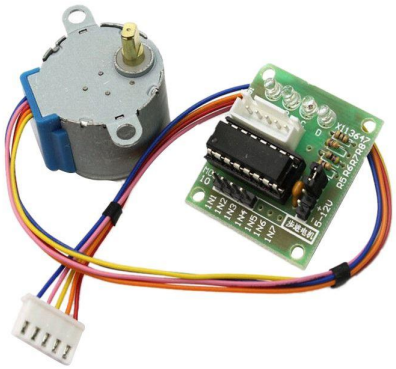
3.3 Dispositivos

No projeto foram usados os seguintes componentes, com exceção à placa Arduino MEGA:

- Motor de passo unipolar 28byj-48 junto ao driver uln2003.
- *Display* LCD 16x2
- Sensor PIR HC-SR501
- Sensor ultrassônico HC-SR04
- Módulo Bluetooth HC-05

3.3.1 Motor de passo

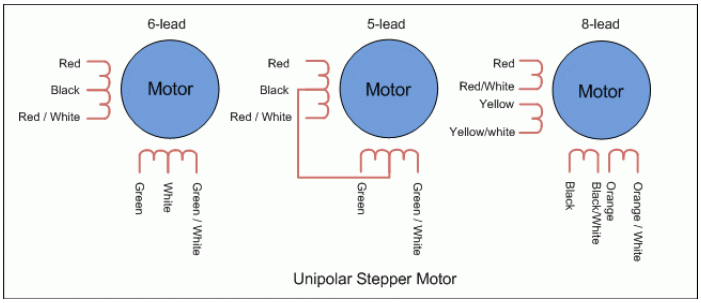
Figura 2 – Motor de passo unipolar 28byj-48 com driver uln2003



Fonte: Stepper motor

Um motor de passo trata-se de um motor formado por bobinas que geram um campo capaz de atrair a peça central ao motor a fim de rotacional, via engrenagens, o motor. O passo de um motor ocorre quando a troca de posição de um extremidade da peça central para uma bobina vizinha. Os motores, assim, variam em quantidade de passos para completar uma volta. A fase corresponde ao estado de cada bobina.

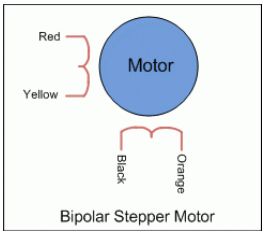
Figura 3 – Motor de passo unipolar



Fonte: Unipolar Stepper Motor vs Bipolar Stepper Motors

Os motores unipolares são constituídos por bobinas que apresentam um fio central em cada, ou há a divisão da bobina em duas. A grande vantagem em se aplicar esse modelo está na simplicidade de trabalho para realizar um passo. Normalmente o fio central é alimentado com VCC enquanto as extremidades da bobinas são aterradas conforme necessário, como as extremidades são opostas, ao se aterrar uma, a corrente se mantém de modo a manter o sentido de rotação desejado, não havendo necessidade da inversão do polo magnético via inversão de corrente.

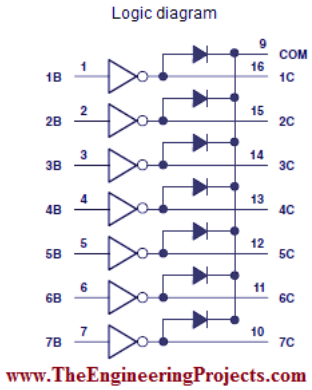
Figura 4 – Motor de passo bipolar



Fonte: Unipolar Stepper Motor vs Bipolar Stepper Motors

Os motores bipolares não apresentam o terminal central como nos unipolares, ou seja, o circuito torna-se mais complexo devido a necessidade da inversão de corrente para manter a rotação desejada, requerendo utilização de circuitos Ponte-H. Porém, esse motor apresenta um maior torque frente ao unipolar.

Figura 5 – Circuito integrado uln2003

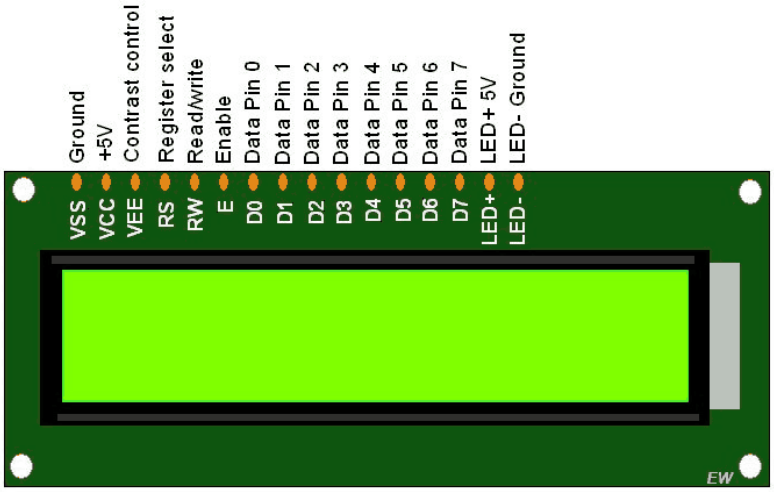


Fonte: Introduction to ULN2003

O módulo uln2003 nada mais é que um circuito integrado que apresenta portas *NOT* junto à diodos. a finalidade do módulo é gerar a inversão do valor das fases, já que ao programar, considera-se o fio comum nos motores unipolares como valor baixo e o valor da bobina como valor alto.

3.3.2 *Display LCD*

Figura 6 – LCD 16x2



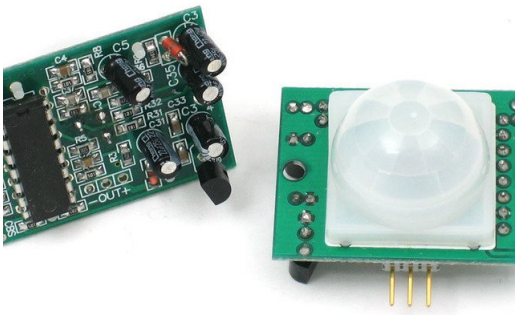
Fonte: LCD 16x2 custom character display using LPC2148

O *display* LCD possui várias matrizes de pontos a fim de gerar um símbolo. Uma memória ROM interna armazena caracteres alfabéticos, fazendo a conversão dos dados enviados em bits para um caractere específico. Possuem um RAM de armazenamento de dados de 80 bytes, sendo capaz de armazenar 80 caracteres de dado.

Com relação a pinagem mostrada na Figura 6, os pinos de D0, mais significativo, até D7 são os pinos de dados, que podem operar em 4 bits (D4 até D7) ou 8 bits (D0 até D7). O pino RS informa se o lcd está em modo de dado ou instrução, nível alto e baixo, respectivamente. O RW indica se haverá leitura ou escrita, nível alto e baixo, respectivamente. O pino E ativa ou desativa a transmissão de dados, nível alto e baixo, respectivamente. Os demais pinos são de alimentação e controle da luz de fundo e contraste.

3.3.3 PIR HC-SR501

Figura 7 – Módulo ULtrassônico HC-SR04



Fonte: How PIRs Work

Os sensores PIR são constituídos de uma lente para aumentar a área de atuação e internamente à lente há dois sensores de infravermelho. Basicamente, quando não há movimento, ambos sensores detectam a mesma quantidade de radiação. Porém com a entrada de um corpo externo emissor de infravermelho no campo de atuação do sensor, imediatamente um dos sensores serão alterados devido a maior presença de infravermelho. A partir dessa diferença de potencial entre os sensores, um sinal é emitido, informando que há a presença de um corpo em movimento no local.

3.3.4 Sensor Ultrassônico HC-SR204

Figura 8 – Módulo Ultrassônico HC-SR04

Fonte: Sensor de Distância Ultrassônico HC-SR04

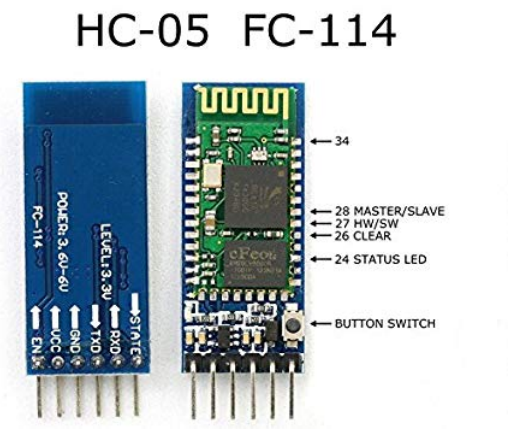
O módulo ultrassônico de distância HC-SR04 tem como finalidade realizar medições da distância de um objeto em relação ao sensor. Apresenta um funcionamento mínimo de 2cm e máximo de 4m. Os pinos presentes correspondem a alimentação do módulo e de emissão e captação do pulso enviado.

A emissão do sinal ocorre com o acionamento do pino *Trigger*, normalmente, utiliza-se um tempo de acionamento de 10 microssegundos. A partir disso, 8 pulso de 40kHz são emitidos e há então o acionamento do pino *Echo*. *Echo*.

Echo

3.3.5 Módulo Bluetooth HC-05

Figura 9 – Módulo Bluetooth HC-05



Fonte: RG Retail HC05 Bluetooth Module for Arduino

O módulo bluetooth utiliza comunicação Serial, utilizando os pinos RX e TX. O pino EN coloca em modo de dados ou de comando AT. TX e RX, transmitem e recebem dados respectivamente. O pino STATE, serve apenas para mostrar o funcionamento do dispositivo, é também conectado ao led de estado presente.

4 Desenvolvimento

Inicialmente houve a busca por informações, basicamente presente no Capítulo 3. Com isso, testes individuais foram feitos, utilizando diversos algoritmos e bibliotecas, até chegara até ao algoritmo que será mostrado a seguir. Alguns dispositivos, não fundamentados, não foram utilizados no projeto, porém testados, a falta desses deu-se por complicações ao longo do projeto.

Com isso, foi atribuída uma função a cada dispositivo. O PIR funciona basicamente para identificar presença, no caso, simulando um porão e acendendo a luz, representada por um led. O motor de passo simula um portão, com abertura feita por meio de um botão ou via comando de um smartphone conectado ao bluetooth. O sensor ultrassônico simula um sensor de proximidade na garagem, onde informa se há a presença de carro ou não e durante o estacionamento de um carro, um *buzzer* deve apitar cada vez mais rápido com a diminuição da distância do carro para o sensor. O bluetooth deve receber status de como está a casa ou abrir o portão, isso de acordo enviado pelo smartphone. O LCD simulará um monitor da casa, em que mostra o estado de um sensor, e ao pressionar um botão muda o sensor monitorado.

A partir disso, foi implementado o algoritmo. A primeira parte do algoritmo é composta pela declaração de bibliotecas, pinos e variáveis, como mostrado nas Figura 10, Figura 11 e Figura 12.

Figura 10 – Algoritmo implementado - Parte 01

```
#include <Wire.h>
#include <SPI.h>
//#include <MFRC522.h>
#include <LiquidCrystal.h>
//#include <Stepper.h>
#include <stdlib.h>
//#include "RTClib.h"
//#include <SoftwareSerial.h>
//SoftwareSerial bluetooth(2,3); //TX, RX (Bluetooth)

/*              MFRC522      Arduino       Arduino   Arduino    Arduino          Arduino
              Reader/PCD   Uno/101       Mega      Nano v3    Leonardo/Micro   Pro Micro
   Signal     Pin          Pin           Pin       Pin        Pin              Pin
   ---------------------------------------------------------------------------------------
   RST/Reset  RST          9             5         D9         RESET/ICSP-5     RST
   SPI SS     SDA(SS)      10            53        D10        10               10
   SPI MOSI   MOSI         11 / ICSP-4   51        D11        ICSP-4           16
   SPI MISO   MISO         12 / ICSP-1   50        D12        ICSP-1           14
   SPI SCK    SCK          13 / ICSP-3   52        D13        ICSP-3           15*/

//rtc
char diaSemana[7][12] = {"Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"};

const int stepsPerRevolution = 32;  // change this to fit the number of steps per revolution
// for your motor
float motorSpeed=50000;
```

Fonte: O autor

Figura 11 – Algoritmo implementado - Parte 02

```
//PINS##########################################################################
//#define SS_PIN 53
//#define RST_PIN 5
//MFRC522 mfrc522(SS_PIN, RST_PIN);
//RTC_DS3231 rtc;

LiquidCrystal lcd(29,28,27,26,25,24);

// motor
int motorPin4=42;
int motorPin3=43;
int motorPin2=44;
int motorPin1=45;

//ultrassom
int trigPin = 40;    // Trigger
int echoPin = 41;    // Echo

//buzzer
int buzzer = 4 ;

//PIR
int pir=30;
int lamp=31;
```

Fonte: O autor

Figura 12 – Algoritmo implementado - Parte 03

```
//VARIAVEIS###########################################################
int k=0;

//ultrassom
float cm=0;
long time_us_d=0;
int timetraveled=0;
long stopb=0;
int stopbuzzer=0;
float lastd=0;
int lockerbuzzer=0;

//motor
long tmotor=millis();
int error=0;

//lcd
int buttonlcd=34;
int bdisplay=1;
int lastButtonState=0;
unsigned long lastDebounceTime=0;
unsigned int debounceDelay=150;
int buttonState=0;
```

Fonte: O autor

Na função *setup()* do arduino foram inicializadas algumas variáveis e a comunicação Serial0 e Serial3, a última utilizada pelo módulo HC-05.

Figura 13 – Algoritmo implementado - Parte 04

```
//SETUP
void setup() {
  Serial.begin(9600);   // Inicia a serial
  Serial3.begin(9600);
  //bluetooth.begin(9600);
  //rtc
  //rtc.begin();

  //rfid
  //SPI.begin();      // Inicia  SPI bus
  //mfrc522.PCD_Init();   // Inicia MFRC522

  //ultrassom/buzzer
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzer, OUTPUT);

  //PIR
  pinMode(lamp, OUTPUT);
  pinMode(pir, INPUT);

  //lcd
  lcd.begin(16, 2); // Define o número de colunas e linhas do display LCD usado.*/
}
```

Fonte: O autor

Foram criadas duas funções de *debounce*, *debounce()* para retirar ruído do botão que seleciona o conteúdo exibido no LCD e *debouncet()* para retirar o ruído do botão que

abre o portão. Em ambas funções, há alterações nos valores de variáveis globais quando o botão é pressionado.

Figura 14 – Algoritmo implementado - Parte 05

```
//botao display LCD
void debounce() {
  //Serial.println(bdisplay);
  int reading = digitalRead(buttonlcd);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == LOW) {
          bdisplay++;
        if(bdisplay==3){
          bdisplay=0;
        }
      }
    }
  }
  lastButtonState = reading;
}

//botao abertura do portao
int por=10;
int lastButtonStatet=0;
unsigned long lastDebounceTimet=0;
unsigned int debounceDelayt=150;
int buttonStatet=0;
void debouncet() {
  int readingt = digitalRead(por);
  if (readingt != lastButtonStatet) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTimet) > debounceDelayt) {
    if (readingt != buttonState) {
      buttonStatet = readingt;
      if (buttonStatet == LOW && k==0) {
          k=1;
          tmotor=millis();
      }
    }
  }
  lastButtonStatet = readingt;
}
```

Fonte: O autor

Foram criadas duas funções de *debounce*, *debounce()* para retirar ruído do botão que seleciona o conteúdo exibido no LCD e *debouncet()* para retirar o ruído do botão que abre o portão. Em ambas funções, há alterações nos valores de variáveis globais quando o botão é pressionado.

Figura 15 – Algoritmo implementado - Parte 05

```
//botao display LCD
void debounce() {
  //Serial.println(bdisplay);
  int reading = digitalRead(buttonlcd);
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == LOW) {
          bdisplay++;
        if(bdisplay==3){
          bdisplay=0;
        }
      }
    }
  }
  lastButtonState = reading;
}

//botao abertura do portao
int por=10;
int lastButtonStatet=0;
unsigned long lastDebounceTimet=0;
unsigned int debounceDelayt=150;
int buttonStatet=0;
void debouncet() {
  int readingt = digitalRead(por);
  if (readingt != lastButtonStatet) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTimet) > debounceDelayt) {
    if (readingt != buttonState) {
      buttonStatet = readingt;
      if (buttonStatet == LOW && k==0) {
          k=1;
          tmotor=millis();
      }
    }
  }
  lastButtonStatet = readingt;
}
```

Fonte: O autor

Já na função *loop()*, inicialmente é armazenado o valor de *millis()* em uma variável e então chamada as funções de *debounce*. Há também a verificação se a comunicação Serial3 está ativa, caso esteja, é lido o valor do comando enviado pelo smartphone, sendo ‘o’ para abertura do portão e ‘k’ para mostrar no celular o estado atual a casa. O software utilizado no smartphone foi o Arduino bluetooth controller, disponível para Android na *Play Store*.

Figura 16 – Algoritmo implementado - Parte 06

```
//int aux=0;
//String id="";
long lcdtime=0; //update lcd each 1 seacond
long m=0; //millis
byte i=0; //bluetooth byte
//long rfid=0;
long serial=0; //serial update each 1 second
//LOOP
int incomingByte;

void loop() {
  m=millis();
  debounce();

  debouncet();
  //look for some command from smartphone bluetooth
  if (Serial3.available() > 0) {
          incomingByte = Serial3.read();
       //open the gate
      if (incomingByte == 'o') {
          k=1; //value for opening gate
          Serial3.println("Comando de abertura");
          tmotor=millis();
      }
      //print house status on smartphone
      else if (incomingByte == 'k') {
          if(cm>15 || cm==0){
            Serial3.println("Garagem Vazia");
          }else{
            Serial3.println("Garagem Com carro");
          }
          if(digitalRead(pir)==1){
            Serial3.println("Ha' gente no porao");
          }else{
            Serial3.println("Ninguem no porao");
          }
          if(k==0){
            Serial3.println("Portao Fechado");
          }
          if(k==1){
            Serial3.println("Portao Abrindo");
          }
          if(k==2){
            Serial3.println("Portao Fechando");
          }
          if(k==3){
            Serial3.println("Portao Aberto");
          }
      }
  }
```

Fonte: O autor

A Figura 17 mostra as fases de um motor de passo. Foi utilizado o método de meio passo, ao inves de passos completos, pode-se perceber isso pela ativação de duas bobinas ao mesmo tempo intercalada por apenas uma. O *delay* faz-se necessário para que o motor possa estabilizar-se por um momento em uma única fase. Na condicional *if* tem-se o movimento em sentido horário enquanto na outra condicional o anti-horário. Na Figura 18 há apenas as mudanças no valor da variável 'k' de acordo com um tempo, sendo essa a que indica o movimento e estado do portão, de 0 á 3, respectivamente, fechado, abrindo, aberto e fechado. Na figura Figura 19 há o dado obtido do sensor PIR junto ao

acionamento ou desligamento do led, simulando uma lampada.

Figura 17 – Algoritmo implementado - Parte 07

```
if(k==1){
   digitalWrite(motorPin1, HIGH); //1
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin4, LOW);
   delayMicroseconds(motorSpeed);
   // 2
   digitalWrite(motorPin1, HIGH);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin4, LOW);
   delayMicroseconds (motorSpeed);
   // 3
   digitalWrite(motorPin1, LOW);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin4, LOW);
   delayMicroseconds(motorSpeed);
   // 4
   digitalWrite(motorPin1, LOW);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin4, LOW);
   delayMicroseconds(motorSpeed);
   // 5
   digitalWrite(motorPin1, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin4, LOW);
   delayMicroseconds(motorSpeed);
   // 6
   digitalWrite(motorPin1, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin4, HIGH);
   delayMicroseconds (motorSpeed);
   // 7
   digitalWrite(motorPin1, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin4, HIGH);
   delayMicroseconds(motorSpeed);
   // 8
   digitalWrite(motorPin1, HIGH);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin4, HIGH);
   delayMicroseconds(motorSpeed);
}
else if(k==2){
   digitalWrite(motorPin4, HIGH);//1
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin1, LOW);
   delayMicroseconds(motorSpeed);
   // 2
   digitalWrite(motorPin4, HIGH);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin1, LOW);
   delayMicroseconds (motorSpeed);
   // 3
   digitalWrite(motorPin4, LOW);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin1, LOW);
   delayMicroseconds(motorSpeed);
   // 4
   digitalWrite(motorPin4, LOW);
   digitalWrite(motorPin3, HIGH);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin1, LOW);
   delayMicroseconds(motorSpeed);
   // 5
   digitalWrite(motorPin4, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin1, LOW);
   delayMicroseconds(motorSpeed);
   // 6
   digitalWrite(motorPin4, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin2, HIGH);
   digitalWrite(motorPin1, HIGH);
   delayMicroseconds (motorSpeed);
   // 7
   digitalWrite(motorPin4, LOW);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin1, HIGH);
   delayMicroseconds(motorSpeed);
   // 8
   digitalWrite(motorPin4, HIGH);
   digitalWrite(motorPin3, LOW);
   digitalWrite(motorPin2, LOW);
   digitalWrite(motorPin1, HIGH);
   delayMicroseconds(motorSpeed);
}
```

Fonte: O autor

Figura 18 – Algoritmo implementado - Parte 08

```
if(k==1 && m-tmotor>10000){
  k=3;
}
else if(k==3 && m-tmotor>15000 && m-tmotor<25010){
  k=2;
}
else if(k==2 && m-tmotor>25000 && m-tmotor<35010){
  k=0;
}
```

Fonte: O autor

Figura 19 – Algoritmo implementado - Parte 09

```
//pir
if(digitalRead(pir)==1){
  //Serial.println("SIMONE IS HERE");
  digitalWrite(lamp, HIGH);
}
else{
  digitalWrite(lamp, LOW);
}
```

Fonte: O autor

Na Figura 20 e Figura 21 há a programação do *buzzer* junto ao ultrassom. Basicamente quanto mais perto o objeto, considere um carro, mais rápido o *buzzer* apita até um valor em que fique constante. Caso o objeto não se mova durante 1s, o *buzzer* é bloqueado e desligado.

Figura 20 – Algoritmo implementado - Parte 10

```
 //ultrassom/buzzer
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
timetraveled = pulseIn(echoPin, HIGH, 20000);
cm = (timetraveled/2)*0.0343;
```

Fonte: O autor

Figura 21 – Algoritmo implementado - Parte 11

```
if(stopbuzzer==0){
  if(cm<6 && cm>5 && m-time_us_d>500){
      time_us_d=m;
      digitalWrite (buzzer, HIGH) ;
      delayMicroseconds(200) ;
      digitalWrite (buzzer, LOW) ;
      delayMicroseconds(200) ;
   }
  else if(cm<5 && cm>4 && m-time_us_d>250){
      time_us_d=m;
      digitalWrite (buzzer, HIGH) ;
      delayMicroseconds(200) ;
      digitalWrite (buzzer, LOW) ;
      delayMicroseconds(200) ;
   }
  else if(cm<4 && cm>3 && m-time_us_d>100){
      time_us_d=m;
      digitalWrite (buzzer, HIGH) ;
      delayMicroseconds(200) ;
      digitalWrite (buzzer, LOW) ;
      delayMicroseconds(200) ;
   }
  if(cm<3 && cm>1){
      digitalWrite (buzzer, HIGH) ;
      delayMicroseconds(200) ;
      digitalWrite (buzzer, LOW) ;
      delayMicroseconds(200) ;
   }

}
if(m-stopb>1000){
  stopb=m;
  lastd=cm;
  lockerbuzzer++;
}
if(cm<lastd+0.4 && cm>lastd-0.4 && lockerbuzzer>=4){
  stopbuzzer=1;
  lockerbuzzer=0;
}
if(cm<lastd-0.4){
  stopbuzzer=0;
}
```

Fonte: O autor

A figura a seguir mostra a implementação dos dados a serem mostrados no LCD de acordo com uma variável, no caso o botão da função *debounce()*. A Figura 22 apenas atualiza o monitor serial do computador a cada 1 segundo.

Figura 22 – Algoritmo implementado - Parte 12

```
//lcd
if(m-lcdtime>1000){
   lcdtime=m;
   lcd.clear();
   if(bdisplay==0){
      lcd.setCursor(0,0);
      lcd.print("Garagem");
      lcd.setCursor(0,1);
      if(cm>15 || cm==0){
        lcd.print("Vazia");
      }else{
        lcd.print("Com carro");
      }
   }
   if(bdisplay==1){
      lcd.setCursor(0,0);
      lcd.print("Porao");
      lcd.setCursor(0,1);
      if(digitalRead(pir)==1){
        lcd.print("Ha' gente");
      }else{
        lcd.print("Ninguem aqui");
      }
   }
   if(bdisplay==2){
      lcd.setCursor(0,0);
      lcd.print("Portao");
      lcd.setCursor(0,1);
      if(k==0){
        lcd.print("Fechado");
      }
      if(k==1){
        lcd.print("Abrindo");
      }
      if(k==2){
        lcd.print("Fechando");
      }
      if(k==3){
        lcd.print("Aberto");
      }
   }
}
```

Fonte: O autor

Figura 23 – Algoritmo implementado - Parte 13

```
if(m-serial>1000){
  Serial.println(cm);
  Serial.print("cm");
  Serial.println(k);
  serial=m;
}//Serial.println(k);
  Serial.println(cm);
  Serial.print("cm");
```

Fonte: O autor

Figura 24 – Pseudocódigo

```
Sempre:
        Se botão_display == presionado:
                valor_display = valor_display+1;
                Se valor_display == 3:
                        valor_display = 0;

        Se botão_portão == pressionado e valor_portao == 0:
                valor_portao=valor_portao=1;
                reinicia_cronometro_do_portao; //cronometro começa a correr
        Se valor_portao == 1 e cronometro_do_portao > 10 segundos:
                valor_portao=3;
        Se valor_portao == 3 e cronometro_do_portao > 15 segundos:
                valor_portao=2;
        Se valor_portao == 2 e cronometro_do_portao > 25 segundos:
                valor_portao=0;

        Recebe dado do smartphone:
                Se dado == 'k':
                        Se valor_display == 0:
                                Se sensor ultrassom DETECTA carro:
                                        Escreve no smartphone => Garagem \n com carro;
                                Senão:
                                        Escreve no smartphone => Garagem \n Vazia;
                        Se valor_display == 1:
                                Se sensor_PIR DETECTA movimento:
                                        Escreve no smartphone => Porão \n Há gente;
                                Senão:
                                        Escreve no smartphone => Porão \n Ninguém aqui;
                        Se valor_display == 2:
                                Se valor_portao == 0:
                                        Escreve no smartphone => Portão \n Portão;
                                Se valor_portao == 1:
                                        Escreve no smartphone => Portão \n Abrindo;
                                Se valor_portao == 2:
                                        Escreve no smartphone => Portão \n Fechando;
                                Se valor_portao == 3:
                                        Escreve no smartphone => Portão \n Aberto;
                Se dado == 'k':
                        Imprima no smartphone => Comando de abertura
                        valor_portao = 1;

        Se valor_portao == 1:
                Motor_gira_sentido_horario;
        Se valor_portao == 2:
                Motor_gira_sentido_anti_horario;

        Se sensor_PIR DETECTA movimento:
                acende_a_luz;
        Senão:
                apaga_a_luz;
```

```
Se sensor_ultrassom DETECTA carro:
        Se a distancia for constante durante 1s:
                buzzer para de apitar;
        Senão:
                Se 5cm < distancia entre sensor e carro < 6cm:
                        buzzer apita a cada 0.5 segundos;
                Se 4cm < distancia entre sensor e carro < 5cm:
                        buzzer apita a cada 0.5 segundos;
                Se 3cm < distancia entre sensor e carro < 4cm:
                        buzzer apita a cada 0.5 segundos;
                Se 1cm < distancia entre sensor e carro < 3cm:
                        buzzer apita constantemente;


A cada 1 segundo:
        Se valor_display == 0:
                Se sensor ultrassom DETECTA carro:
                        Escreve no display => Garagem \n com carro
                Senão:
                        Escreve no display => Garagem \n Vazia
        Se valor_display == 1:
                Se sensor_PIR DETECTA movimento:
                        Escreve no display => Porão \n Há gente
                Senão:
                        Escreve no display => Porão \n Ninguém aqui
        Se valor_display == 2:
                Se valor_portao == 0:
                        Escreve no display => Portão \n Portão
                Se valor_portao == 1:
                        Escreve no display => Portão \n Abrindo
                Se valor_portao == 2:
                        Escreve no display => Portão \n Fechando
                Se valor_portao == 3:
                        Escreve no display => Portão \n Aberto

A cada 1 segundo:
        Imprima a distância entre o sensor e o carro
```

Fonte: O autor

5 Resultados Obtidos e Discussões

Como pode-se perceber, não foi finalizada a função *loop()* no Capítulo 4. Como não houve a implementação de fato, preferiu-se discutir sobre nesse capítulo. A Figura 25 mostra o final da função em questão. Nela há o algoritmo previamente implementado utilizando o RFID, apresenta até mesmo a tag decodificada lida no momento. Porém, a biblioteca utilizada para tratar o RFID apresentava, provavelmente, *delays()* internos, os quais causavam consequentes erros nos outros dispositivos devido ao atraso.

Há ainda a presença de configurações feitas para implementação do módulo RTC. Ao contrário disso, ele não foi utilizado devido ao custo de tempo envolvido para solucionar o funcionamento da maior quantidade possível de dispositivos sem acarretar em atrasos ou falhas destes. No caso o motor demonstrou-se complexo fazer funcionar, já que grande parte do algoritmos encontrados na internet utiliza bibliotecas que também possuem atrasos grandes, tal fato consumiu de grande forma o tempo envolvido com o projeto. Além disso, no primeiro momento o módulo HC-05 demonstrou um grau moderado de necessidade compreensão para implementação.

Figura 25 – Algoritmo implementado - Parte 14

```
/*if (m-rfid>10000){
rfid=m;
  Serial.println("ID:");
  if(getID()==1){
  for (i = 0; i < mfrc522.uid.size; i++)
  {
     //Serial.print(mfrc522.uid.uidByte[i]);
     //Serial.print(" ");
     id.concat(String(mfrc522.uid.uidByte[i],HEX));
     mfrc522.uid.uidByte[i]='\0';
   }
  }
   Serial.print(id);
   //delay(2000);
}
//Serial.print(id);
if (id == "96f0a4ac"){
    //Serial.print("ENTREI");
    tmotor=m;
    k=1;
    id="";
}
else if (id != "0000" || id != ""){
    tmotor=m;
    k=5;
}else{Serial.print("TAG ERRADA");}*/

/*DateTime now = rtc.now(); //CHAMADA DE FUNÇÃO
Serial.print("Data: "); //IMPRIME O TEXTO NO MONITOR SERIAL
Serial.print(now.day(), DEC); //IMPRIME NO MONITOR SERIAL O DIA
Serial.print('/'); //IMPRIME O CARACTERE NO MONITOR SERIAL
Serial.print(now.month(), DEC); //IMPRIME NO MONITOR SERIAL O MÊS
Serial.print('/'); //IMPRIME O CARACTERE NO MONITOR SERIAL
Serial.print(now.year(), DEC); //IMPRIME NO MONITOR SERIAL O ANO
Serial.print(" / Dia: "); //IMPRIME O TEXTO NA SERIAL
Serial.print(diaSemana[now.dayOfTheWeek()]); //IMPRIME NO MONITOR SERIAL O DIA
Serial.print(" / Horas: "); //IMPRIME O TEXTO NA SERIAL
Serial.print(now.hour(), DEC); //IMPRIME NO MONITOR SERIAL A HORA
Serial.print(':'); //IMPRIME O CARACTERE NO MONITOR SERIAL
Serial.print(now.minute(), DEC); //IMPRIME NO MONITOR SERIAL OS MINUTOS
Serial.print(':'); //IMPRIME O CARACTERE NO MONITOR SERIAL
Serial.print(now.second(), DEC); //IMPRIME NO MONITOR SERIAL OS SEGUNDOS
Serial.println(); //QUEBRA DE LINHA NA SERIAL
delay(1000); //INTERVALO DE 1 SEGUNDO*/

}//loop end
```

Ativar

Fonte: O autor

Ademais, os resultados com os dispositivos utilizados foram satisfatórios, não demonstrando grandes problemas, exceto o ultrassom com um ruído inconstante durante os testes, porém notáveis. O resultado pode ser observado no vídeo Projeto 01 Embarcados 2019.01 - Unifesp - RA112185

6 Considerações Finais

Apesar de complicações ao longo do projeto e grande tempo envolvido, foi de grande aprendizado sobre vários componentes não vistos em sala de aula, considerando-se o projeto como bem executado e como uma forma de adquirir conhecimento.