CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

CAMPUS DIVINÓPOLIS

GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

Victor Aguiar do Rosário

Automação do controle e monitoramento de veículos de um estacionamento

Divinópolis

2018

Victor Aguiar do Rosário

Automação do controle e monitoramento de veículos de um estacionamento

Relatório de Trabalho de Conclusão de Curso apresentado ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Áreas de integração: Eletrônica e Computação

Orientador: Prof. Thiago Magela Rodrigues Dias
Co-orientador: Prof. Marlon Henrique Teixeira
Co-orientador: Marcos Alberto Saldanha

Divinópolis.
2018.

(Catalogação - Biblioteca Universitária – Campus Divinópolis – CEFET-MG)

R789a Rosário, Victor Aguiar do.

Automação do controle e monitoramento de veículos de um estacionamento. / Victor Aguiar do Rosário. – Divinópolis, 2018.

73f. ; il.

Orientador: Prof. Dr. Thiago Magela Rodrigues Dias.

Coorientador: Prof. M. Sc. Marlon Henrique Teixeira.

Trabalho de Conclusão de Curso (graduação) – Colegiado de Graduação em Engenharia Mecatrônica do Centro Federal de Educação Tecnológica de Minas Gerais.

1. Mecatrônica. 2. Automação. 3. Estacionamento-Veículos. 4. *Raspberry*. 5. Otimização. I. Dias, Thiago Magela Rodrigues. II. Teixeira, Marlon Henrique. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDU: 62(043)

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG / Campus Divinópolis
Curso de Engenharia Mecatrônica

Monografia intitulada “AUTOMAÇÃO DO CONTROLE E MONITORAMENTO DE VEÍCULOS DE UM ESTACIONAMENTO “, de autoria do graduando Victor Aguiar do Rosário, aprovada pela banca examinadora constituída pelos seguintes professores:

_____

Prof. Dr. Thiago Magela Rodrigues Dias - CEFET-MG / Campus Divinópolis – Orientador

_____

Prof. Dr. Daniel Morais dos Reis - CEFET-MG / Campus Divinópolis

_____

Prof. M.Sc. Fernando Thomé de Azevedo Silva - CEFET-MG / Campus Divinópolis

_____

Prof. M.Sc. Marlon Henrique Teixeira - CEFET-MG / Campus Divinópolis

_____

Prof. Dr. Lúcio Flávio Santos Patrício
Coordenador do Curso de Engenharia Mecatrônica
CEFET-MG / Campus Divinópolis

Divinópolis - Dezembro de 2018

RESUMO

Estacionar o carro nas grandes metrópoles tem se tornado uma tarefa cada vez mais difícil, devido ao aumento do número de carros nas ruas, principalmente nos horários de pico (IBGE, 2017). Observando como é feito o controle e monitoramento de veículos do estacionamento de um shopping em Divinópolis, é possível observar grandes filas na entrada do mesmo. A partir disto, surgiu a ideia deste Trabalho de Conclusão de Curso (TCC) como proposta para automatizar o controle e monitoramento de um estacionamento. Deste modo, o sistema terá sensores de luz para detectar a presença dos veículos nas vagas, amplificadores e multiplexadores para diminuir a quantidade necessária de entradas do *Raspberry*. Este irá tratar os dados lidos pelos sensores, controlar os multiplexadores e indicar qual a vaga mais próxima para o motorista estacionar, na área em que ele escolheu ao entrar no estacionamento, além de indicar em um monitor na entrada uma visão do estacionamento juntamente com a localização da sua vaga, sendo que as vagas serão indicadas pela sua numeração. O motorista ao chegar, deverá somente acionar um botão para iniciar o processo de escolha da vaga e liberar a sua entrada. As áreas envolvidas neste projeto serão a eletrônica e a computação. A primeira área é devido ao circuito que envolve os sensores os amplificadores, multiplexadores e o *Raspberry*. Já a segunda área é por causa da programação no *Raspberry* para indicar a melhor vaga, gerar a interface gráfica no monitor, gerenciar os multiplexadores de modo a receber o sinal dos sensores e controlar a entrada e saída dos veículos. O sistema proposto neste trabalho possibilitaria uma maior rapidez no atendimento e no caso de superlotação, evitaria a necessidade de mais funcionários para verificar as vagas, como ocorre na prática, podendo ser aplicado em qualquer estacionamento. Portanto, espera-se que este projeto possa otimizar e agilizar o gerenciamento de grandes estacionamentos e ter como resultado a redução de filas nos casos de superlotação e diminuir o tempo nelas.

Palavras-chave: Automação, Estacionamento, *Raspberry*, Otimização.

SUMÁRIO

1- INTRODUÇÃO

A dificuldade em estacionar é presente dentro de estacionamentos públicos e privados devido à demora em encontrar vaga disponível. Esse problema tem se tornado questão mundial, inúmeras empresas se atentam para implementar sistemas de controle e automação para realizar este gerenciamento. Um bom estacionamento é a porta da frente, a primeira recepção aos clientes. Isso devido o estacionamento, geralmente, ser o primeiro estágio ao qual o cliente tem acesso, possuindo impacto na opinião do mesmo em relação ao atendimento da empresa. (BANDEIRA *et al*, 2014 *apud* CHAVES, 2010).

Em estacionamentos, a chegada massiva de veículos pode causar atrasos na entrada quando a taxa de chegada é semelhante (ou maior que) à taxa de serviço. Além disso, se o número de veículos que saem da instalação for menor que o número de chegadas, uma longa fila poderá ser observada. A literatura recente mostra que o gerenciamento de informações de estacionamento em tempo real reduz o tempo de busca. Percebemos que, quando a instalação está cheia, os motoristas aguardando na fila não precisam esperar a saída de outros clientes, eles podem entrar quando qualquer detector espacial mudar seu status de "ocupado" para "livre". Esta não é uma estratégia de gerenciamento de informações, mas uma facilidade do sistema que reduz o tempo de espera em até 24,7% e o número de veículos em linha em 21,0% (F. Caicedo, J. Vargas, 2012). Uma das formas mais simples de gerenciamento de estacionamentos é a utilização de funcionários que estacionam os carros para o cliente, mas este método não é seguro, pois o funcionário tem acesso aos bens do cliente. Outra forma é utilizar robôs autônomos para estacionar o carro, como no trabalho de Alves e Capovilla (2015) mas este é um método que teria um custo financeiro elevado para implantação.

Analisando o estacionamento de um shopping localizado em Divinópolis-MG como estudo de caso, foi observado que a equipe de gerenciamento deste estacionamento é composta por três funcionários. A função destes era distribuída da seguinte maneira: dois são responsáveis por coordenar a entrada e saída dos veículos, enquanto outro dentro do shopping responsável por receber o pagamento do cliente e entregar o comprovante. Também é possível observar que, nos casos de superlotação, forma-se filas de veículos para entrar no estacionamento, isto porque os funcionários precisam verificar se há vagas, mas durante este tempo os veículos permanecem na fila. Nesse contexto, pode-se verificar a necessidade de

estudos voltados para a otimização de entradas de veículos dentro de estacionamentos. O presente estudo visa essa otimização por meio de um protótipo, o qual possuirá baixo custo de implementação e também não necessitará de treinamento especializado para sua operação.

1.1- DEFINIÇÃO DO PROBLEMA

Alguns estacionamentos quando estão lotados ou com atraso no atendimento, fazem surgir filas que causam transtornos para os motoristas, pelo fato de que ficam esperando ser atendidos para entrar no estacionamento, consequentemente acaba-se utilizando muitos funcionários para tentar agilizar o atendimento, o que nem sempre é alcançado. Logo, surge a necessidade de se automatizar o processo de monitoramento dos veículos no estacionamento com uma verificação em tempo real, para agilizar a entrada dos veículos e mostrar as vagas livres.

1.2- MOTIVAÇÃO

A grande afinidade com a área da computação e o interesse nas tecnologias que envolvem automação são algumas das motivações teóricas deste projeto.

As filas observadas na entrada do estacionamento de um *shopping* em Divinópolis foram também motivações deste projeto, pelo fato das filas causarem transtornos.

A ideia para solucionar este problema foi a de utilizar sensores para detectar a presença dos carros nas vagas e indicar na entrada do estacionamento quantas estão livres e onde elas estão localizadas.

1.3- OBJETIVO GERAL

Desenvolver um sistema que monitora as vagas de um estacionamento, verifica e indica, por meio de um monitor, a vaga mais próxima no setor escolhido pelo cliente que entrar no estacionamento, durante sua trajetória até a vaga escolhida um sistema de placas o orientará até ela e quando o estacionamento estiver lotado o sistema não permitirá a entrada de mais clientes.

1.4- OBJETIVOS ESPECÍFICOS

- Projetar um sistema de sensoriamento com baixo custo financeiro;
- Projetar um sistema simples de comunicação dos sensores com o computador central;
- Simular o funcionamento dos sensores comunicando com o computador central;
- Realizar o mapeamento de um estacionamento real;
- Simular o funcionamento do algoritmo com um número reduzido de vagas;
- Programar no computador a interface gráfica que indica a vaga mais próxima;
- Programar um sistema de levantamento de dados para análise estatística;
- Montar um protótipo com uma resposta suficientemente rápida.

1.5- ESTADO DA ARTE

Atualmente, de acordo com o levantamento bibliográfico realizado entre os anos de 2013 a 2018, os estudos feitos na área de automação de estacionamento estão focados principalmente em dois segmentos. Sendo que o primeiro tipo de estudos é voltado para os estacionamentos públicos nas cidades, já que o número de veículos cresce todo ano e provoca uma superlotação das vagas públicas, consequentemente, causando muita dificuldade para se encontrar uma vaga disponível (IBGE, 2017). Esses estudos utilizam de aplicativos em dispositivos móveis, sistemas de monitoramento e gerenciamento em tempo real e arquiteturas automatizadas (estacionamento vertical) (K. A. SUNITHA *et al.,* 2010) para possibilitar uma rápida busca de vagas ao motorista e melhor gerenciamento destas para os agentes de trânsito. O segundo é voltado para estacionamentos fechados como indústrias e *shoppings*. Nesses casos é possível realizar projetos mais automatizados e implementar arquiteturas diferenciadas e mais sofisticadas, pelo fato de o número de vagas e o espaço do estacionamento ser mais limitado. Esses estudos utilizam desde simples sistemas de monitoramento das vagas até outros totalmente automatizados, que dispensam a necessidade de um motorista para estacionar, como plataformas que estacionam o carro para o motorista (ALVES, 2015), ou estacionamentos verticais que poupam espaço do estacionamento (MAGRI, 2016).

O grande diferencial dessas pesquisas se deve ao fato de os recentes progressos na indústria de automação (Barone, 2013). Tudo isso possibilitou o monitoramento em tempo real das vagas a grandes distâncias, maior facilidade nos pagamentos para os estacionamentos privados, através do celular via internet, no auxílio ao motorista através da disponibilização das informações das vagas e até mesmo escolhendo a vaga ou estacionando o carro para ele**,** mas a maioria dos trabalhos não detalha uma forma de sensoriamento barato e em grande escala, além de verificar a melhor vaga disponível para o motorista e a localização da mesma, este trabalho irá apresentar uma solução para estas questões.

2- REVISÃO DA LITERATURA

Por meio de estudos voltados para o controle e monitoramento de estacionamentos, foi possível observar um grande crescimento por volta de 2010, sendo que antes disso os estudos se limitavam ao uso de câmeras para tal tarefa, explorando somente os recursos possíveis com o tratamento das imagens destas câmeras. Mais recentemente, estes estudos estão voltados não somente para automação do controle e monitoramento, mas também para processos de auxílio no ato de estacionar.

2.1- TRABALHOS RELACIONADOS

O primeiro trabalho encontrado que se pode relacionar com o controle e monitoramento de estacionamento, foi em um artigo de Alastair McLeod (1995), onde o autor fala da importância dos circuitos fechados de televisão para os sistemas de segurança, e cita como exemplo, o controle da entrada de um estacionamento. Pode-se observar que a única ferramenta utilizada no trabalho é uma câmera, tanto que o autor restringe a discussão sobre as gravações dela.

Pouco depois, os avanços foram relacionados ao tratamento das imagens para reconhecimento das placas de identificação dos veículos, em um artigo de Thanongsak Sirithinaphong e Kosin Chamnongthai (1999). Esse artigo tem como objetivo realizar esse reconhecimento para um sistema de estacionamento automático. É possível observar que a mudança foi somente no tratamento das imagens das câmeras através de técnicas de lógica *fuzzy* e redes neurais.

Alguns anos depois, no artigo de T. Nukano, M. Fukumi e M. Khalid (2004) foram obtidos avanços somente nas técnicas de reconhecimento das placas, porém não são propostas melhorias no controle e monitoramento do estacionamento, fato que é citado pelos autores deste artigo.

Algum tempo depois, foi realizado um estudo de um sistema automático multi-level de estacionar carros, para ambientes com pouco espaço. Este estudo já se relaciona com os estudos mais recentes de automação de estacionamento e é um grande avanço científico se comparado aos estudos anteriormente citados, isto é inclusive citado pelos autores (K. A. SUNITHA *et al.,* 2010).

Posteriormente, um trabalho de Caicedo e Vargas (2012) foi o primeiro a se ater ao tempo na fila que os veículos ficam em estacionamento com grande taxa

de entrada e saída de veículos. Para isso ele dividiu o trabalho em 4 estágios, o primeiro estágio inclui um dispositivo eletromecânico como uma máquina de *tickets* com uma caixa de dinheiro e relógio, o segundo para realizar o controle de acesso dos portões, o terceiro inclui leitores de cartões e ejetor de *tickets* e o quarto um sistema de reconhecimento de placas. Os autores também afirmam que o sistema reduz em 21% o tamanho da fila se comparados aos sistemas anteriores.

Poucos anos depois, a maioria dos trabalhos acadêmicos nesta área utilizam o Arduíno como controlador e sensores ultrassônicos para verificar a presença dos carros, como o trabalho de Bandeira (2014), porém os números de vagas destes trabalhos estão limitados ao número de entradas do Arduíno ou de qualquer outro controlador que utilizam.

No mesmo ano, Cavamura (2014) fez um trabalho de gerência do estacionamento que é dividido em duas partes, um modulo em cada vaga com um microcontrolador e um sensor ultrassônico, a segunda parte é o Raspberry, que é a central que recebe os dados através de uma comunicação sem fio e converte para uma interface de usuário através de um monitor, de forma semelhante a este trabalho.

Dois anos após, Puerari (2016) também fez um trabalho de automação de um estacionamento utilizando Arduíno com sensores visuais nas vagas, porém está focado no aplicativo programado no Arduíno para que possa ter flexibilidade de funcionalidades como controle das cancelas e identificação dos carros nas vagas, e não especifica a parte de hardware utilizada.

Neste mesmo ano, Gonçalves (2016) realizou um trabalho de controle de vagas em estacionamento, além de controlar as vagas para deficientes através de cadastro biométrico. Para isto, utilizou um CLP (Controlador Lógico Programável), um sistema supervisório e sensores indutivo para detectar a presença dos carros nas vagas, mas tudo foi projeto para um modelo de estacionamento de 6 vagas apenas.

Recentemente, o trabalho de Santos do Souza (2018), se baseia em uma simulação computacional para otimizar a rotatividade de veículos em um pátio de estacionamento, isso é feito a partir de métodos estatísticos. É notável que este é mais um trabalho destinado a otimizar o tempo para se estacionar, além de aproveitar melhor o espaço do estacionamento.

Com base nestes trabalhos citados é possível ver que até meados de 2010, os trabalhos relacionados com automação do controle de acesso e monitoramento

de estacionamentos não apresentaram grandes avanços, mas após este momento surgiram diversos estudos relacionados, os quais vem permitindo os últimos avanços neste campo.

2.2- FUNDAMENTAÇÃO TEÓRICA

2.2.1- *Raspberry Pi 3 Model B*

O *Raspberry Pi* é um computador do tamanho de um cartão de crédito desenvolvido no Reino Unido pela Fundação *Raspberry Pi*. Todo o *hardware* é integrado em uma única placa. A CPU (do Inglês – *Central Processinng Unit* – Unidade Central de Processamento) da *Raspberry Pi 3 - Model B*, que é a utilizada no projeto, é baseado em um *system on a chip* (SoC – Sistema em um Chip) *Broadcom* BCM2837, que inclui um processador ARMv8 de 1,2 GHz QUAD Core e 1 GB de memória RAM (*Random Access Memory*) em sua última revisão (NEWARK, 2017).

O *Raspberry Pi* (Figura 1) é compatível com sistemas operacionais baseados em *Linux*. O sistema operacional é normalmente armazenado no cartão SD (*Secure Digital Card*). Qualquer linguagem que possa ser compilada na arquitetura ARMv8 pode ser usada para o desenvolvimento de *software*.

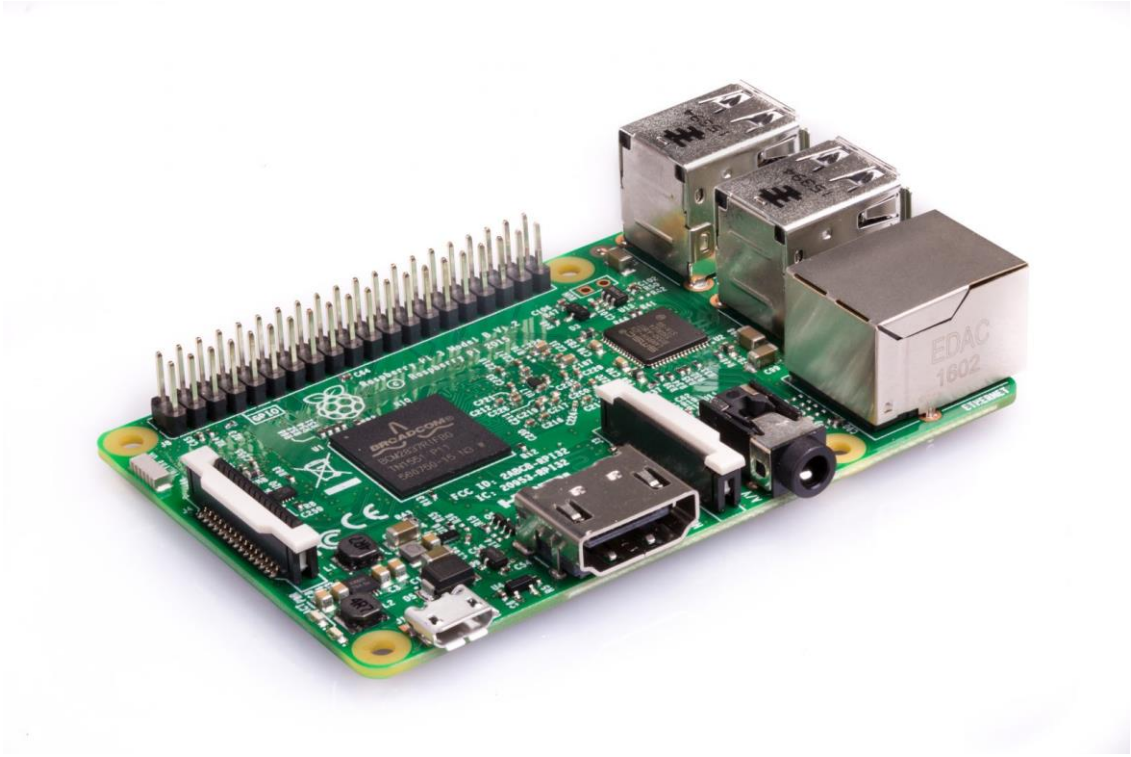


Figura 1 - *Raspberry Pi 3 – Model B* (*Raspberry Pi*, 2017)

Além disso, também conta com 40 pinos de GPIO (*General Purpose Input/Output*) (Figura 2), que podem ser usados para fazer uma interface serial, SPI (*Serial Peripheral Interface*) e I²C (*Inter-Integrated Circuit*), além de poderem ser usado como pinos de *I/O* (*Input/Output*).

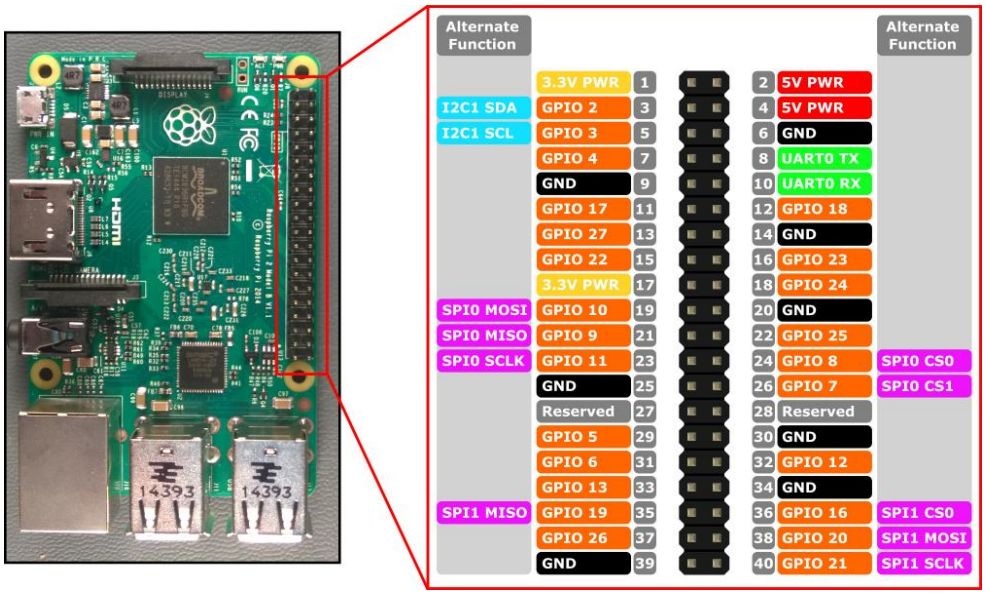


Figura 2 - Pinagem da porta P1 da *Raspberry Pi – Model B* (Developer Microsoft, 2017)

Os pinos de GPIO sobre os pinos de *header* 2x20 incluem SPI, I²C, UART (*Universal Asynchronous Receiver/Transmitter*) serial, 3,3V e 5V. Essas interfaces não são "*plug and play*"(basta conectar e utilizar) e requerem cuidados. Os níveis de tensão aceitáveis nas entradas das GPIOs são de 3,3V, não tolerando tensões superiores.

O *Raspberry Pi* possui grande versatilidade no que tange aos periféricos, por possuir pinos de GPIO, capacidade de processamento maior que a de um Arduíno, além de contar com várias outras interfaces, como Ethernet, DVI (*Digital Visual Interface*), HDMI (*High-Definition Multimedia Interface*), que permitem que o projeto possa ter mais opções de periféricos com os quais se pode fazer a interface (*Raspberry Pi Foundation - FAQs*).

Baseado nestas características, o *Raspberry* será utilizado neste projeto para controlar os multiplexadores e assim receber a informação dos estados dos sensores, além de gerar e controlar a interface gráfica desenvolvida que simulará o funcionamento do estacionamento.

2.2.2- Multiplexador CD4051B

O CD4051B é um *switch* analógico controlado digitalmente com baixa impedância quando “ligado” e baixa corrente de fuga quando “desligado”.

O CD4051B é multiplexador de 8 canais e uma saída, com 3 entradas digitais de controle, A, B, C e uma entrada “*inhibit*”, que quando está ativada nenhuma entrada vai para saída, como pode ser visto na Figura 3. As três entradas digitais selecionam 1 dos 8 canais para serem ligados a saída.

O CD4051B também pode ser usado como demultiplexador, ou seja, os 8 canais de entrada viram saída e vice-versa.

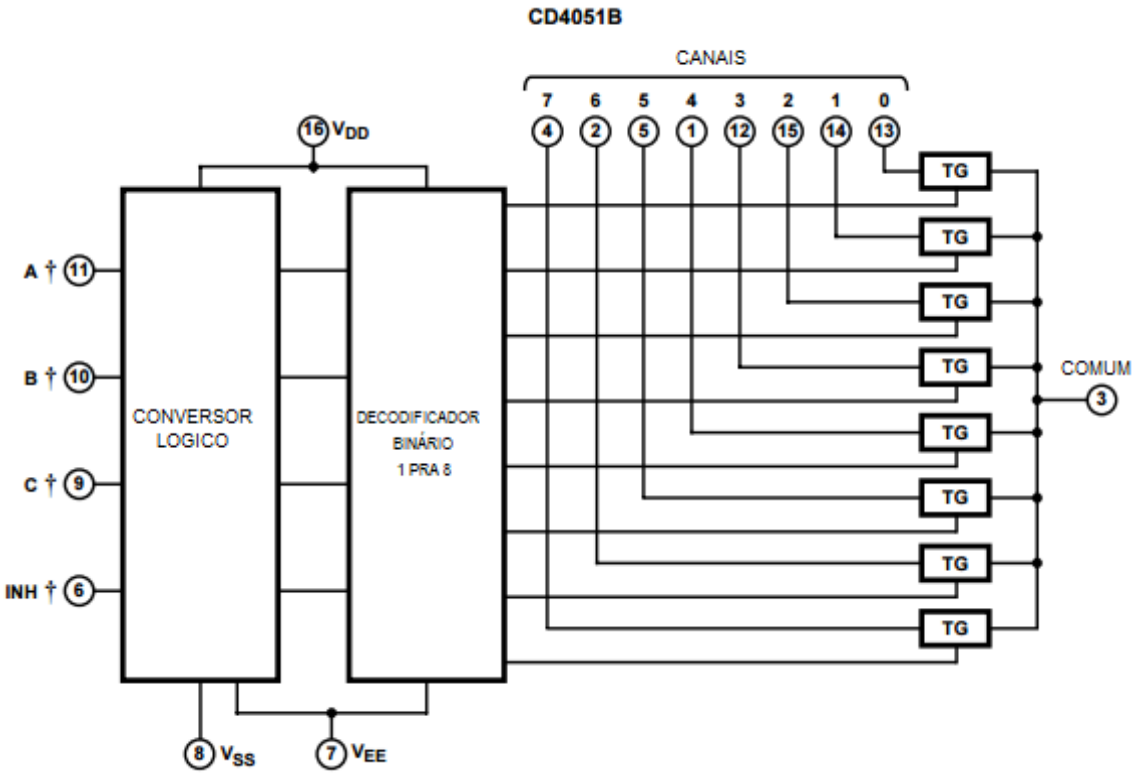


Figura 3 – Entradas e saídas do CD4051B e seu funcionamento (Datasheet, 2017)

Na Figura 3 pode-se observar os 8 canais (0 a 7) que podem ser entrada ou saída, as entradas de controle (A, B e C) e o pino 3 que é a entrada/saída comum, ou seja, a entrada de controle seleciona qual canal vai estar conectado ao pino 3 (TEXAS INSTRUMENTS “CD4051B”, 2017).

Neste trabalho, a função do CD4051B é a de otimizar a quantidade de pinos utilizados pelo *Raspberry*. Considerando que são centenas de vagas e não seria possível conectar todas diretamente no *Raspberry*, com os multiplexadores é possível diminuir consideravelmente o número de pinos, e como será explicado mais a frente, diminui de 320 para 11 pinos necessários.

2.2.3- Amplificador LM358

O LM358 consiste de 2 amplificadores independentes de alto ganho, com frequência interna compensada e foi projetado especificamente para operar com uma única fonte de alimentação com larga faixa de tensão. Este tipo de amplificador possui elevado valor de ganho de tensão, sendo que na prática para baixos níveis de tensão de alimentação, é considerado infinito. Sua função é fornecer na saída uma tensão máxima, igual à da alimentação, se a entrada não-inversora for maior que a inversora, caso contrário a saída de tensão será zero (terra) (TEXAS INSTRUMENTS “LMx58”, 2017). A Figura 4 ilustra os pinos de conexão do CI (Circuito Integrado).

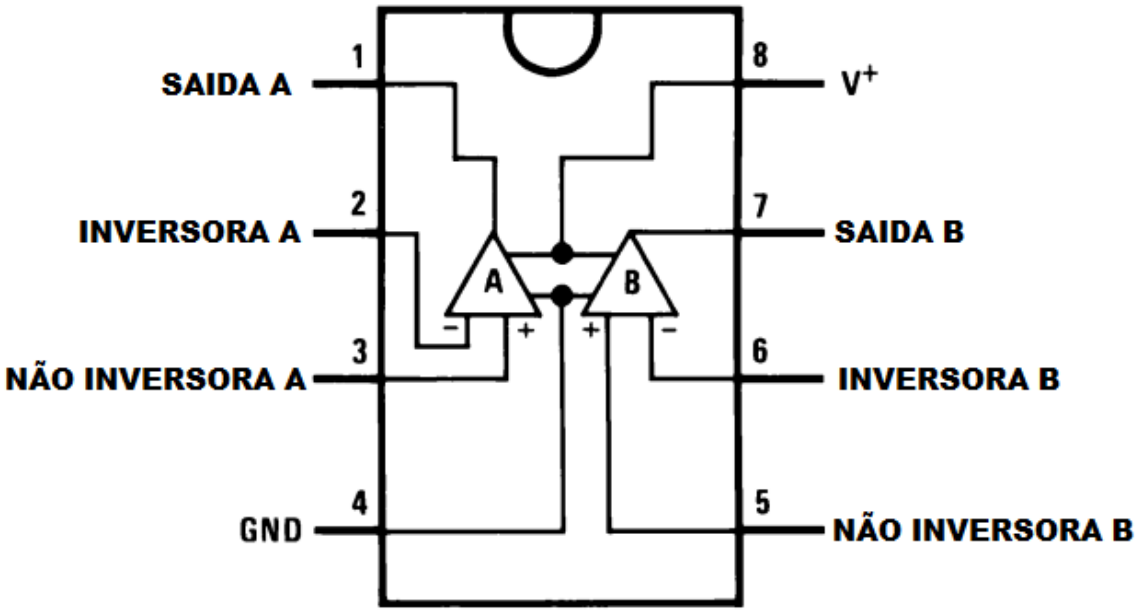


Figura 4 – Pinagem do LM358 (Datasheet, 2017)

Neste trabalho, o LM358 terá a função de comparar o valor de tensão sobre o sensor e a referência, com isto, fornecer na sua saída um sinal lógico alto caso a tensão do sensor seja maior e um sinal lógico baixo se a tensão da referência for maior.

2.2.4- LDR

Os LDRs (do Inglês - *Light Dependent Resistors* – Resistor Dependente de Luz) são componentes sensíveis à luz, ou seja, dispositivos eletrônicos que podem agir sobre um circuito em função da luz incidente numa superfície sensível dos mesmos.

Os LDRs não devem ser confundidos com fotocélulas. Enquanto os LDRs são resistores, cuja resistência depende da intensidade de luz que incide nos mesmos, portanto, não fornece energia alguma como as fotocélulas que convertem energia luminosa (radiante) em energia elétrica (INSTITUTO NEWTON C. BRAGA, 2017).

Figura 5 – Ilustração de um LDR (Project Shop, 2017)

Neste trabalho, o LDR terá a função de intentificar a presença do carro estacionado, isto é feito através da sombra que o carro faz no chão provocando um aumento na resistência do LDR.

2.2.5- Linguagem Python

O Python é uma programação de alto nível altamente extensível e de propósito geral que permite aos programadores expressarem conceitos em menos linhas de código do que seria possível em outras linguagens. A linguagem suporta nativamente paradigmas de programação processuais e orientados a objetos, e recentemente, se tornou a linguagem de *script* escolhida para muitas tarefas administrativas do sistema.

Uma das grandes vantagens do Python é o fato de que vem com um pacote muito completo de bibliotecas padrão, isto permite fazer muitas coisas como baixar arquivos da internet, descompactar arquivos comprimidos ou criar um *web server*, tudo isso com poucas linhas de código.

Python é uma linguagem multiplataforma, em geral, o mesmo programa em python pode ser executado no Windows, Linux, BSD e Mac OS X, simplesmente copiando o arquivo e compilando onde será executado. É póssivel criar programas para uma plataforma específica, mas é raramente necesssário, sendo que as bibliotecas padrão e as de terceiros são na grande maioria multiprataforma.

Python é provavelmente a linguagem de programação mais fácil de aprender e melhor de usar das linguagens mais utilizadas. Seu código é fácil de ler e escrever, o que normalmente torna seu programa menor se comparado aos programas em C++ ou Java, por exemplo (MENEZES, 2016).

2.2.6- Biblioteca de Interface Gráfica Tkinter

O módulo Tkinter ("interface Tk") é a interface padrão do Python para a GUI (*Grafic User Interface*) Tk kit de ferramentas da Scriptics (anteriormente desenvolvido pela Sun Labs).

Ambos Tk e Tkinter estão disponíveis na maioria das plataformas Unix, bem como no Windows e Sistemas Macintosh. Começando com o lançamento 8.0, o Tk oferece aparência nativa em todos plataformas.

O Tkinter consiste em vários módulos. A interface Tk está localizada em um módulo binário chamado *_tkinter* (isso foi *tkinter* em versões anteriores). Este módulo contém o nível baixo para a interface Tk, e nunca deve ser usado diretamente por programadores de aplicativos. Isto é geralmente uma biblioteca compartilhada (ou DLL), mas em alguns casos pode ser estaticamente vinculada ao interpretador do Python (FREDRIK LUNDH, 1999).

Neste trabalho o módulo Tkinter tem a função de gerar a interface gráfica que simulará o funcionamento do estacionamento.

3- METODOLOGIA

Neste cápitulo serão apresentados os procedimentos realizados neste trabalho, bem como todos os materiais, métodos, ferramentas, *softwares* e linguagens de programação.

3.1- MATERIAIS E MÉTODOS

Os procedimentos para a realização deste estudo foram executados nos Laboratórios de Eletrônica e de Protótipos, localizados na instituição de ensino Centro Federal de Educação Tecnológica de Minas Gerais, Campus Divinópolis.

Os softwares utilizados para realizar as simulações e projeções dos circuitos utilizados foi o Proteus 8.0 (versão de estudante) e o Eagle 7.5 (versão de estudante) para fazer o desenho do circuito na placa.

Os equipamentos e ferramentas empregados na implementação do projeto foram:

- Osciloscópio;
- Multímetro;
- Fonte de Alimentação;
- Ferro de Solda;
- *Raspberry Pi 3 Model B*;
- Alicate;
- Monitor com entrada HDMI;
- Mouse e Teclado.

Os materiais utilizados foram: componentes eletrônicos (resistores, circuitos integrados LM358 (Amplificador) e CD4051B (Multiplexador), LED, LDR), *protoboard*, cola quente, estanho e cabos.

O LDR foi utilizado para detectar a presença do carro.

O LM358 foi utilizado para comparar a tensão do LDR da vaga com o LDR de referência.

O multiplexador CD4051B foi utilizado para reduzir o número de pinos necessários no *Raspberry*.

O *Raspberry* é responsável por receber os dados das vagas, controlar o sistema de sensoriamento (sensores e multiplexadores), encontrar a vaga mais próxima, calcular a quantidade de carros e gerar a interface gráfica desenvolvida.

Para a programação no *Raspberry* foi utilizado a linguagem python, padrão do *Raspberry*. Para gerar a interface gráfica, foi utilizada a biblioteca Tkinter que é padrão do Python. A Figura 6 mostra um diagrama dos componentes utilizados e as conexões entre eles.

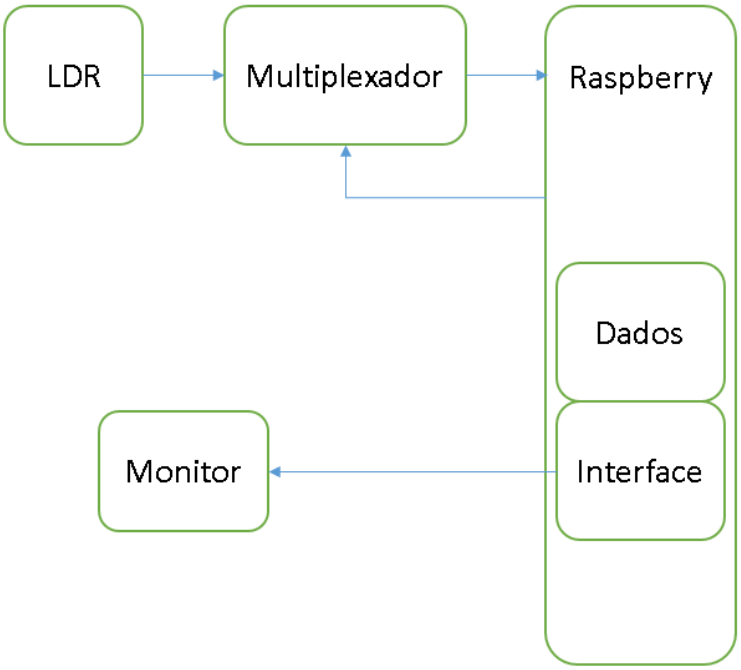


Figura 6 – Diagrama do sistema e suas conexões

3.2- PROCEDIMENTOS REALIZADOS

3.2.1 – Pesquisa e Projeto

Inicialmente foram pesquisados os tipos de sensores que poderiam realizar a detecção de um carro. Os sensores ultrassônico, de luz (LDR), indutivo e infravermelho podem ser utilizados como sensores de presença do carro. O sensor ultrassonico era o mais utilizado e o sensor infravermellho possui sinal digital, mas LDR é o mais barato, simples de implementar e possui distância de medição suficiente. Este trabalho prentende abordar um sistema que possa ser aplicado a um grande número de vagas e apresentar como seria a comunicação destas muitas vagas com o dispositivo de controle, que no caso é um *Raspberry*. Para isso propõe-se o uso de multiplexadores. Desta forma, o CD4051B foi o escolhido, pelo fato de ser mais facilmente encontrado do que os que possuem mais canais de multiplexação.

3.2.2 – Resistência do LDR

Após definido o sensor, foi realizado a medição da resistência do mesmo nas diferentes situações de atuação. Inicialmente nos extremos das condições ideais do projeto, muita luz do sol e a noite embaixo do carro sob luz artificial, nisto obte-se 60 Ω e 110 kΩ, respectivamente. Em seguida, foram feitas medições de resistência do LDR no estacionamento no andar superior que não possui telhado, portanto, recebe luz direta do sol, e no andar inferior que é todo coberto e possui iluminação artificial o dia todo. Nos dois locais foram realizadas 20 medições de forma aleatória pelo espaço do estacionamento, sendo que essas 20 foram realizadas sem a presença do carro. Esse mesmo número de medições foram realizadas a noite e de dia. Não foi possível realizar 20 medições nas mesmas condições anteriores com a presença do carro sobre o sensor, pelo fato de que as vezes não havia carros suficientes estacionados, portanto foram feitas somente 5 medições de cada, e no caso do andar superior de dia, não havia nenhum carro e as medições foram feitas com objetos que simulam a sombra de um veículo. A Tabela 1 mostra as medições obtidas sem a presença do carro.

Tabela 1 – Medições de resistência (Ω) do LDR sem presença de carro

DIA				NOITE			
Inferior		Superior		Inferior		Superior	
13,35k	6,67k	194	163	11,70k	9,30k	28k	93k
10,00k	5,60k	188	184	18,20k	9,30k	65k	91k
10,73k	10,40k	184	180	30,00k	10,10k	82k	54k
8,30k	14,40k	180	182	11,70k	12,60k	28k	62k
5,40k	6,40k	160	185	13,50k	14,00k	6k	37k
6,60k	10,00k	230	185	13,50k	15,80k	6k	90k
24,00k	3,70k	173	180	6,70k	28,80k	35k	96k
4,60k	5,60k	174	167	7,90k	6,40k	70k	120k
8,30k	5,40k	186	155	14,90k	7,20k	80k	174k
18,00k	14,70k	170	200	11,50k	10,00k	58k	220k

Na primeira parte da tabela estão as medições feitas durante o dia no andar superior e no inferior. Posteriormente estão as medições feitas a noite.

Lembrando que as medições foram feitas utilizando o mesmo LDR e um multímetro, além disto, todas foram feitas direcionando a face do LDR para cima, mesmo que a luz estivesse em outra direção, para manter o padrão de medição.

A Tabela 2 mostra as medições de resistência do LDR com a presença do carro sobre ele.

Tabela 2 - Medições de resistência (Ω) do LDR com presença de carro

DIA		NOITE	
Inferior	**Superior**	**Inferior**	**Superior**
45k	3,0k	61k	1200k
60k	3,5k	114k	860k
83k	2,6k	200k	960k
196k	1,9k	83k	1300k
160k	1,2k	152k	700k

Nas medições sobre o carro teve-se o cuidado de posicionar o LDR o mais próximo do meio da parte de baixo do carro, onde ele estaria na prática e teoricamente a sombra está com mais intensidade.

Lembrando que o andar superior é mal iluminado, o indicado para a aplicação deste projeto é uma iluminação como a do andar inferior, pois a noite todas as vagas são iluminadas.

A partir das 20 medições realizadas sem a presença do carro, foi calculado a média, a máxima e a mínima resistência em cada horário do dia e em cada andar. A Tabela 3 mostra as medidas obtidas.

Taleba 3 – Média, Mínima e Máxima resitência do LDR sem o carro

	DIA		NOITE	
	Inferior	**Superior**	**Inferior**	**Superior**
Mínima	3,70k	155	6,40k	6k
Média	9,60k	181	13,15k	75k
Máxima	24k	230	30,00k	220k

Com isto pode se observar que a média das duas situações do andar inferior são proximas de 10 kΩ. Deste modo, foi escolhido um resistor de 10 kΩ pra ficar em série com o LDR.

Sobre as medições com a presença do carro, também foi avaliado os valores, mínimo, médio e máximo da resistência do LDR. A Tabela 4 apresenta os valores obtidos.

Tabela 4 - Média, Mínima e Máxima resitência do LDR com carro

	DIA		NOITE	
	Inferior	**Superior**	**Inferior**	**Superior**
Mínima	45k	1,2k	61k	700k
Média	109k	2,4k	122k	1000k
Máxima	196k	3,5k	200k	1300k

Pode-se observar que nenhum valor obtido sem o carro, no seu respectivo andar, é maior que os valores obtidos com o carro. Desta forma, pode-se garantir que mesmo a variação de luminozidade no decorrer do dia não altera a capacidade do sensor de indicar a presença do carro. Pode-se observar também, que usando um resistor de 10kΩ em série com o LDR e uma alimentação de 5V, as medições do andar superior a noite são muitas vezes maior que 10kΩ, mesmo considerando o valor médio, ainda assim, a diferênça entre a resistência com carro e sem o carro é cerca 10 vezes, assim, na tensão do LDR, essa diferênça seria em torno de 10% da tensão de alimentação. Portanto, mesmo com pouca iluminação o LDR garante indicação da presença do carro.

Considerando que o sensor é passível de medições incorretas, como o mal posicionamento do carro na vaga, de forma que não faça sombra no carro, a presença de outro objeto ou animal que inflija sombra no sensor, ainda assim o sistema corrige indicações que não foram selecionados pelo sistema como já foi explicado na parte de programação. Ainda assim, qualquer sensor estaria passível destas falsas indicações, alguns menos que outros, mas não compensa utilizar um sensor com maior preço para aumentar insignificamente a confiabilidade da deteção. A Tabela 5 mostra os valores médios de tensão que estariam no LDR utilizando um LDR de 10kΩ, os valores estão em porcentagem da alimentação.

Tabela 5 – Tensão prevista no LDR com e sem o carro (%)

	DIA		NOITE	
	Inferior	**Superior**	**Inferior**	**Superior**
S/ Carro	48,97%	1,77%	56,80%	88,23%
C/ Carro	91,59%	19,35%	92,42%	99,00%
Diferença	42,62%	17,58%	35,62%	10,77%

Os valores utlizados foram as medições médias obtidas e calculado a porcentagem de tensão no LDR, além da diferença entre os valores com e sem o carro. Pode-se observar que a diferença de tensão é suficiente para evitar falsas medições devido leves variações na iluminação, pois na situação de iluminação desejada pelo projeto, a menor diferença de tensão foi de 35,62% (cerca de 1,7V com alimentação de 5V). A menor diferença de tensão foi de 10,77% (cerca de 500mV com alimentação de 5V). É possível que ruídos possam afetar a detecção, mas pouco provável sendo que este teria que ser de cerca de 500mV constantes na pior das situações, ainda assim, caso ocorra, pode-se analisar o ruído e projetar um filtro para ser adicionado na entrada dos multiplexadores.

A fórmula para calular a tensão no LDR é a do divisor de tensão, um exemplo pode ser visto na Equação 3.1.

$$V_{LDR} = \frac{V_{CC} \times a}{a+b} \tag{3.1}$$

Sendo que “a” é a resistência do LDR, “b” é o resistor em série, e V_{cc} é a tensão de alimentação. Para obter o valor em porcentagem é só remover da equação a tensão de alimentação e multiplicar por 100, como pode ser visto na Equação 3.2.

$$V\%_{LDR} = \frac{100a}{a+b} \tag{3.2}$$

Os valores presentes na Tabela 5 foram calculados utilizando a Equação 3.2 e os valores médios das Tabelas 3 e 4.

3.2.3 – Circuito Sensor

Baseado no teste do LDR foi projetado um circuito com 2 LDRs por vaga, sendo 1 para medição na vaga (em baixo do carro) e outro para referência (luz ambiente), de modo que a diferença de medição dos dois gerasse um sinal lógico alto ou baixo na saída (Compensação). É possível utilizar só um LDR de referência para todo o sistema, ao invés de uma para cada vaga. A quantidade de LDRs utilizados como referência será definido pela homogeneidade da luz no estacionamento.

Para isso, foi projetado inicialmente um amplificador diferencial com ganho 1 para calcular a diferença de tensão do sensor e da referência e a partir desse valor, comparar com a tensão de um potenciômetro de ajuste, de modo que tenha sinal alto na saída, caso a diferença medida seja maior do que o valor medido no potenciômetro.

Porém observou-se que era possível utilizar somente o comparador simples entre o sensor e a referência, aumentando o valor da resistência que está em série com o sensor, evitando assim falsas medições causadas por pequenas diferenças entre sensor e referência, como pode ser visto na Figura 7. A saída será digital, pois o comparador simples possui ganho muito grande, ou seja, no momento que a tensão do sensor for maior que a referência, a saída terá nível lógico alto.

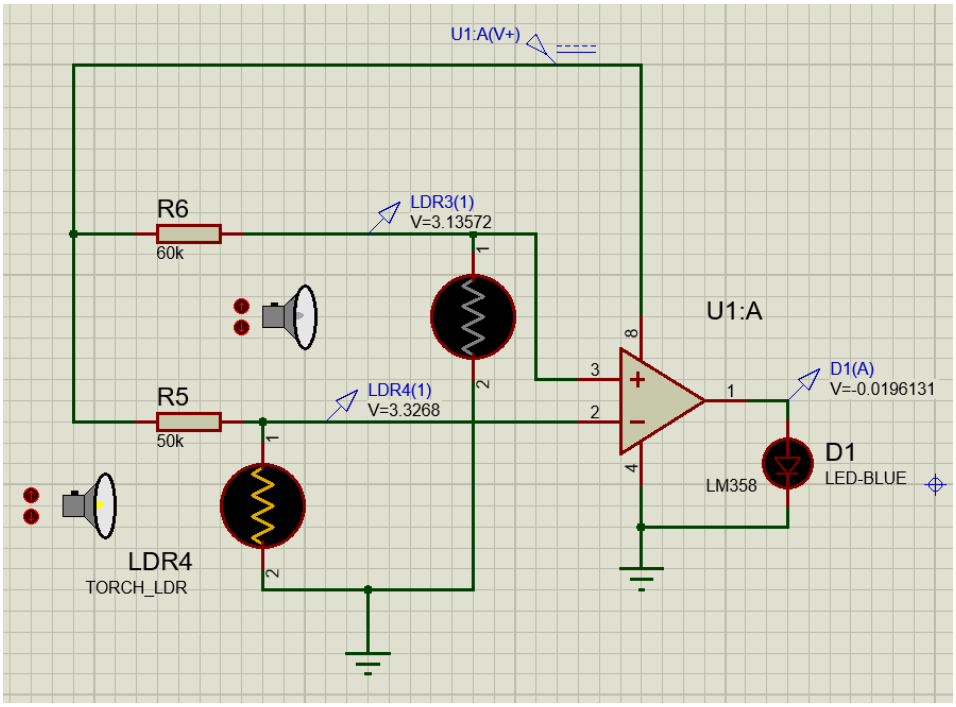


Figura 7 – Simulação do circuito sensor

Os valores de resistência utilizados na simulação foram baseados no valor máximo medido pelo LDR, que no caso foi 110kΩ. Deste modo optou-se por uma resistência com metade do valor, pois se o LDR chegar ao valor máximo de resistência a tensão medida sobre ele será 75% da alimentação, caso chegue ao valor minimo de resistência (60Ω) a tensão sobre ele seria de aproximadamente 0,1%.

Utilizando este mesmo circuito, foi realizado um teste para avaliar a resposta da saída em relação a tensão no LDR. Primeiramente mediu-se o valor de tensão no LDR de referência para verificar a sua estabilidade, obteve-se um valor estável

de 900mV, sendo que o circuito era alimentado com 3,3V do *Raspberry*. Com isso, mediu-se no osciloscópio a tensão no LDR sensor e na saída.

Em função do teste citado acima, foi realizado outro para verificar o nível de tensão que o *Raspberry* entende como nível lógico alto. Para isso, utilizou-se um potenciômetro para ajustar a tensão inserida no pino do *Raspberry*, em outro pino ligou-se um LED, desta forma, quando o pino conectado ao potenciômetro ler um sinal lógico alto, o pino conectado ao LED será ligado. Para verificar a tensão no potenciômetro foi utilizado um multímetro e avaliado qual a maior tensão em que o LED estaria desligado e a menor tensão em que o LED estaria ligado. Para finalizar, mediu-se com o multímetro as saídas dos amplificadores, verificou-se qual o menor valor medido e comparou-se com os valores obtidos com o potenciômetro, assim foi possível avaliar a confiabilidade do nivel de tensão da saída dos amplificadores.

3.2.4 - Multiplexadores

Baseado no número de vagas (320) foi projetado um sistema de multiplexadores e botões no Proteus para atender este números de vagas, no final da análise chegou-se em um sistema com 45 multiplexadores, um microcontrolador e o *Raspberry*, o diagrama de blocos com o funcionamento deste sistema poder ser visto na Figura 8.

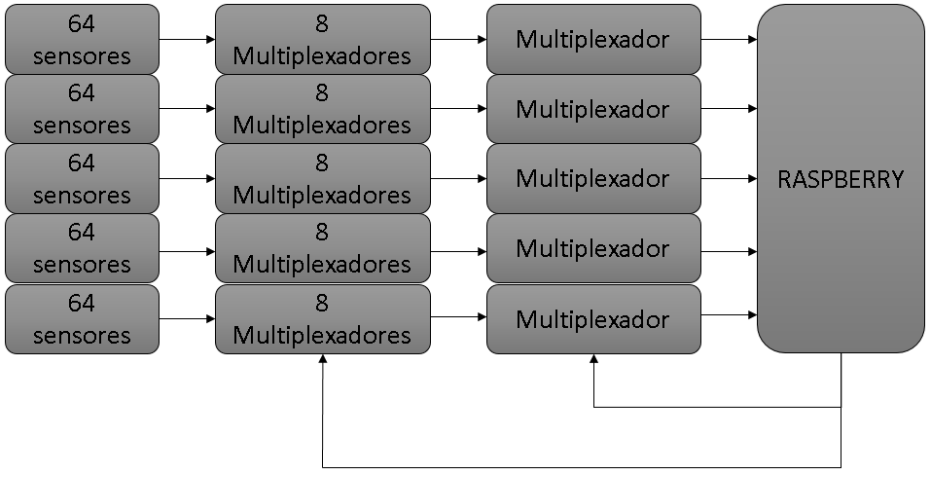


Figura 8 – Diagrama de blocos dos multiplexadores

Em seguida, um sistema mais simples foi montado no Proteus para verificar o funcionamento dos multiplexadores, para isso foram utilizados apenas 3 multiplexadores, 2 na primeira cascata e um na segunda cascata, deste modo foram simulados 16 sensores em funcionamento, que no *software* foram

substituídos por botões. Podemos observar na Figura 9 o diagrama de funcionamento deste sistema simulado.

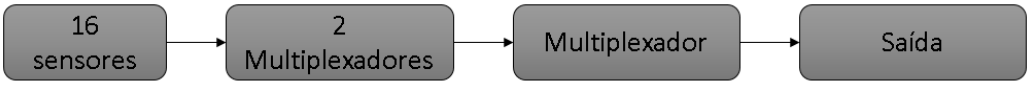


Figura 9 – Diagrama de blocos do sistema simples dos multiplexadores

Com este teste foi possível validar o sistema de multiplexadores que será utilizado, sendo necessário somente expandir para o sistema com os 320 sensores para aplicar no estacionamento real.

3.2.5 - Mapeamento

Em seguida foi feito o mapeamento das vagas do estacionamento de um *shopping* em Divinópolis para usar como referência. O mapeamento realizado obteve um total de 318 vagas, sendo que 115 estão no andar coberto e 203 no andar superior que é aberto, como pode ser visto nas Figuras 10 e 11.

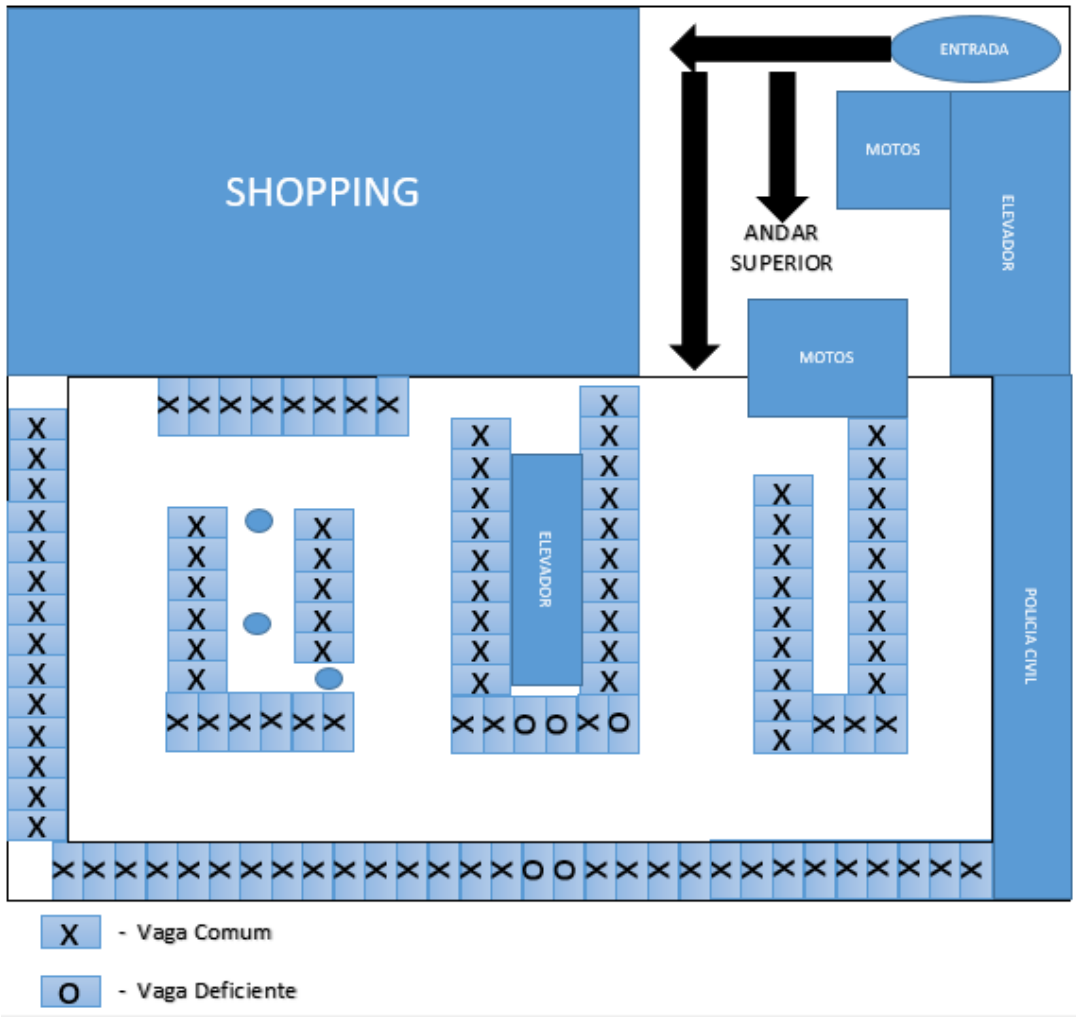


Figura 10 – Mapa das vagas do andar coberto

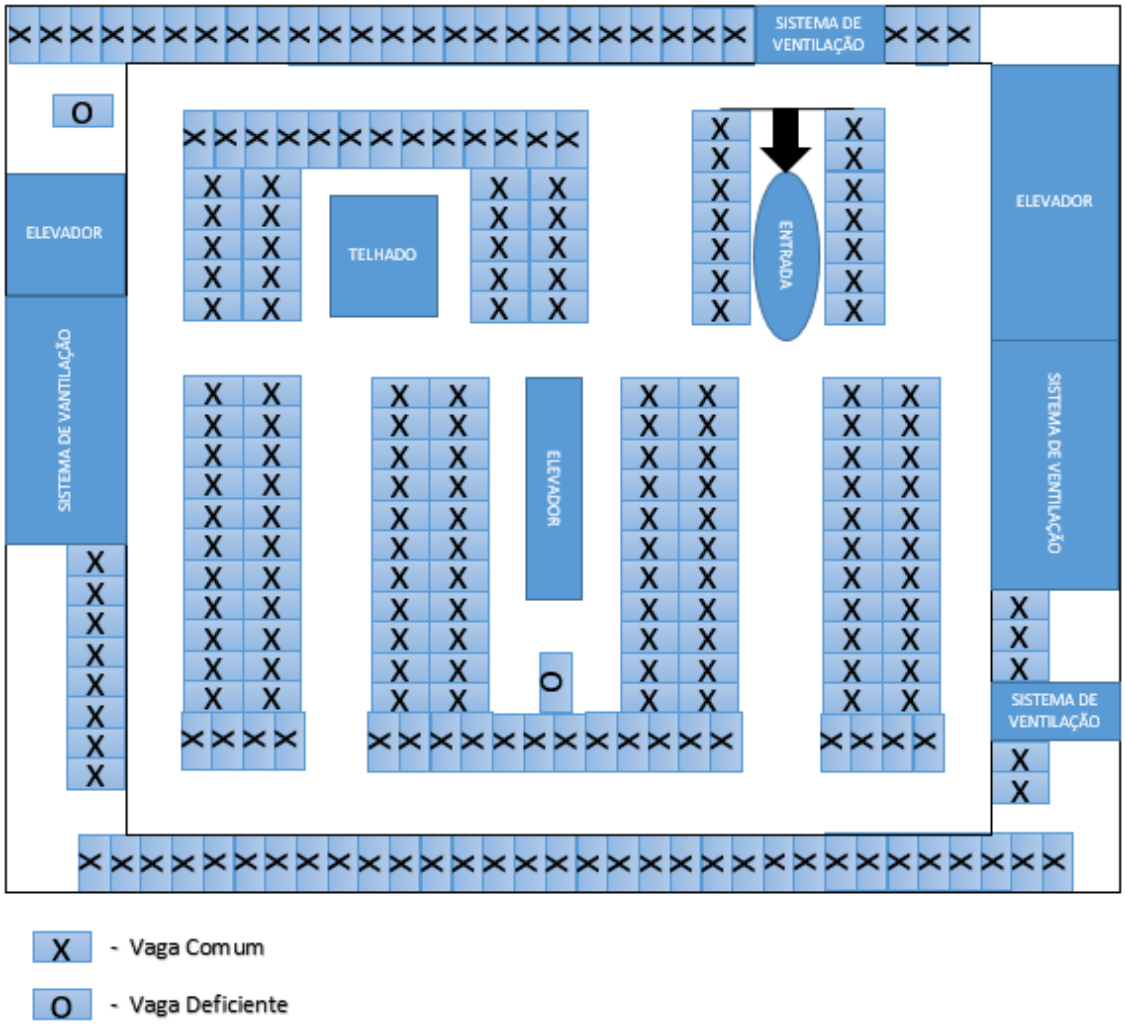


Figura 11 – Mapa das vagas do andar superior

A parte em branco da Figura 10 e 11 representa a área transitável, as região escrita “motos” é a área designada para estacionar as motos, a região escrita “Policia Civil” representa as vagas destinadas à Policia Civil. A seta de cor preta na Figura 10 representa a rampa que dá acesso ao andar superior, assim como tambem está representada da Figura 11.

Todas as regiões transitáveis do andar superior e do inferior são de mão dupla, mas o andar superior tem a vantagem de não possuir pilastras, portanto é possível transitar livremente sobre as vagas. Pode-se observar que as vagas para deficientes estão próximas aos elevadores, como recomendado.

3.2.6 – Implementação da Interface Gráfica

No *Raspberry* foi realizado inicialmente a simulação do funcionamento do estacionamento. Para isso, foi criada no *Raspberry* uma interface gráfica com 16 botões que representavam as vagas, 1 botão que simulava a entrada de um

motorista e 1 botão que simulava a saída. Também foi adicionado um elemento para mostrar a quantidade de veículos dentro do estacionamento.

A simulação consistia de dois arquivos de execução (“janela” e “cria_vagas”), um para gerar dois arquivos com os dados das vagas (as distâncias de cada uma delas até a entrada do estacionamento e qual bloco do estacionamento ela pertence) e outro para rodar a simulação, o código presente nos dois arquivos podem ser vistos nos APÊNDICES A e B, respectivamente. No executável chamado “janela” era lido os arquivos dos dados das vagas, criado uma matriz de representação delas, instanciado os objetos utilizados na interface gráfica, alterado a cor dos botões das vagas para verde (livre), vermelho (preenchida) e amarelo (escolhida), calculado a quantidade de carros que estão dentro do estacionamento e escolhido qual vaga livre no bloco desejado é a mais próxima da entrada. A Figura 12 mostra o fluxograma do funcionamento geral do algoritmo presente no *Raspberry* .

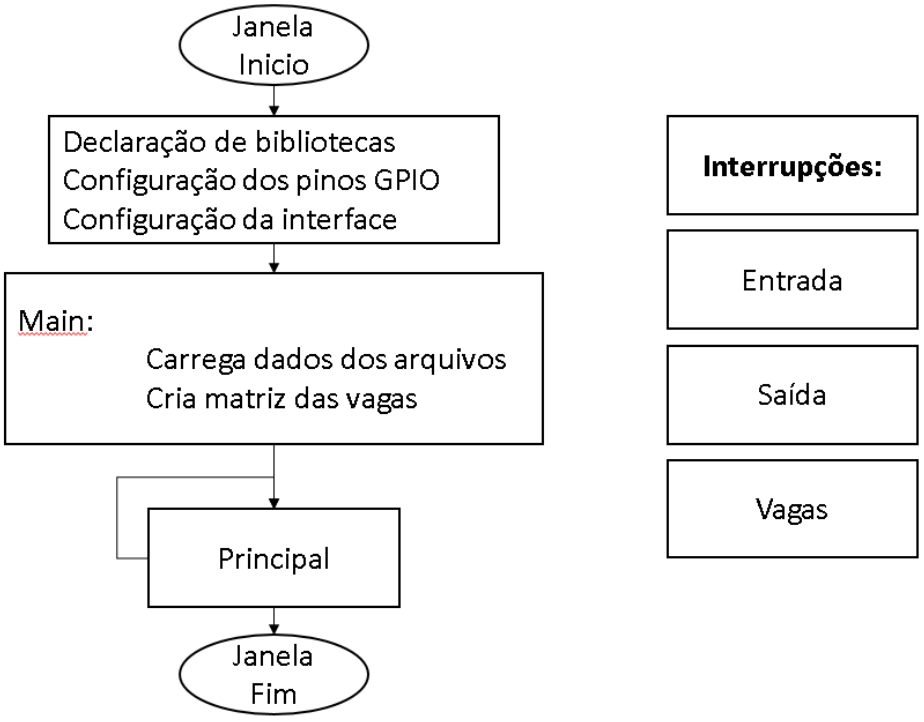


Figura 12 – Fluxograma do algoritmo no *Raspberry*

Como dito anteriormente, a programação no *Raspberry* consiste da execução de dois arquivos, um para entrar com os dados das distâncias da entrada até as vagas e qual é o bloco em que ela esta localizada no estacionamento, e o outro, que é o arquivo principal chamado “Janela”. A Figura 12 mostra o fluxograma do código implementado neste arquivo, mas está representado de forma geral,

sendo que o *loop* principal e as interrupções serão detalhadas a seguir. As interrupções são iniciadas durante a execução do *loop* principal, quando finalizadas, o *loop* principal volta a ser executado. O *loop* principal executa somente três funções, a função “Verifica”, que faz a leitura dos sensores e guarda seus estados nas variáveis, a função “Corrige”, que verifica se o cliente estacionou em uma vaga que não foi selecionada, remove a seleção da vaga mais próxima que esteja somente selecionada e seleciona a que o cliente estacionou, e por fim, a função de atualizar as tarefas da interface gráfica. A Figura 13 mostra o fluxograma do código imlplementado para a função “Atualiza”.

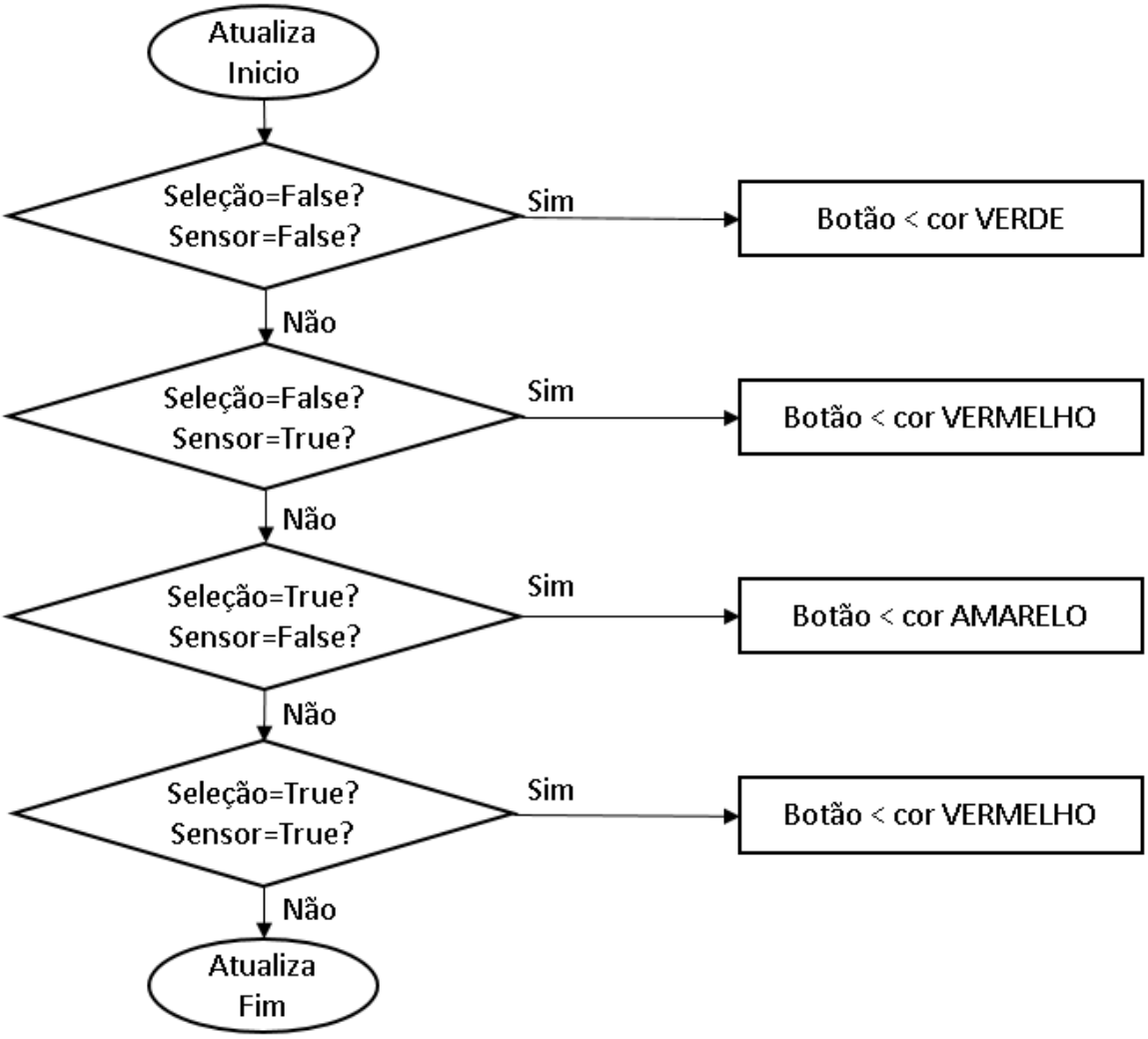


Figura 13 – Fluxograma da função “Atualiza”

Esta função recebe o número da vaga como parâmetro e a partir do estado das variáveis do sensor e da seleção ele altera a cor do botão daquela vaga. Se os dois estados forem falso a cor será verde (vaga disponível), se somente o sensor for verdadeiro a cor será vermelha (vaga ocupada), se somente a seleção for verdadeira a cor será amarela (vaga selecionada) e se os dois forem verdadeiros a cor será vermelha. A Figura 14 mostra o fluxograma do código implementado para a função “Totaliza” e o Quadro 3 mostra seu código.

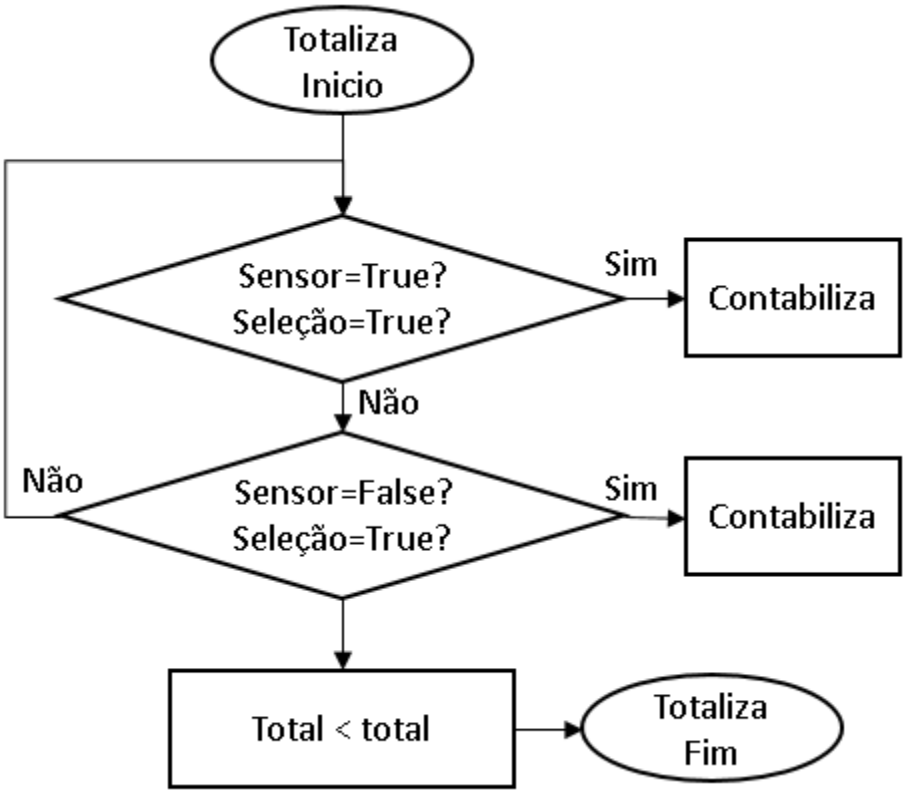


Figura 14 – Fluxograma da função “Totaliza”

Esta função realiza a contagem do número de carros dentro do estacionamento, para isto, ela verifica quantas vagas foram selecionadas, pois pra cada carro que entra uma vaga é selecionada e quando sai uma é deselecionada. A Figura 15 mostra o fluxograma do código implementado para a função “Corrige”.

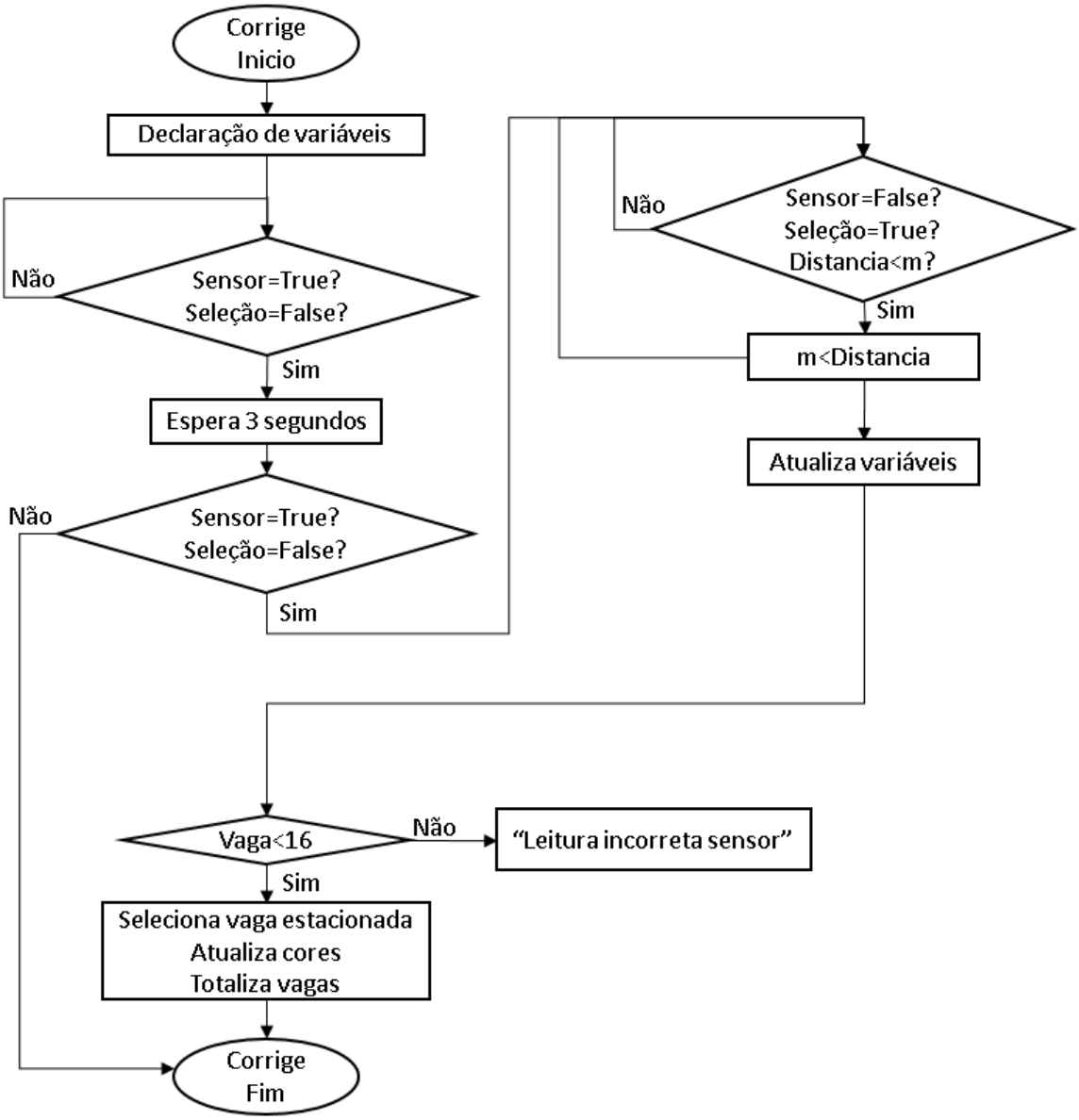


Figura 15 – Fluxograma da função "Corrige"

Como já mencionado anteriormente esta função faz uma correção no caso do cliente estacionar em uma vaga que não foi selecionada para ele, para isto, o código realiza uma verredura verificando se existe vagas com o sensor ativado mas que não esteja selecionada, depois disto, foi inserido um delay de 3 segundos para o caso de ter sido uma ativação momentânea, se a situação se mantiver o código procura a vaga mais próxima que esteja somente selecionada, com isto esta vaga é deselecionada e a vaga que foi estacionada é marcada, por fim, as cores são atualizadas e as vagas totalizadas. A Figura 16 mostra o fluxograma do código implementado para a interrupção ativada pelo botão da entrada.

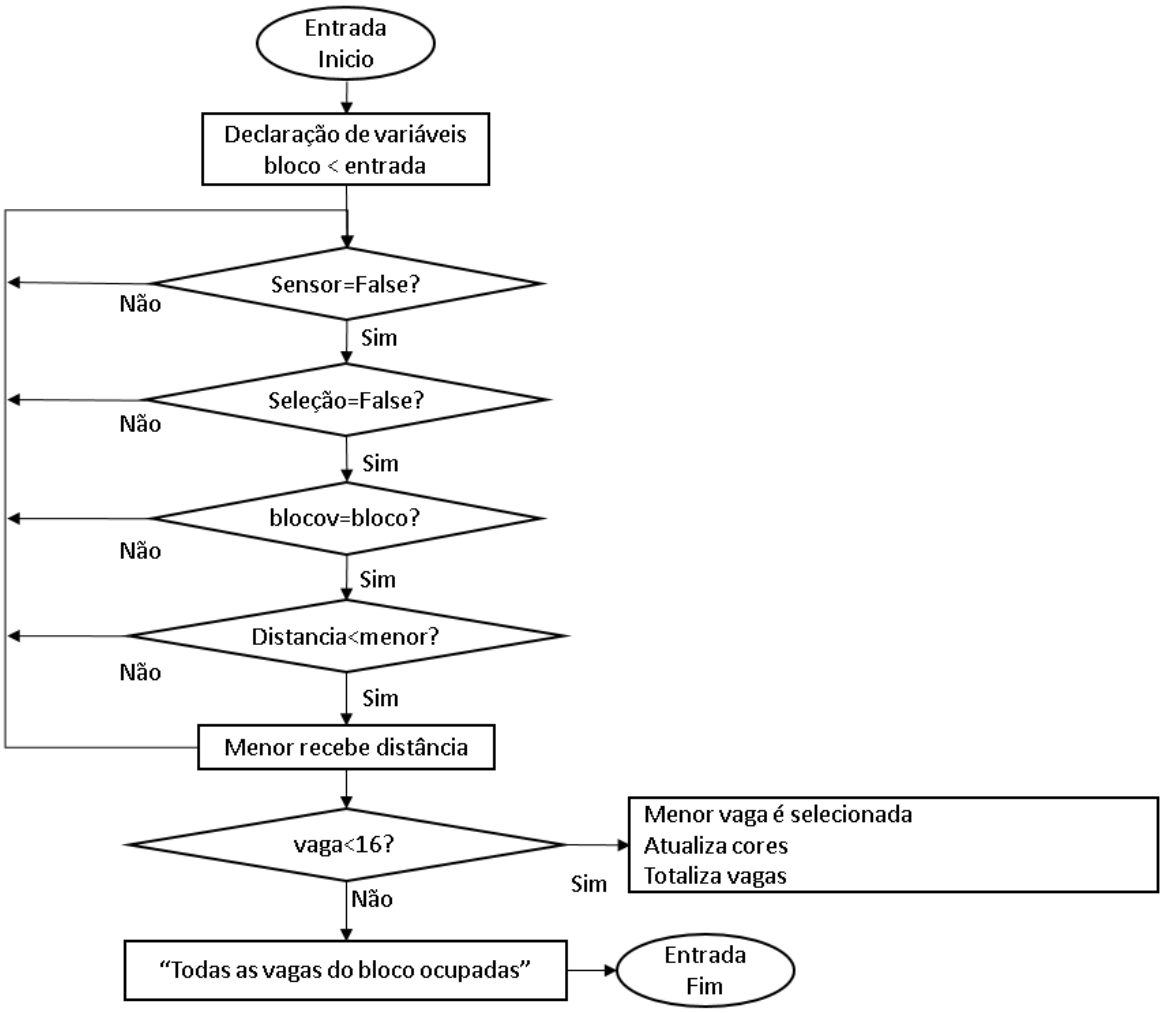


Figura 16 – Fluxograma da interrupção "Entrada"

Acionando-se o botão da entrada, o código simplesmente faz uma varredura verificando as vagas que não estão selecionadas, não estão com o sensor ativado e são do mesmo bloco que foi escolhido na entrada, com isso encontra-se a vaga disponível mais próxima no bloco escolhido. No fim do código, se nenhuma vaga for encontrada, é porque todas as vagas naquele bloco, foram escolhidas e um mensagem é exibida, em seguida o cliente deverá escolher outro bloco. A Figura 17 mostra o fluxograma do código implementado para a interrupção acionada pelo botão da saída.

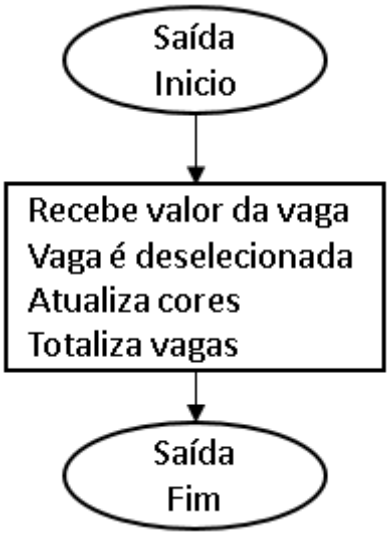


Figura 17 – Fluxograma da interrupção “Saída”

Esta interrupção tem o objetivo de verificar qual vaga que o carro estava estacionado e deseleciona-la. Por fim, é executado as funções para atualizar as cores e contabilizar a quantidade de carros no estacionamento. A Figura 18 mostra o fluxograma do código implementado para a função “Verifica”.

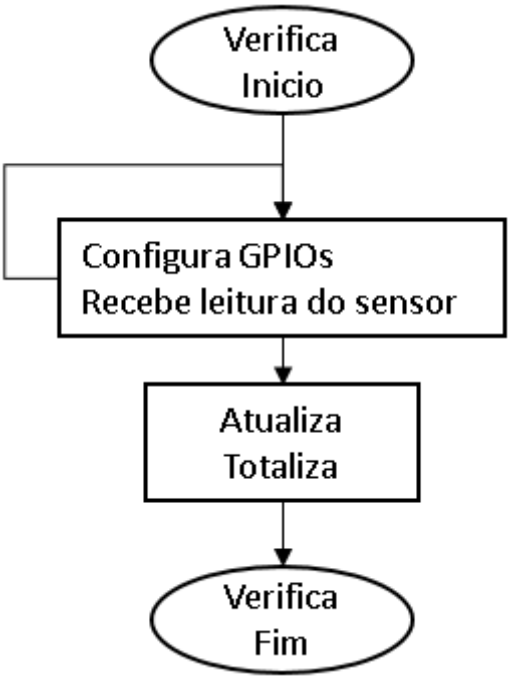


Figura 18 – Fluxograma da função “Verifica”

Esta função faz a configuração dos pinos das GPIOs e salva na variável o estado lido naquele sensor, no fim da varredura, as funções “Atualiza” e “Totaliza” são chamadas. A Figura 19 mostra o fluxograma do código implementado para a interrupção acionada pelos botões das vagas.

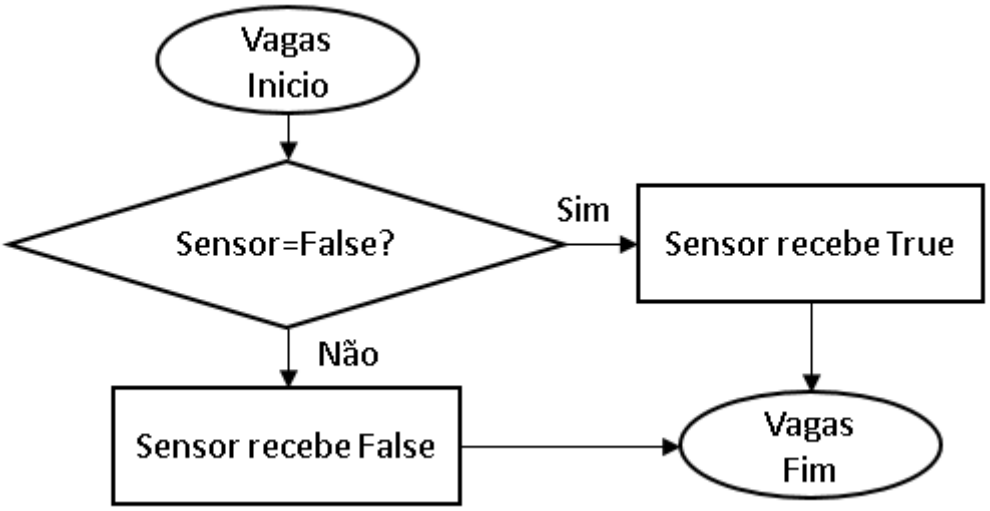


Figura 19 – Fluxograma da interrupção "Vagas"

Esta interrupção, foi utilizada na fase de simulações com o *Raspberry*, pois tem a função de simular a leitura dos sensores. Esta interrupção verifica o estado do sensor e troca seu estado. A Figura 20 mostra o fluxograma do código implementado no arquivo "Cria Vagas.txt".

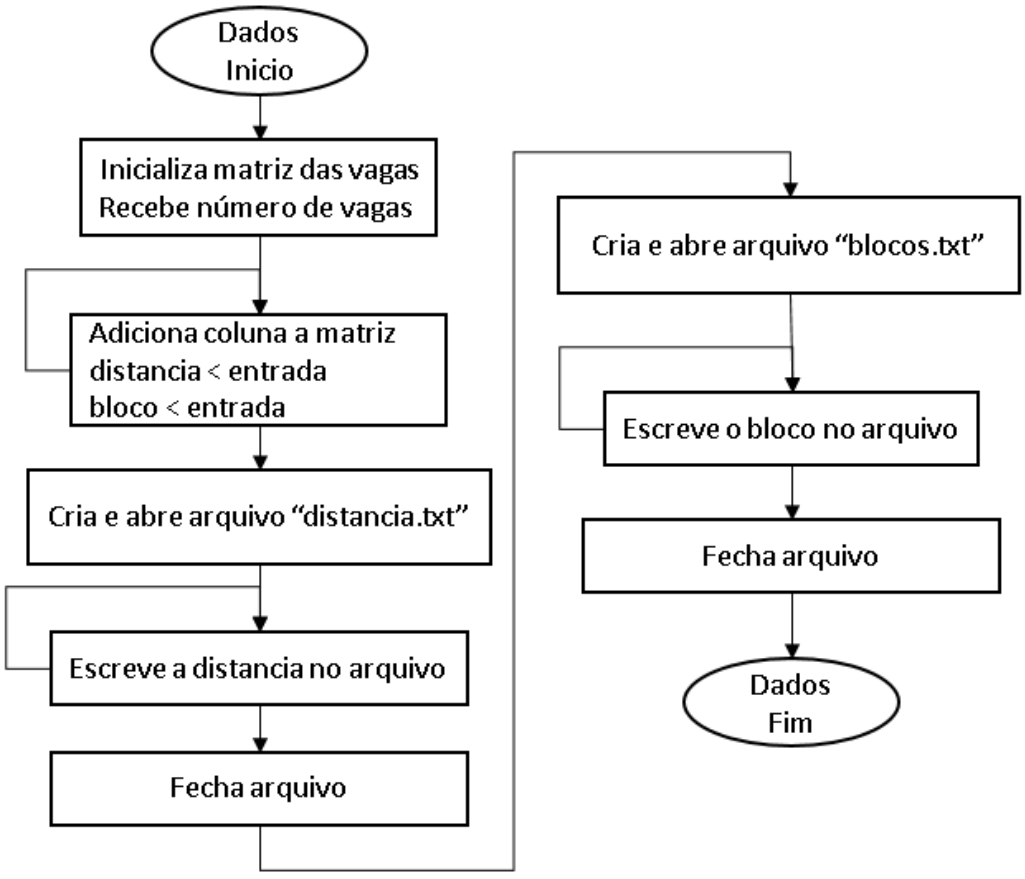


Figura 20 – Fluxograma do arquivo "Cria Vagas"

O código inicialmente solicita entrar com valor de vagas do estacionamento e em um *loop* da quantidade de vagas, cria uma coluna da matriz de vagas e solicita a entrada dos dados da distância e do bloco da vaga. Por fim, cria e abre um arquivo para cada dado, e o preenche com os dados digitados. Este arquivo foi criado para evitar que toda vez que o código principal seja executado, não seja necessário entrar com os dados novamente, pois eles serão carregados dos arquivos salvos.

3.2.7 – Implementação da Aquisição de Dados

Tendo finalizado a interface gráfica, decidiu-se implementar um sistema de aquisição de dados do estacionamento, possibilitando uma análise estatística, para isso, foi adicionado uma variável no vetor de cada vaga para armazenar a informação da hora de entrada, além disto, algumas funções e as interrupções foram modificadas para se acrescentar este sistema. No código da interrupção "Entrada" foi adicionado um comando para registrar a hora da entrada e armazenar esta informação na variável correspondente da vaga que os sistema escolheu.

Na função "Corrige", quando houver a alteração da seleção de vagas, a vaga anterior terá a variável da hora de entrada zerada, e a nova vaga receberá o horário da anterior.

Na interrupção "Saída", quando é acionada, além do desmarcar a vaga do veículo que está saindo, registra-se a hora da saída, calcula-se a diferença de tempo entre o momento da entrada e a saída, estas três informações são guardadas em um arquivo com o nome do dia e mês (Exemplo: 18-9), a Figura 21 mostra como o número da vaga mais as três informações (hora da entrada, tempo de permanência e hora da saída) ficam gravadas no arquivo, respectivamente.

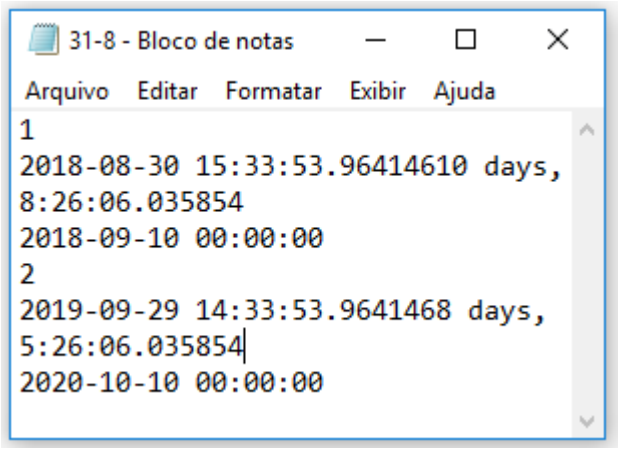


Figura 21 – Arquivo com exemplo dos dados obtidos na sáida de um veículo

Para evitar um erro ao abrir o arquivo da data, ao iniciar o sistema foi adicionado um processo que cria um arquivo com o nome da data do dia seguinte, para garantir que sempre exista um arquivo com o nome da data para ser aberto.

Após o sistema criar o arquivo com os dados de entrada e saída do dia, para ajudar a realizar uma análise destes dados é necessário um gráfico. Para isto, opto-se pela linguagem de programação JavaScript para gerar um gráfico com estes dados e dar 3 exemplos de análises que podem ser realizadas com estes.

Escolheu-se um código do Google Charts para gerar gráficos de linha, sendo necessário entrar com os vetores de dados no código. Para entrar com estes vetores de dados fez-se um código em Python para tratar os dados já guardados e assim criar um arquivo texto para ser adicionado no código em JavaScritp. Deste modo, ao executar o código em Javascript será possivel realizar uma análise estatistica do estacionamento.

Para análise, foram escolhidos 3 tipos de gráficos, o primeiro é para se analisar a quantidade de carros que tem no estacionamento em cada horário do dia, com isto é possivel ver quais horários que tem maior fluxo por exemplo. O segundo gráfico foi para analisar que momento do dia os carros ficam estacionados por mais tempo, e nisso utilizou-se os dados de permanência gerados no sistema e a contagem de veículos que entraram no estacionamento. Por ultimo, pode-se analisar as vagas que são mais utilizadas, através da contagem de veículos que utilizaram cada vaga. Todos os 3 gráficos são tratados e gerados pelos códigos no APENDICE C e no ANEXO A, respectivamente.

Lembrando que estes gráficos são exemplos de análises que podem ser feitas com os dados gerados pelo sistema, e cabe ao gerente do estacionamento decidir o que será analisando. Deste modo, o objetivo deste projeto é entregar os dados gerados pelos sistema.

Para o código de tratamento dos dados, a primeira parte é para preparar o gráfico de números de carros com a sua permanência no estacionamento. Como cada informação é escrita em uma linha do arquivo texto, é feita uma varredura somente nas terceiras linhas, pois são as que registram o tempo de permanência. Em seguida, é contado a quantidade de dados gerados, que são a quantidade de veículos que entraram no estacionamento. Por fim, é gerado um arquivo texto com o nome “PxN_(data)”(Permanência x Numero de Carros).

Posteriormente é preparado os dados para o gráfico de quantidade de carros presente no estacionamento com sua respectiva hora de entrada ou sáida. Para isto, foi feita uma varredura nas segundas e quartas linhas, que registram o horário de entrada e de saída, respectivamente e o horário é convertido para somente minutos. Nisto é criado 3 listas ordenadas, uma com todos os horários de entrada e sáida, outra com somente os de entrada e outra com somente os de saída. Em seguida, para cada horário, é somado 1 á uma variável para cada horário anterior de entrada e -1 para cada horario anterior de sáida, o resultado é a quantidade de carros naquele horário, em seguida é gerado o arquivo texto com nome “QxH_(data)”(Quantidade de Carros x Hora do dia).

Por fim, para tratar os dados para o gráfico de quantidade de carros que estacionaram em cada vaga é criado um vetor de zeros de 16 posições e é feita uma varredura nas primeiras linhas do arquivo texto, pois é a linha que registra o número da vaga e somado 1 no vetor da respectiva vaga que foi estacionado. Pra finalizar é gerado um arquivo texto com o nome “QxV_(data)”(Quantidade de carro x Cada Vaga). Todos estes arquivo são gerados entre colchetes e virgula, para poderem ser adicionados diretamente no código em JavaScript que por sua vez pode ser executado nos principais navegadores de internet. A Figura 21 mostra um gráfico de quantidade de carros versus hora do dia com os dados obtidos em testes.

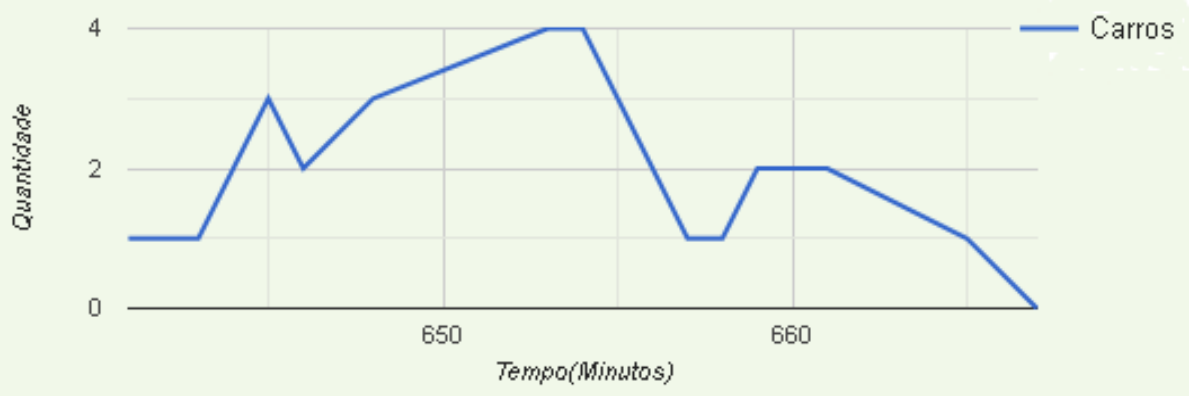


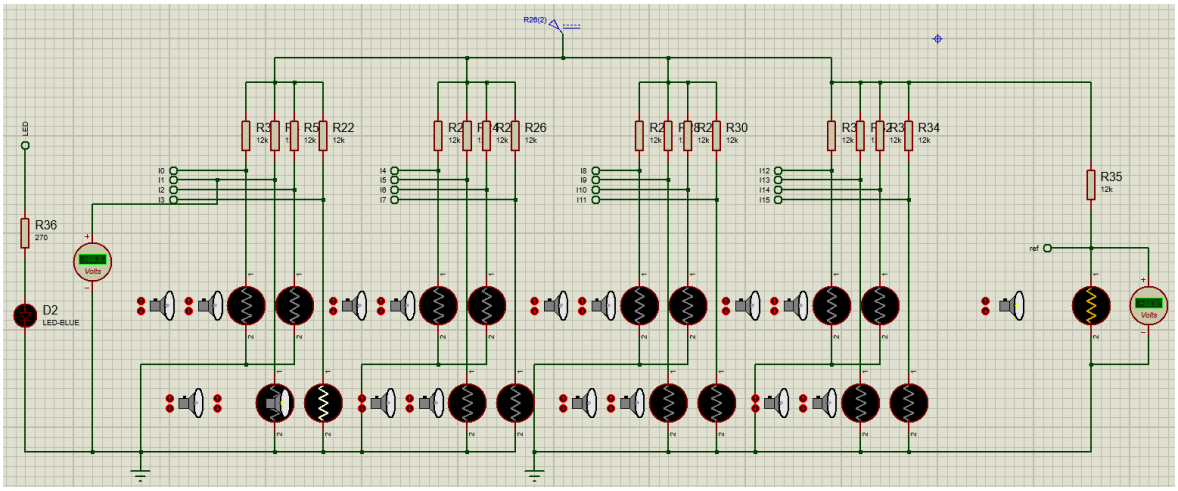
Figura 21 – Gráfico de linha para análise estatística

3.2.8 – Implementação Eletrônica

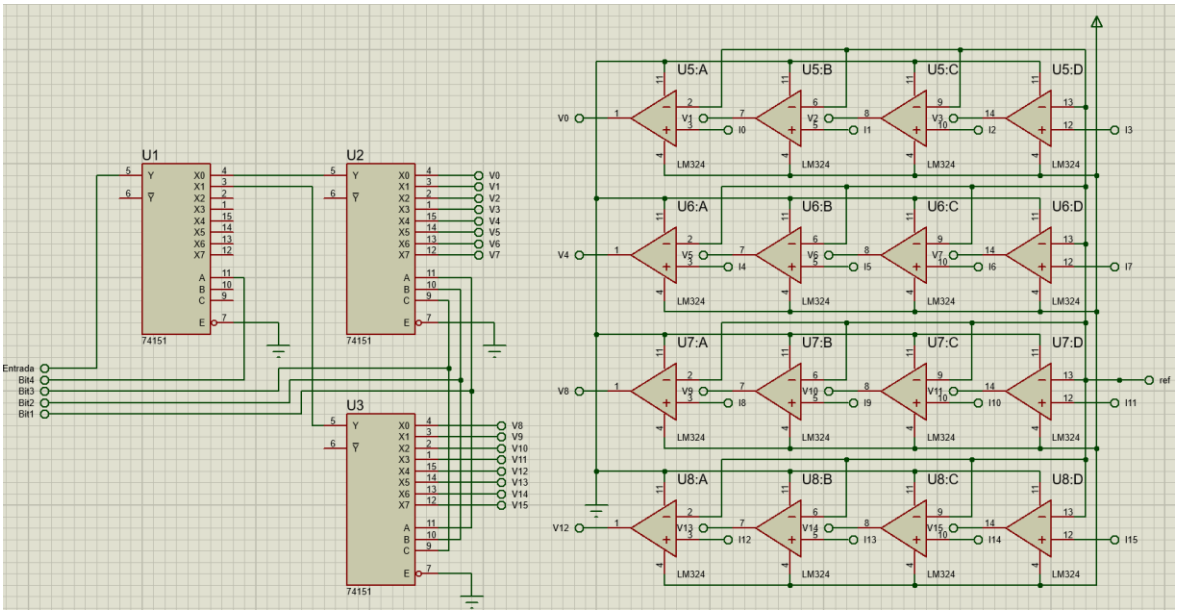
3.2.8.1 – Simulação

Sobre a parte eletrônica, utilizando o modelo mais simples do sensor (somente 1 comparador), foi montado um circuito na *protoboard* para testar o circuito sensor na prática. Para isso foi utilizado um LED para indicar quando a vaga está preenchida.

Em seguida, foi realizada uma simulação do circuito sensor completo no Proteus, ou seja, todos os 16 sensores conectados aos multiplexadores e estes controlados pelo *Raspberry*, como pode ser visto na Figura 22.



(a)



(b)

Figura 22 – Simulação do circuito sensor completo - (a) Parte dos sensores e (b) parte dos amplificadores e dos multiplexadores

No circuto da Figura 22.a estão os LDRs e seus resistores, na Figura 22.b estão os amplificadores e os multiplexadores. Os pinos no canto esquerdo da Figura 22.b estão representados os pinos que se conectam no *Raspberry*, os 4 pinos de controle dos multiplexadores e a entrada do sinal lógico do sensor selecionado.

3.2.8.2 – *Layout*

No Eagle foram feitos os *layouts* das placas dos multiplexadores e a do amplificador com os sensores. Porém, pelo fato de ser mais fácil verificar e corrigir erros elétricos, foi optado utilizar uma placa de matriz de furos, ao invés da placa de fenolite, para a confecção das placas dos sensores. Já o circuito dos multiplexadores foi montado em uma *protoboard*. Nas Figuras 21 e 22 a seguir podem ser vistos os *layouts* das placas.

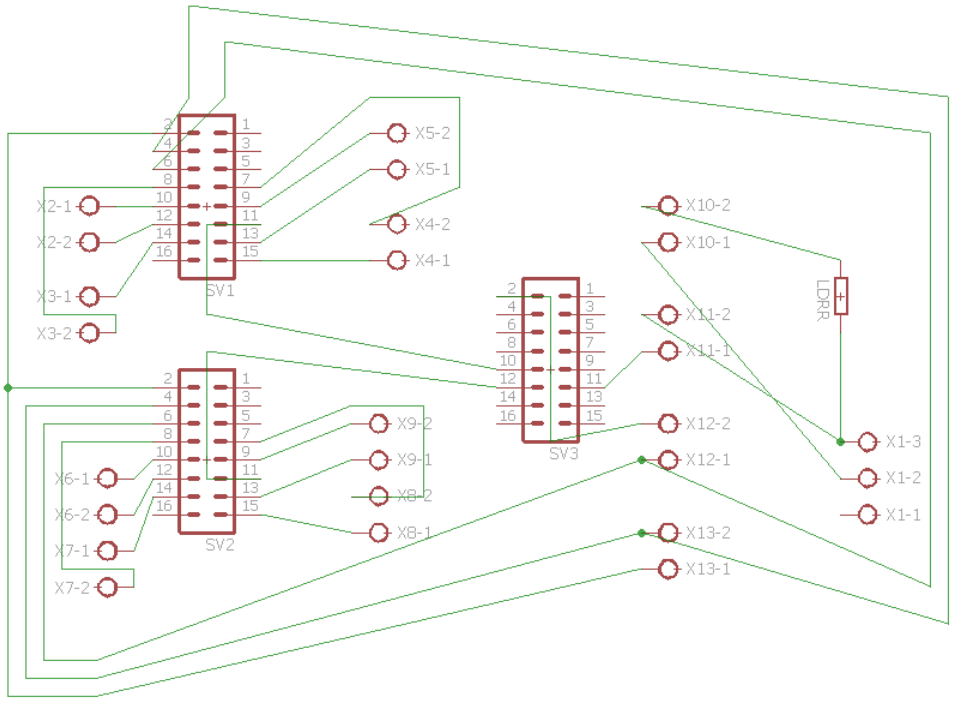


Figura 23 – Desenho da placa dos multiplexadores

Os componentes SV1, 2 e 3 são os multiplexadores, o X1 representa o potenciômetro, de X2 a X9 são os pinos de entrada dos sensores, X10-2 é o VCC X10-1 é o terra, X11-2 é saida do sensor de referência, X11-1 é o pino de saída dos MUXs que vai para o *Raspberry*, X12 e X13 são as entradas de controle dos MUXs que vem do *Raspberry*.

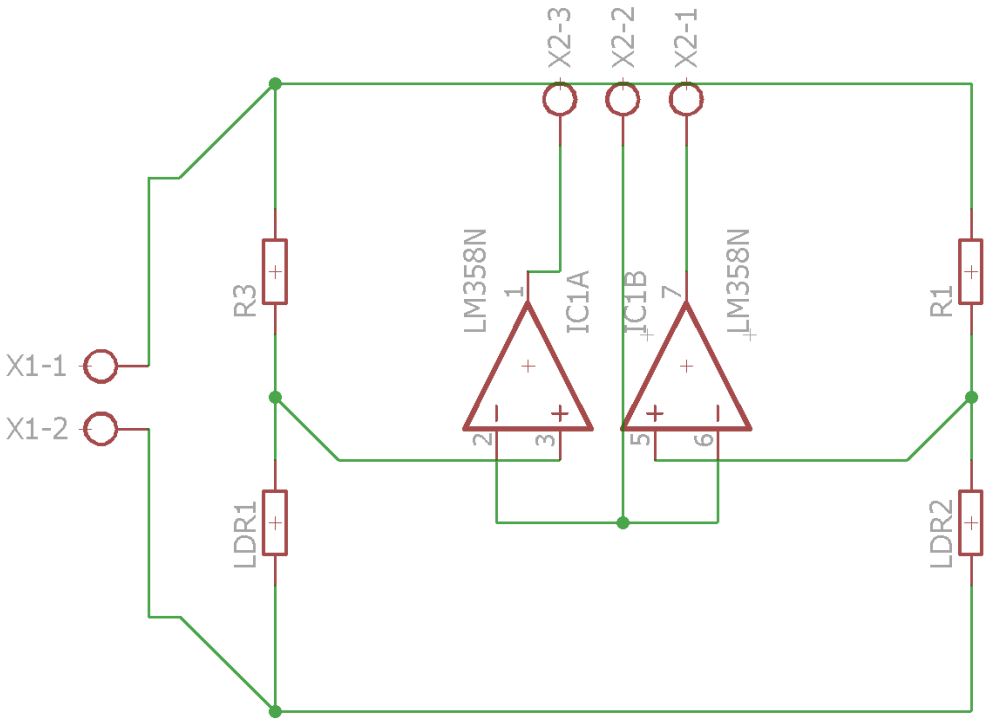


Figura 24 – *Layout* placa do Aplificador e sensores

Os pinos X1-1 e X1-2 são VCC e terra, respectivamente, enquanto os pinos X2-3 e X2-1 são o sinal lógico de saída dos sensores e X2-2 e o pino que recebe o sinal do sensor de referência.

3.2.8.3 - Montagem

Na montagem do circuito, todo o sistema foi dividido em três partes, o *Raspberry*, a placa dos multiplexadores junto com o LDR de referência (foi montado na *protoboard*) e as placas de cada amplificador, que possui 2 conjuntos de sensores LDR cada.

Depois, foi realizado os testes de cada placa (sensores e multiplexadores) em separado e corrigido os erros encontrados. Em seguida foi testado o funcionamento das GPIOs e então foi verificado o funcionamento de todo o sistema conectado ao *Raspberry*.

Por fim, foram realizadas algumas modificações no código anterior da interface gráfica para funcionar agora com os sensores reais, para isso, os botões das vagas forão desativados, mantendo somente a alteração das cores, foi adicionado a bibiloteca das GPIOs e a configuração dos pinos utilizados, no código da varredura dos sensores foi feito um *loop* em que o *Raspberry* escolhe um sensor de cada vez e altera o valor da variável que o representa, de acordo com o valor lido no sensor. No final ele verifica se algum sensor está ativo sem ter sido

selecionado, caso contrário ele troca a vaga selecionada para aquela em que o sensor foi ativado, para evitar que um carro estacione em uma vaga não selecionada e a vaga selecionada continue selecionada. Por fim, atualiza as cores das vagas e as tarefas da interface gráfica.

4- RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados os resultados obtidos neste trabalho, além das observações e considerações.

4.1- SISTEMA SENSORIAL

O sistema proposto para atender as 318 vagas pode ser visto na Figura 25.

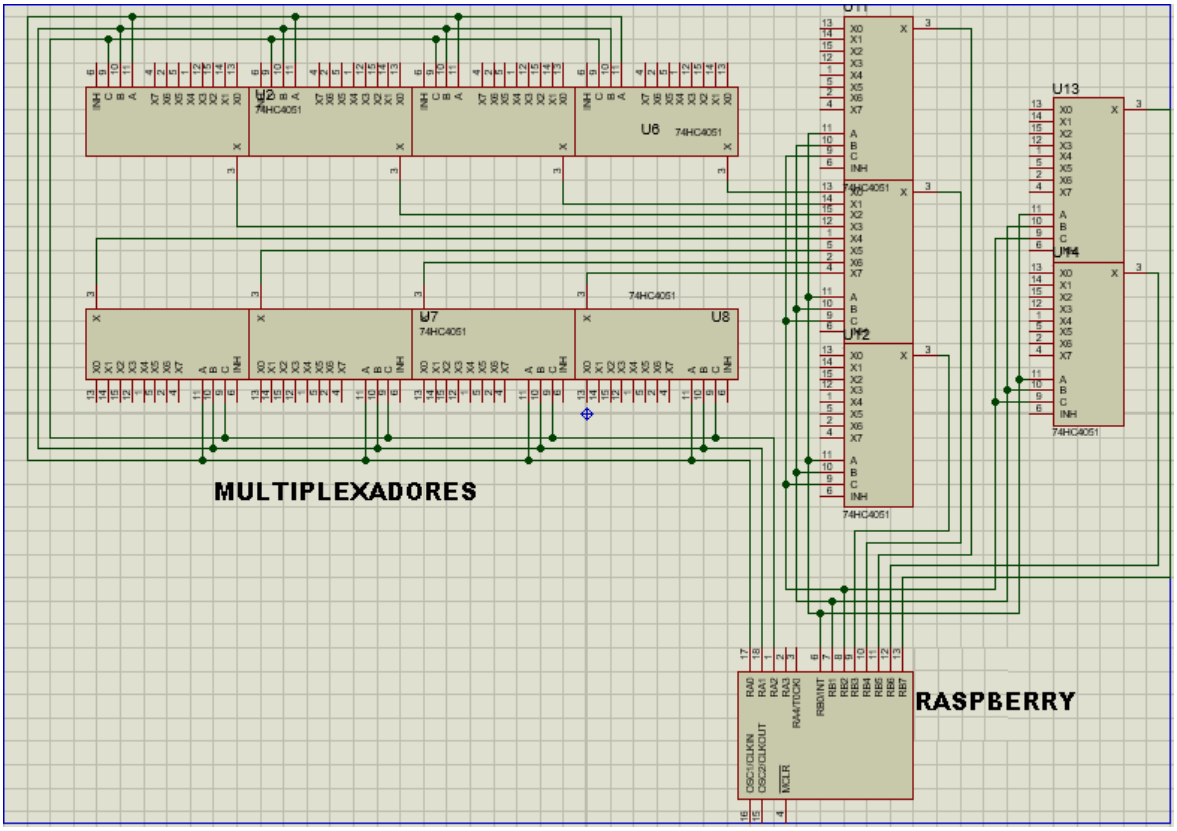


Figura 25 – Sistema sensorial montado no Proteus

A Figura 25 apresenta apenas 13 multiplexadores, mas é possível observar que 8 deles na horizontal estão conectados a outro na vertical, o outros que estão na na vertical também teriam 8 multiplexadores conectados a cada um deles, o que totaliza os 45. Cada multiplexador na horizontal está conectado a 8 sensores, cada um destes multiplexadores está conectado à outro na vertical, como são 320 sensores, se dividirmos por 8 teremos o número de 40 multiplexadores necessários

na horizontal, se dividirmos por 8 teremos 5 na vertical. Deste modo, com 11 pinos do *Raspberry* é possivel monitorar os 320 sensores. Seria possível diminuir a quantidade necessária de pinos para 8, se colocarmos uma terceira cascata com 1 multiplexador, mas a ideia é utilizar uma quantidade mínima de multiplexadores para aproveitar o máximo de pinos do *Raspberry*. Lembrando que de acordo com o número de pinos disponíveis do *Raspberry* é possível aumentar a quantidade de vagas para mais de 500, por exemplo, pois para cada multiplexador adicionado na segunda cascata, 64 vagas são incluidas, sendo que o ideal seria 8 por cascata, teriamos 512 vagas, ou seja, com uma terceira cascata, cada multiplexador adicionaria 512 vagas.

Este sistema possui duas cascatas de multiplexadores, uma cascata de 5 e outra de 40 (8 para cada 1 dos 5), por isso é possivel observar que no *Raspberry*, 6 (3 para cada cascata) pinos são responsáveis pelo controle dos multiplexadores e 5 pinos para receber os dados dos sensores.

Portanto, tem-se 64 vagas monitoradas por cada um dos 5 multiplexadores (total de 320 vagas) e para controlar e receber os dados de tudo isso foi necessário 6 pinos de controle e 5 de entrada, ou seja, um microcontrolador é o suficiente para todo o sistema sensorial.

4.2- SISTEMA SENSORIAL DE TESTE

O sistema sensorial básico testado no Proteus pode ser visto nas Figura 26 e 27.

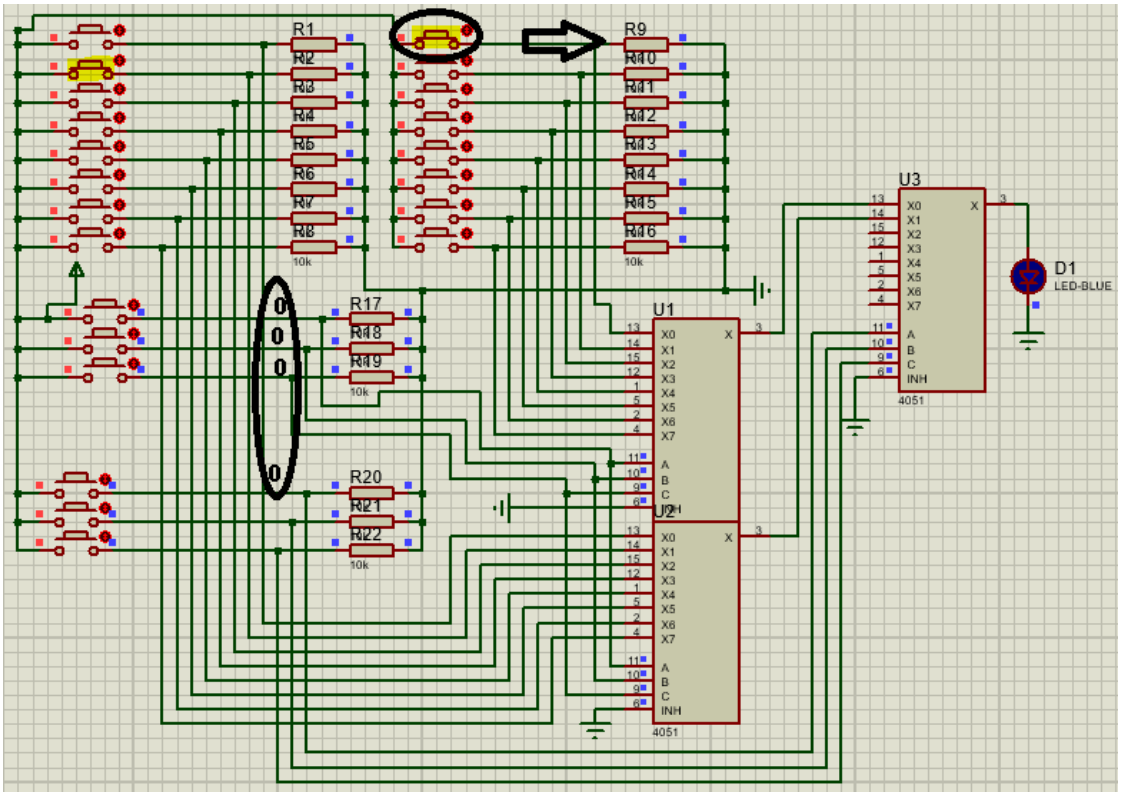


Figura 26 – Simulação do sistema sensorial para U1

Para enviar o sinal do resistor R9, os botões de controle deveriam enviar 0000 em binário, pode se observar que o LED está ligado porque o resistor R9 está acionado, portanto o teste funcionou como esperado.

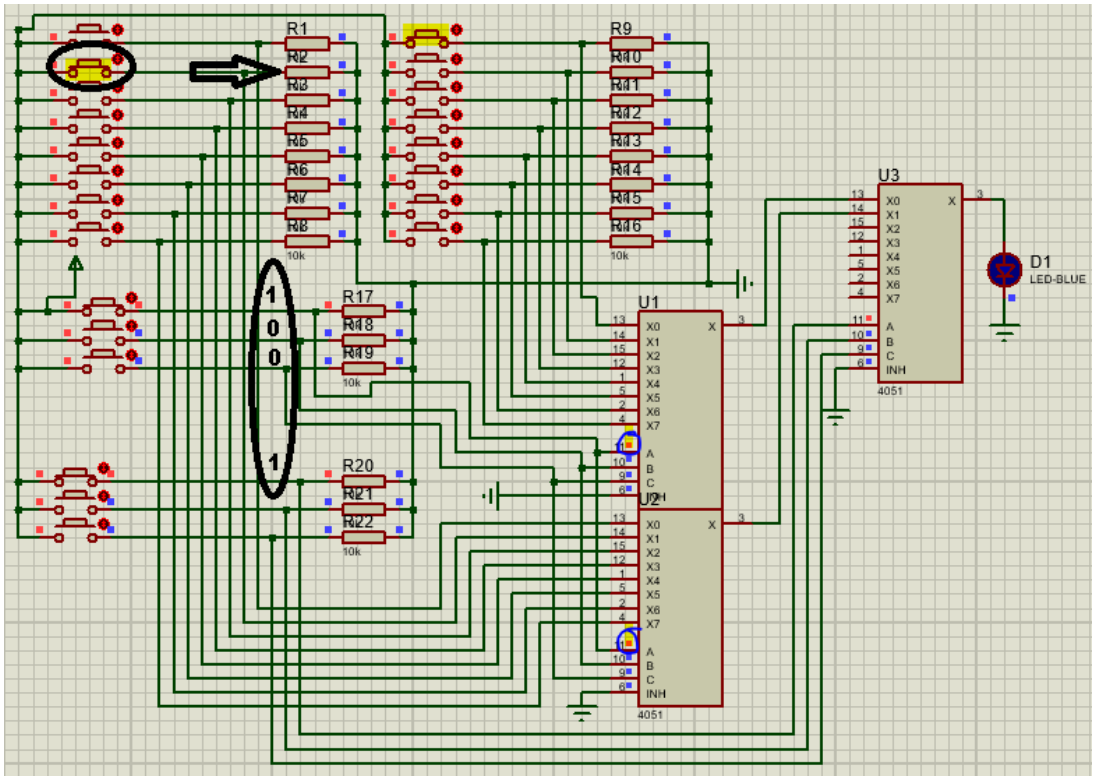


Figura 27 – Simulação do sistema sensorial para U2

Para enviar o sinal do resistor R2, os botões de controle deveriam enviar 1001 em binário, como pode ser visto na Figura 27. Novamente, o LED está acesso pois o botão do R2 está acionado e ele foi o selecionado para ter o sinal na sáida, portanto o teste funcionou como esperado.

Com estes testes foi possível verificar o funcionamento do sitema projeto para os multiplexadores e o próprio modo de funcionamento dos multiplexadores.

4.3- CIRCUITO SENSOR LDR

Nos testes do circuito sensor com um amplificador e dois sensores LDR foi observado que utilizar um LDR de referência para muitos sensores pode ser problemático. Isto porque as vezes alguns sensores estavam ativos sem a adição de sombra, o que exigia regular o potenciômetro do LDR de referência reduzindo sua resistência. Este problema ocorre nos casos de mau posicionamento do LDR sensor e o da referência ou quando a luminosidade era fraca, forçando assim, o ajuste do potenciômetro a uma resistência tão pequena que mesmo sob sombra o sensor não acionava. Porém, ao posicionar bem os sensores e mantendo um boa luminosidade neles, os sensores funcionam rápido e corretamente. A Figura 28 mostra a montagem feita com um LED na saída dos sensores para realizar os testes.

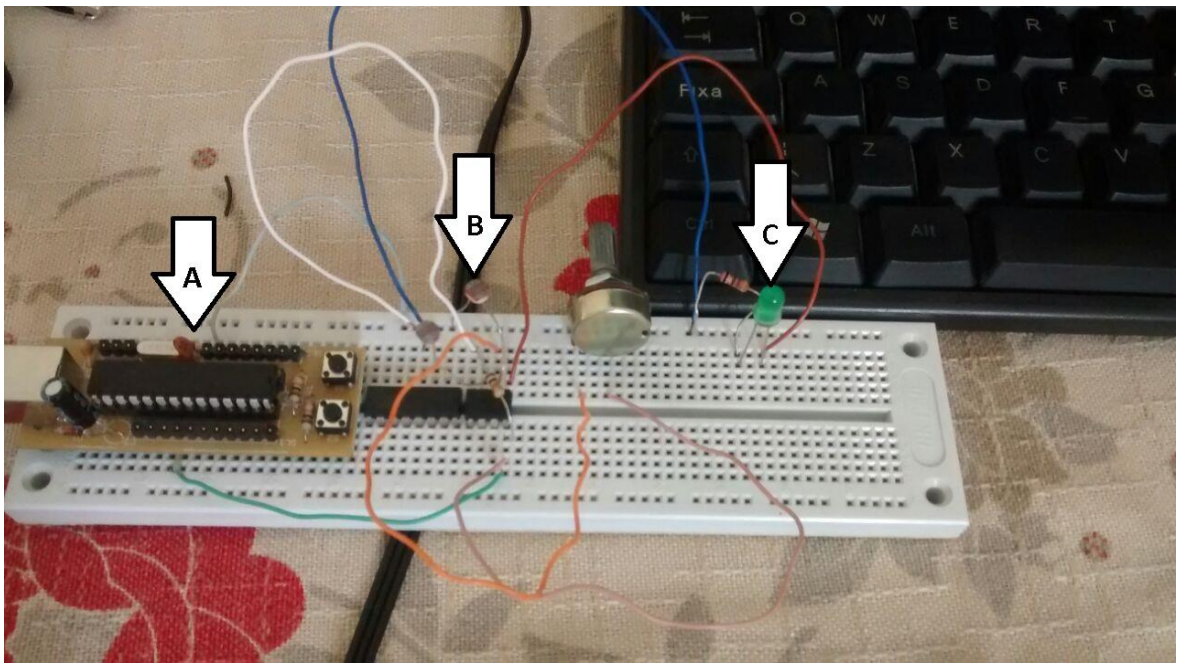


Figura 28 – Teste do circuito sensor

LEGENDA: A - Microcontrolador, B – LDR, C - LED

A presença de um microcontrolador a esquerda da Figura 28 é somente para alimentar o circuito. Neste teste não se teve o cuidado de utilizar fios curtos pelo fato de que seriam feitas as placas dos sensores posteriosmente.

Sobre o teste realizado sobre as tensões nos LDR, a Figura 29 mostra a forma de onda obtida no osciloscópio.

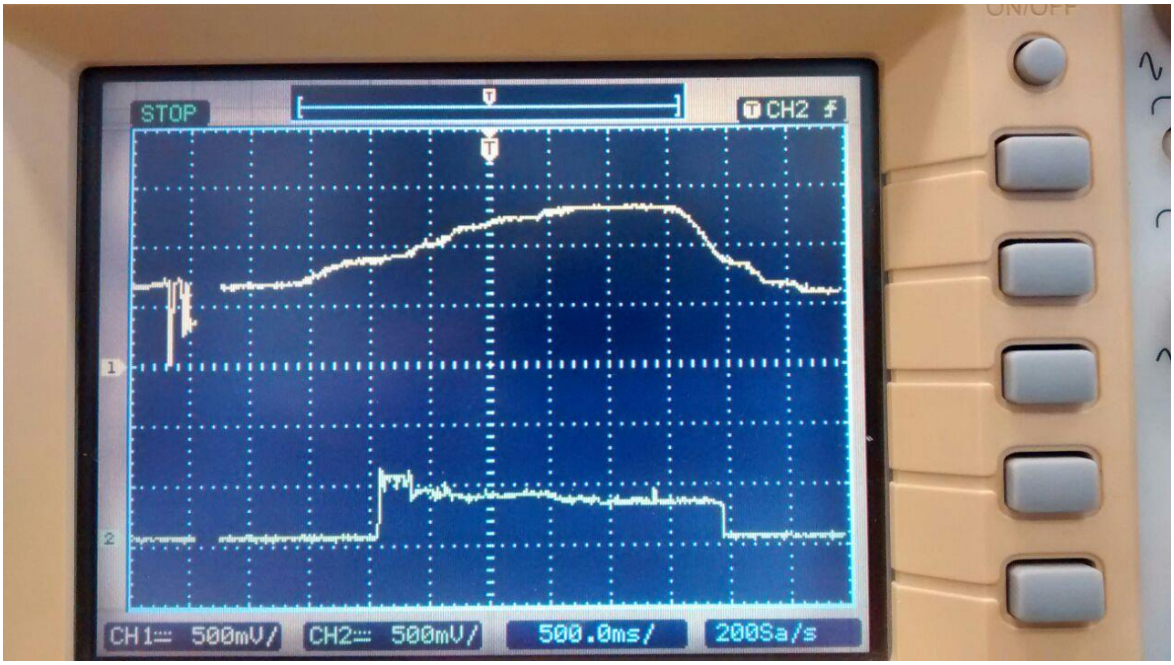


Figura 29 – Tensão no LDR sensor e na saída

A forma de onda superior é a tensão no LDR e a inferior é a da saída, pode-se observar que no momento em que a tensão do LDR ultrapassa 900mV a saída é acionada com aproximadamente 500mV de tensão, quando a tensão do LDR volta a ter menos de 900mV a saída retorna para 0V. O valor de 500mV medido na saída provavelmente é devido à algum errode configuração na medição. Pois quando foi medido no multímetro o valor foi consideravelmente maior com será mostradado a seguir.

Considerando o teste feito para verificar qual a tensão que o *Raspberry* entende como nível lógico alto, a Figura 30 a seguir mostra os valores de tensão obtidos no teste.

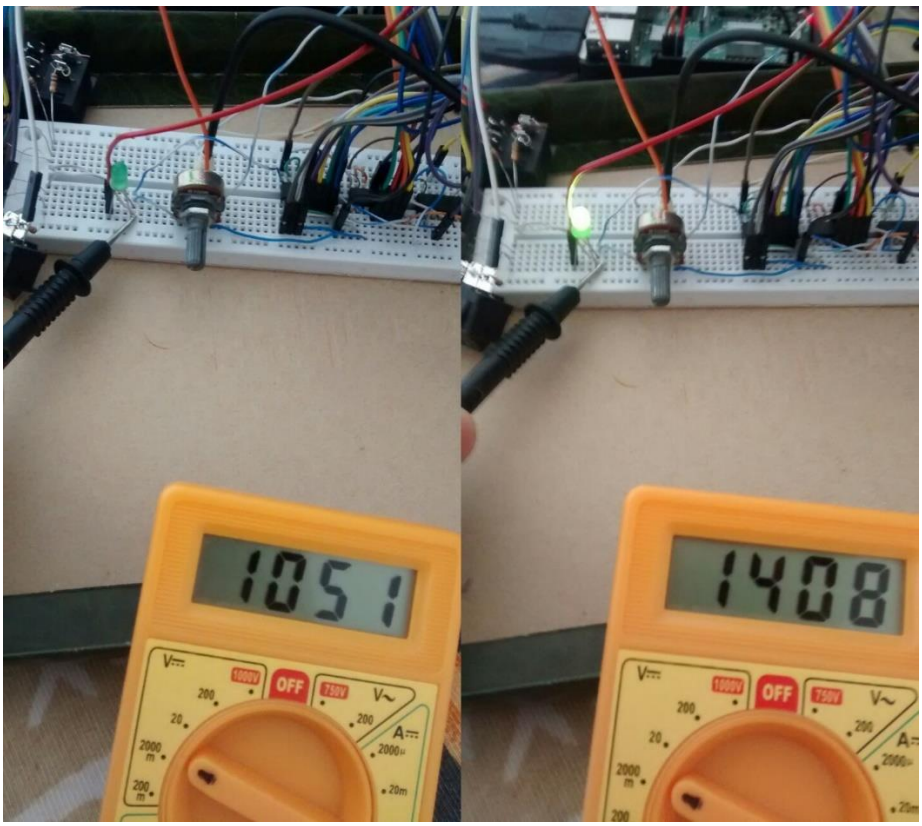


Figura 30 – Tensões limiares para nível lógico no *Raspberry*

Quando os valores de tensão estavam entre 1051mV e 1408mV o LED não ficava estável. Com isso, qualquer valor de tensão acima de 1408mV ou abaixo de 1051mV que esteja no pino será confiável, considerando que a tensão de alimentação é de 3.3V.

Para garantir a confiabilidade do sensor LDR, a menor medição de tensão obtida nas saídas dos amplificadores foi de 1945mV, ou seja, bem acima de 1408mV, garante-se que quando o carro estiver sobre o sensor, o *Raspberry* irá entender como nível lógico alto. Uma pequena barra preta foi utilizada para cobrir o LDR e simular a sombra do carro.

Todos estes valores foram obtidos com uma alimentação de 3.3V, mas ainda que a tensão de alimentação fosse 5V o resultado seria o mesmo, pois a saída do amplificador depende da comparação das tensões, a diferença estaria somente na resolução da resistência pela tensão no LDR.

4.4- INTERFACE GRÁFICA

O estacionamento coberto, mostrado na Figura 10, foi escolhido como base para se implementar a interface gráfica da Figura 31. Pode-se observar que ela

possui somente 16 vagas funcionais, sendo que as outras vagas estão marcadas com “xx” para representar as vagas do estacionamento, mas somente as numeradas são utilizadas. Ná pratica, as mudanças necessárias seriam a alteração do número total de vagas e inserir as distâncias e o bloco de cada uma delas e a alteração visual da interface gráfica inserindo os seus respectivos números. Também necessitaria de mudança no circuito, aumentando o número de sensores e multiplexadores. No APENDICE B está o código de configuração da interface gráfica.

Nele também está configuração das dimensões e do título da interface, além da configuração de alguns *labels* com suas dimensões, cores, posicionamento e texto. Além do que é mostrado no quadro, o código também configura os botões.

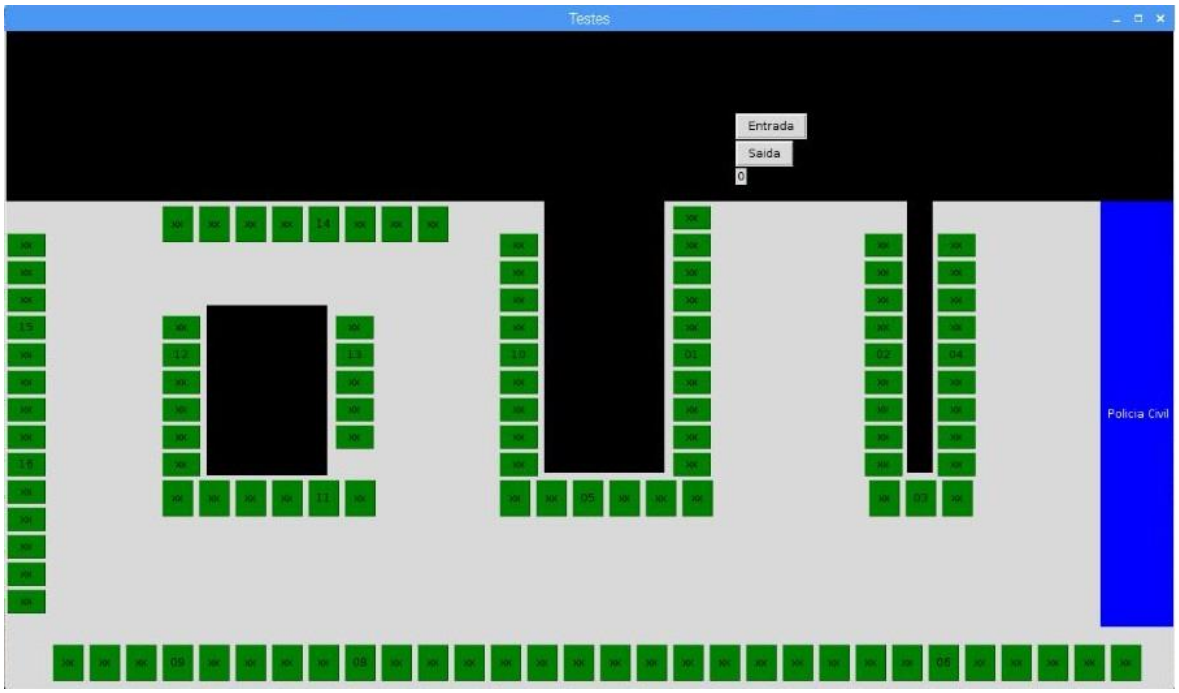


Figura 31 – Interface gráfica pragramada no *Raspberry*

As vagas tem 3 cores para representar seus *status*, vermelho e verde representam os sensores ativados e desativados, respectivamente. A vaga em amarelo representa uma vaga que foi escolhida pelo sistema e ficarará vermelha somente quando o motorista ocupar a mesma. Quando ele sair da vaga ela fica amarela e somente quando sair do estacionamento é que a vaga voltará a ficar verde. A área em preto representa a parte estrutural do estacionamento a azul representa as vagas destinadas a Polícia Civíl e a branca representa a região transitável. Para contar a quantidade de carros no estacionamento, o algoritimo verifica quantas vagas estão selecionadas. Para simular este sistema, as vagas

são botões que ao serem pressionados alternam as cores de verde para vermelho e vice-versa, na prática eles não realizam nada. O botão da entrada (na simulação) abre a linha de comando para escolher o bloco que deseja estacionar e o algoritimo encontra a vaga a mais próxima e muda sua cor para amarelo.

A orientação do cliente deverá ser feita com placas indicando a direção dos blocos e dos números da vagas, como é feito no trânsito, por exemplo, tendo o cliente que memorizar o bloco e o número da sua vaga e se orientar pelas placas que devem ser colocadas nas posisões em que houver mais de uma direção possível a se seguir.

4.5 – TESTE INTEGRADO

Após os testes de funcionamento de cada placa dos sensores e da placa dos multiplexadores, todos foram conectados e em seguida foi realizado o teste de todas as partes conectadas (placas de sensores, circuito dos multiplexadores, *Raspberry* e a interface gráfica), a Figura 32 mostra todas as partes conectadas.

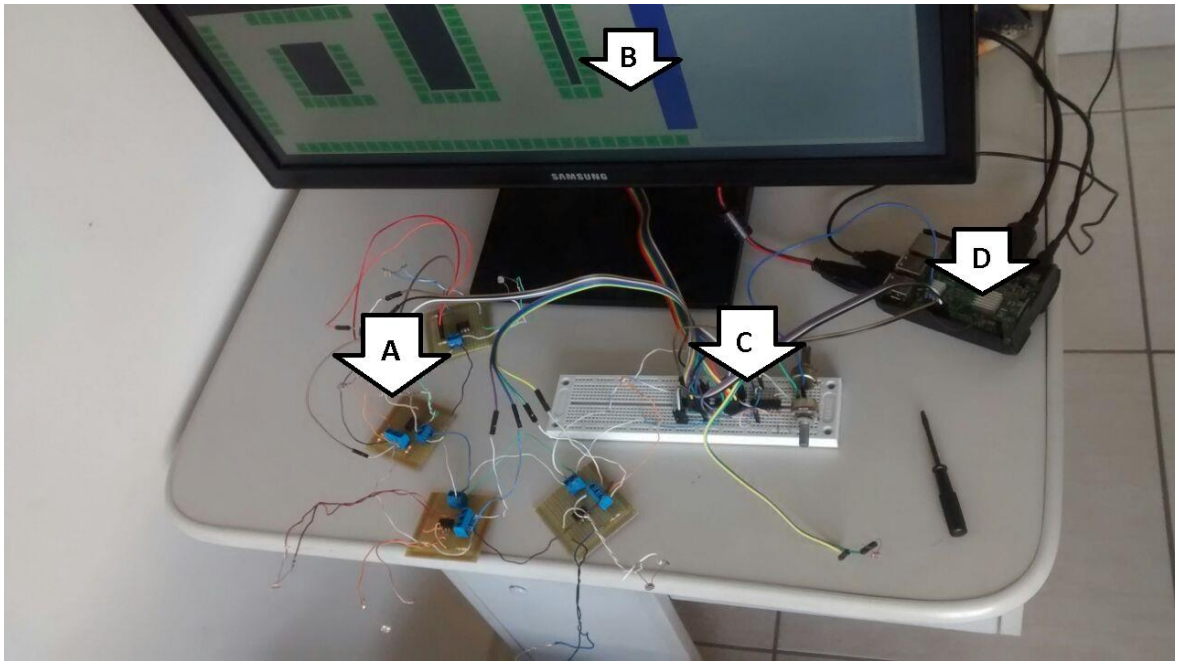


Figura 32 – Sistema completo em funcionamento
LEGENDA: A – Sensores, B – Monitor, C – Multiplexadores, D – *Raspberry*

Na Figura 32 também é possível ver as 4 placas de sensores (A), com 2 sensores em cada placa, o circuito dos multiplexadores e do sensor de referência (C), o *Raspberry* (D) e a interface gráfica vista no monitor (B).

Nos testes das GPIOs verificou-se o funcionamento dos circuitos sendo alimentados com 3V e 5V, já que o *Raspberry* tem a opção dos dois. Foi tomado o cuidado de colocar um poteciômetro no pino de entrada do *Raspberry* pra dividir a tensão de 5V para 3V, pois ele não suporta acima disto.

Nos testes das placas dos multiplexadores foi observado que para seu correto funcionamento, os pinos VEE e INH devem ser aterrados, sendo que anteriormente não estavam conectados, pois não são utilizados, mas ainda sim é necessário aterrá-los.

5- CONCLUSÕES

Com base nas pesquisas realizadas, nenhum dos trabalhos acadêmicos encontrados envolvia o número de vagas reais de um estacionamento, muito menos indicava a vaga mais próxima para o motorista, portanto este é um dos pontos diferenciados deste trabalho.

O LDR apresentou inúmeras vantagens em relação a maioria dos trabalhos levantados pela revisão bibliográfica, principalmente por ser mais barato, a implementação ser mais simples e ter maior alcance de medição se comparado com o sensor infravermelho ou o ultrassônico.

O sistema de multiplexadores é uma excelente ferramenta para monitorar muitos sinais digitais, diminuindo a quantidade de pinos necessários.

A grande flexibilidade do *Raspberry Pi 3*, no que diz respeito aos periféricos unido com a aplicabilidade dos pinos GPIOs agiliza muito a forma de receber, tratar e comandar sinais.

6- PROPOSTAS PARA TRABALHOS FUTUROS

Para a continuação deste projeto, propõe-se a implementação de um código que disponibilize as informações das vagas no estacionamento via internet para consulta. Também propõe-se a implementação de um sistema de pagamento automatizado.

Pelo fato do sistema sensorial e da comunicação do mesmo com o *Raspberry* ser via cabo, também propõe-se a implementação de uma comunicação sem fio entre os sensores e o computador.

A parte de geração dos gráficos estatisticos foi de exemplo, portanto propõe-se um trabalho mais elaborado para se fazer a geração e análise dos gráficos.

7- REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Jonathan Tomaz; CAPOVILLA, Galesandro Henrique. Protótipo de estacionamento autônomo com integração multiplataforma. Revista Ciência e Tecnologia, v. 18, n. 32, 2015.

BANDEIRA, Thayanne Barros et al. PROTÓTIPO DE ESTACIONAMENTO AUTOMATIZADO UTILIZANDO MODELO COMPUTACIONAL MATRICIAL E MICROCONTROLADOR ARDUINO. Blucher Mathematical Proceedings, v. 1, n. 1, p. 817-824, 2015.

BARONE, Rosamaria Elisa et al. Architecture for parking management in smart cities. IET Intelligent Transport Systems, v. 8, n. 5, p. 445-452, 2013.

CAICEDO, Felix; VARGAS, Jorge. Access control systems and reductions of driver's wait time at the entrance of a car park. In: Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on. IEEE, 2012. p. 1639-1644.

CAVAMURA, Humberto Fernando Massaharu; MITSUHASHI, Marlos Kenjy. Sistema de gerência de vagas de estacionamento. 2014. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

CHAVES, G. C. 2010. Estacionamento – um negócio da China! (e do Brasil, dos EUA, da Índia...). G9 Investimentos disponível em: https://www.g9investimentos.com.br/biblioteca/estacionamento-um-negocio-da-china-e-do-brasil-dos-eua-da-india Acesso em: 7 de maio. 2018.

DEVELOPER MICROSOFT. “Pinmappingsrpi”. Disponível em:< https://developer.microsoft.com/en-us/windows/iot/docs/pinmappingsrpi>. Acesso em 3 de junho de 2017.

«FAQs»*. Raspberry Pi Foundation*.

GADZOVIC, Almir; LEKIC, Nedeljko; RADUSINOVIC, Igor. Automation of parking services using UHF RFID technology and magnetometers. In: 2013 21st Telecommunications Forum Telfor (TELFOR). 2013.

GALLINDO, G. et al. Control System for Parking Management. IEEE Latin America Transactions, v. 9, n. 6, p. 916-920, 2011.

GONÇALVES, Pablo Henrique. Controle de vagas de estacionamento com cadastro biométrico para vagas de deficientes. 2016.

IBGE. “Pesquisa Industrial Mensal – Produção Física – PIM - PF” Disponível em: https://www.ibge.gov.br/estatisticas-novoportal/economicas/industria/9294-pesquisa-industrial-mensal-producao-fisica-brasil.html?edicao=19776&t=destaques. Acesso em: 10 de julho. 2018.

LDR/ LGIHT DEPENDENT PHOTO RESISTOR 8mm. Disponível em:< http://projectshopbd.com/product/ldr-big-l12/>. Acesso em :14 de novembro de 2017.

LDR(ALM332). Disponível em:< http://www.newtoncbraga.com.br/index.php/almanaque-tecnologico/201-l/7547-ldr-alm332>. Acesso em :14 de novembro de 2017.

LMx58- LOW POWER, DUAL-OPERATIONAL AMPLIFIERS. Disponível em:< http://www.ti.com/lit/ds/symlink/lm158-n.pdf>. Acesso em :14 de novembro de 2017.

LUNDH, Fredrik. An introduction to tkinter. URL: www. pythonware. com/library/tkinter/introduction/index. htm, 1999.

MCLEOD, Alastair. Digital video storage for security applications. In: Security Technology, 1995. Proceedings. Institute of Electrical and Electronics Engineers 29th Annual 1995 International Carnahan Conference on. IEEE, 1995. p. 382-387.

MENEZES, Nilo Ney Coutinho. Introdução à programação com Python–2ª edição: Algoritmos e lógica de programação para iniciantes. Novatec Editora, 2016.

NEWARK. "*Raspberry Pi 3 Model B*". Disponível em:< http://www.newark.com/buy-raspberry-pi?rd=raspberry%20pi&anyFilterApplied=false>. Acesso em: 27 de abril de 2017.

NUKANO, TOMOHIKO; FUKUMI, MINORU; KHALID, MARZUKI. Vehicle license plate character recognition by neural networks. In: Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004. Proceedings of 2004 International Symposium on. IEEE, 2004. p. 771-775.

OLIANI, Vitor; MIYOSHI, Juliana. Estacionamento vertical. Revista Ciência e Tecnologia, v. 18, n. 32, 2015.

PUERARI, Rosicler Felippi; TECCHIO, Fernando. AUTOMAÇÃO DE UM ESTACIONAMENTO DE VEÍCULOS UTILIZANDO ARDUINO. Seminário de Iniciação Científica, Seminário Integrado de Ensino, Pesquisa e Extensão e Mostra Universitária, 2016.

Raspberry Pi. "*Raspberry Pi 3 Model B*". Disponível em <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acesso em 10 de julho de 2018.

ROY, Amit et al. Smart traffic & parking management using IoT. In: Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual. IEEE, 2016. p. 1-3.

SIRITHINAPHONG, Thanongsak; CHAMNONGTHAI, Kosin. The recognition of car license plate for automatic parking system. In: Signal Processing and Its Applications, 1999. ISSPA'99. Proceedings of the Fifth International Symposium on. IEEE, 1999. p. 455-457.

SUNITHA, K. A. et al. Fuzzy based automatic multi-level vehicle parking using lab view. In: Frontiers in Automobile and Mechanical Engineering (FAME), 2010. IEEE, 2010. p. 363-367.

SILVA, Luan Gabriel da; JESUS, Ricardo Krohn de. Plataforma para controle de estacionamento municipal. 2016.

TEXAS INSTRUMENTS. “CD4051B, CD4052B, CD4053B”. Disponível em:< https://global.oup.com/us/companion.websites/fdscontent/uscompanion/us/pdf/microcircuits/students/logic/cd4051-ti.pdf>. Acesso em: 3 de junho de 2017.

APÊNDICE A

Código para criar os arquivos com as informações das distancias da entrada do estacionamento até cada vaga e o respectivo setor do estacionamento que ela está.

```
vagas=[]
nv=int(input("Numero de vagas: "))
for i in range(nv):
    x=i+1
    vagas.append([x,0.0,False,"nd",False])
    vagas[i][1]=float(input("Distancia de %d: "%x,))
    vagas[i][3]=input("Bloco de %d:"%x,)
    print()

arquivo = open("distancia.txt","w")
for l in range(nv):
    arquivo.write(str(vagas[l][1]))
    arquivo.write("\n")
arquivo.close()

arquivo = open("blocos.txt","w")
for l in range(nv):
    arquivo.write(vagas[l][3])
arquivo.close()
```

APÊNDICE B

Código principal que gera a interface, monitora o estado das vagas, salva os dados estatísticos do estacionamento e gerencia a entrada e saída dos veículos como já foi explicado anteriormente.

```
from tkinter import *
from functools import partial
import time as delay
import RPi.GPIO as GPIO
from datetime import datetime,timedelta
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(29,GPIO.OUT)
GPIO.setup(31,GPIO.OUT)
GPIO.setup(33,GPIO.OUT)
GPIO.setup(35,GPIO.OUT)
GPIO.setup(37,GPIO.IN)

# Cria arquivo dia seguinte
dt=datetime.now()+timedelta(days=1)
arquivo = open(str(dt.day)+"-"+str(dt.month),"w")
arquivo.close()

# Cria matriz a partir dos arquivos
vagast=[]
arquivo = open("distancia.txt","r")
x=0
t=0
for l in arquivo.readlines():
    x=x+1
    vagast.append([x,float(l),False,"nd",False,t])
arquivo.close()
x=0
arquivo = open("blocos.txt","r")
for l in arquivo.read():
```

```
    vagast[x][3]=l
    x=x+1
arquivo.close()
print(vagast)

# Calcula total de vagas
def totaliza():
    total=0
    for i in range(16):
        if vagast[i][2]==True:
            if vagast[i][4]==True:
                total=total+1
        if vagast[i][2]==False:
            if vagast[i][4]==True:
                total=total+1
        lbt["text"]=str(total)

# Atualiza a cor do botão selecionado
def atualiza(num):
    if bool(vagast[num-1][2])==False:
        if bool(vagast[num-1][4])==False:
            bt[num]["bg"]="GREEN"
    if bool(vagast[num-1][2])==True:
        if bool(vagast[num-1][4])==False:
            bt[num]["bg"]="RED"
    if bool(vagast[num-1][2])==False:
        if vagast[num-1][4]==True:
            bt[num]["bg"]="YELLOW"
    if bool(vagast[num-1][2])==True:
        if vagast[num-1][4]==True:
            bt[num]["bg"]="RED"

# Simula o sinal dos sensores
def btc(btn):
    num=int(btn["text"])
```

```
    print(num)
    # Simula sinal alto e baixo do sensor
    #''
    #if vagast[num-1][2]==False:
    #    vagast[num-1][2]=True
    #else:
    #    vagast[num-1][2]=False
    print(vagast[num-1][2])
    print(vagast[num-1][4])
    atualiza(num)
    totaliza()

# Simula a entrada
def bec():
    nv=16
    if int(lbt["text"])<16:
        bloco=str(input("Escolha o bloco: "))
        menor=float(50)
        menorv=16
        for i in range(nv):
            if bool(vagast[i][2])==False:
                if bool(vagast[i][4])==False:
                    if vagast[i][3]==bloco:
                        if vagast[i][1]<menor:
                            menor=vagast[i][1]
                            menorv=vagast[i][0]-1
        print(menorv+1)
        if menorv<16:
            vagast[menorv][4]=True
            vagast[menorv][5]=datetime.now()
            print(vagast[menorv][5])
            print(vagast[menorv][2])
            print(vagast[menorv][4])
            lbn["width"]=6
            lbn["height"]=3
```

```
            lbn["text"]=str(menorv+1)
            lbn.place(x=490)
            tk.update_idletasks()
            delay.sleep(3)
            lbn["width"]=1
            lbn["height"]=1
            lbn["text"]=""
            lbn.place(x=590)
            tk.update_idletasks()
            atualiza(menorv+1)
            totaliza()
        else:
            print("Todas as vagas neste bloco estão ocupadas")
    else:
        print("Estacionamento lotado espere uma vaga ser liberada")

#Corrige vagas estacionadas errado
def nmark():
    m=float(50)
    n=8
    for i in range(8):
        if bool(vagast[i][2])==True:
            if bool(vagast[i][4])==False:
                delay.sleep(3)
                if bool(vagast[i][2])==True:
                    if bool(vagast[i][4])==False:
                        for j in range(8):
                            if bool(vagast[j][2])==False:
                                if bool(vagast[j][4])==True:
                                    if vagast[j][1]<m:
                                        m=vagast[j][1]
                                        n=vagast[j][0]-1
                        print(n+1)
                        if n<8:
                            vagast[i][4]=True
```

```
                    vagast[i][5]=vagast[n][5]
                    vagast[n][4]=False
                    vagast[n][5]=0
                    atualiza(n+1)
                    totaliza()

# Verifica sensores
def verifica():
    for i in range(16):
        GPIO.output(29,int(bin(i+32)[7]))
        GPIO.output(31,int(bin(i+32)[6]))
        GPIO.output(33,int(bin(i+32)[5]))
        GPIO.output(35,int(bin(i+32)[4]))
        vagast[i][2]=bool(GPIO.input(37))
        atualiza(i+1)
        #nmark()
        totaliza()

# Simula a saida
def bsc():
    s=int(input("Qual a vaga: "))
    print(s)
    vagast[s-1][4]=False
    temp=datetime.now()
    dif=temp-vagast[s-1][5]
    ar = open(str(temp.day)+"-"+str(temp.month),"r")
    txt=ar.read()
    ar.close()
    ar = open(str(temp.day)+"-"+str(temp.month),"w")
    ar.write(txt)
    ar.write(str(s)+"\n"+str(vagast[s-1][5].strftime('%d-%m-%Y
%H:%M:%S'))+"\n"+str(dif.total_seconds())+"\n"+str(temp.strftime('%d-%m-%Y
%H:%M:%S'))+"\n")
    ar.close()
    atualiza(s)
```

```
    totaliza()

bt=[]
for i in range(121): # 120 vagas
    bt.append(0)
tk=Tk()
W=1280
H=720
tk.geometry("1280x720")
tk.title("Testes")

# Label PCivil
lbc=Label(tk,text="Policia Civil",fg="WHITE",bg="BLUE",width=10,height=33)
lbc.place(x=1200,y=186)
# Label Superior
lbs=Label(tk,bg="BLACK",width=160,height=13)
lbs.place(x=0,y=0)
# Label Parede esquerda
lbe=Label(tk,bg="BLACK",width=16,height=13)
lbe.place(x=220,y=300)
# Label Parede meio
lbm=Label(tk,bg="BLACK",width=16,height=21)
lbm.place(x=590,y=185)
# Label Parece Direita
lbd=Label(tk,bg="BLACK",width=3,height=21)
lbd.place(x=988,y=185)
# Mostra total de vagas no estacionamento
lbt=Label(tk,text="0")
lbt.place(x=W-480,y=150)
# Botão simulando a escolha da entrada
be=Button(tk,text="Entrada")
be["command"]=bec
be.place(x=W-480,y=90)
# Botão simulando a saida do carro
bs=Button(tk,text="Saida")
```

```
bs["command"]=bsc
bs.place(x=W-480,y=120)

# Botões representando vagas
bt[1]=Button(tk,text="01",bg="GREEN",width=2,height=1)
bt[1]["command"]=partial(btc,bt[1])
bt[1].place(x=640+90,y=H/2-20)
bt[2]=Button(tk,text="02",bg="GREEN",width=2,height=1)
bt[2]["command"]=partial(btc,bt[2])
bt[2].place(x=(W*0.75)-20,y=H/2-20)
bt[3]=Button(tk,text="03",bg="GREEN",width=1,height=2)
bt[3]["command"]=partial(btc,bt[3])
bt[3].place(x=W*0.75+25,y=490)
bt[4]=Button(tk,text="04",bg="GREEN",width=2,height=1)
bt[4]["command"]=partial(btc,bt[4])
bt[4].place(x=(W*0.75)+60,y=H/2-20)
bt[5]=Button(tk,text="05",bg="GREEN",width=1,height=2)
bt[5]["command"]=partial(btc,bt[5])
bt[5].place(x=640-20,y=490)
bt[6]=Button(tk,text="06",bg="GREEN",width=1,height=2)
bt[6]["command"]=partial(btc,bt[6])
bt[6].place(x=320+690,y=H-50)
bt[7]=Button(tk,text="07",bg="GREEN",width=1,height=2)
bt[7]["command"]=partial(btc,bt[7])
bt[7].place(x=320+330,y=H-50)
bt[8]=Button(tk,text="08",bg="GREEN",width=1,height=2)
bt[8]["command"]=partial(btc,bt[8])
bt[8].place(x=W/4+50,y=H-50)
bt[9]=Button(tk,text="09",bg="GREEN",width=1,height=2)
bt[9]["command"]=partial(btc,bt[9])
bt[9].place(x=W/4-150,y=H-50)
bt[10]=Button(tk,text="10",bg="GREEN",width=2,height=1)
bt[10]["command"]=partial(btc,bt[10])
bt[10].place(x=W/2-100,y=H/2-20)
bt[11]=Button(tk,text="11",bg="GREEN",width=1,height=2)
```

```
bt[11]["command"]=partial(btc,bt[11])
bt[11].place(x=W/4+10,y=490)
bt[12]=Button(tk,text="12",bg="GREEN",width=2,height=1)
bt[12]["command"]=partial(btc,bt[12])
bt[12].place(x=W/4-150,y=340)
bt[13]=Button(tk,text="13",bg="GREEN",width=2,height=1)
bt[13]["command"]=partial(btc,bt[13])
bt[13].place(x=W/4+40,y=H/2-20)
bt[14]=Button(tk,text="14",bg="GREEN",width=1,height=2)
bt[14]["command"]=partial(btc,bt[14])
bt[14].place(x=W/4+10,y=190)
bt[15]=Button(tk,text="15",bg="GREEN",width=2,height=1)
bt[15]["command"]=partial(btc,bt[15])
bt[15].place(x=0,y=310)
bt[16]=Button(tk,text="16",bg="GREEN",width=2,height=1)
bt[16]["command"]=partial(btc,bt[16])
bt[16].place(x=0,y=460)

# Botões Auxiliares coluna 1
bt[17]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[17].place(x=640+90,y=190)
bt[18]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[18].place(x=640+90,y=220)
bt[19]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[19].place(x=640+90,y=250)
bt[20]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[20].place(x=640+90,y=280)
bt[21]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[21].place(x=640+90,y=310)
bt[22]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[22].place(x=640+90,y=370)
bt[23]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[23].place(x=640+90,y=400)
bt[24]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[24].place(x=640+90,y=430)
```

```
bt[25]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[25].place(x=640+90,y=460)
# Coluna 10
bt[26]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[26].place(x=640-100,y=220)
bt[27]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[27].place(x=640-100,y=250)
bt[28]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[28].place(x=640-100,y=280)
bt[29]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[29].place(x=640-100,y=310)
bt[30]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[30].place(x=640-100,y=370)
bt[31]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[31].place(x=640-100,y=400)
bt[32]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[32].place(x=640-100,y=430)
bt[33]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[33].place(x=640-100,y=460)
# Linha 5
bt[34]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[34].place(x=640-60,y=490)
bt[35]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[35].place(x=640-100,y=490)
bt[36]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[36].place(x=640+20,y=490)
bt[37]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[37].place(x=640+60,y=490)
bt[38]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[38].place(x=640+100,y=490)
# Coluna 2
bt[39]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[39].place(x=(W*0.75)-20,y=220)
bt[40]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[40].place(x=(W*0.75)-20,y=250)
```

```
bt[41]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[41].place(x=(W*0.75)-20,y=280)
bt[42]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[42].place(x=(W*0.75)-20,y=310)
bt[43]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[43].place(x=(W*0.75)-20,y=370)
bt[44]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[44].place(x=(W*0.75)-20,y=400)
bt[45]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[45].place(x=(W*0.75)-20,y=430)
bt[46]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[46].place(x=(W*0.75)-20,y=460)
# Coluna 4
bt[47]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[47].place(x=(W*0.75)+60,y=220)
bt[48]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[48].place(x=(W*0.75)+60,y=250)
bt[49]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[49].place(x=(W*0.75)+60,y=280)
bt[50]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[50].place(x=(W*0.75)+60,y=310)
bt[51]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[51].place(x=(W*0.75)+60,y=370)
bt[52]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[52].place(x=(W*0.75)+60,y=400)
bt[53]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[53].place(x=(W*0.75)+60,y=430)
bt[54]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[54].place(x=(W*0.75)+60,y=460)
# Linha 3
bt[55]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[55].place(x=W*0.75-15,y=490)
bt[56]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[56].place(x=W*0.75+65,y=490)
# Coluna 12
```

```
bt[57]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[57].place(x=W/4-150,y=310)
bt[58]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[58].place(x=W/4-150,y=370)
bt[59]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[59].place(x=W/4-150,y=400)
bt[60]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[60].place(x=W/4-150,y=430)
bt[61]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[61].place(x=W/4-150,y=460)
# Coluna 13
bt[62]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[62].place(x=W/4+40,y=310)
bt[63]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[63].place(x=W/4+40,y=370)
bt[64]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[64].place(x=W/4+40,y=400)
bt[65]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[65].place(x=W/4+40,y=430)
# Linha 11
bt[66]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[66].place(x=W/4+50,y=490)
bt[67]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[67].place(x=W/4-30,y=490)
bt[68]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[68].place(x=W/4-70,y=490)
bt[69]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[69].place(x=W/4-110,y=490)
bt[70]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[70].place(x=W/4-150,y=490)
# Linha 14
bt[71]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[71].place(x=W/4+50,y=190)
bt[72]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[72].place(x=W/4+90,y=190)
```

```
bt[73]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[73].place(x=W/4+130,y=190)
bt[74]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[74].place(x=W/4-30,y=190)
bt[75]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[75].place(x=W/4-70,y=190)
bt[76]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[76].place(x=W/4-110,y=190)
bt[77]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[77].place(x=W/4-150,y=190)
# Coluna 15
bt[78]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[78].place(x=0,y=220)
bt[79]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[79].place(x=0,y=250)
bt[80]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[80].place(x=0,y=280)
bt[81]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[81].place(x=0,y=340)
bt[82]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[82].place(x=0,y=370)
bt[83]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[83].place(x=0,y=400)
bt[84]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[84].place(x=0,y=430)
bt[85]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[85].place(x=0,y=490)
bt[86]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[86].place(x=0,y=520)
bt[87]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[87].place(x=0,y=550)
bt[88]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[88].place(x=0,y=580)
bt[89]=Button(tk,text="xx",bg="GREEN",width=2,height=1)
bt[89].place(x=0,y=610)
```

```
# Linha 9
bt[90]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[90].place(x=320-270,y=H-50)
bt[91]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[91].place(x=320-230,y=H-50)
bt[92]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[92].place(x=320-190,y=H-50)
bt[93]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[93].place(x=320-110,y=H-50)
bt[94]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[94].place(x=320-70,y=H-50)
bt[95]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[95].place(x=320-30,y=H-50)
bt[96]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[96].place(x=320+10,y=H-50)
bt[97]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[97].place(x=320+90,y=H-50)
bt[98]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[98].place(x=320+130,y=H-50)
bt[99]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[99].place(x=320+170,y=H-50)
bt[100]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[100].place(x=320+210,y=H-50)
bt[101]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[101].place(x=320+250,y=H-50)
bt[102]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[102].place(x=320+290,y=H-50)
#bt[103]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
#bt[103].place(x=320+330,y=H-50)
bt[104]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[104].place(x=320+370,y=H-50)
bt[105]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[105].place(x=320+410,y=H-50)
bt[106]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[106].place(x=320+450,y=H-50)
```

```
bt[107]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[107].place(x=320+490,y=H-50)
bt[108]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[108].place(x=320+530,y=H-50)
bt[109]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[109].place(x=320+570,y=H-50)
bt[110]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[110].place(x=320+610,y=H-50)
bt[111]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[111].place(x=320+650,y=H-50)
bt[112]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[112].place(x=320+730,y=H-50)
bt[113]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[113].place(x=320+770,y=H-50)
bt[114]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[114].place(x=320+810,y=H-50)
bt[115]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[115].place(x=320+850,y=H-50)
bt[116]=Button(tk,text="xx",bg="GREEN",width=1,height=2)
bt[116].place(x=320+890,y=H-50)
# Label Numero na Tela
lbn=Label(tk,bg="BLACK",text="",fg="WHITE",font="none 80",width=0,height=0)
lbn.place(x=590,y=185)
while(1):
    verifica()
    tk.update_idletasks()
    tk.update()
```

APENDICE C

Código que trata os dados gerados pelo sistema para serem inseridos no código que gera o gráfico de análise estatística.

```
from functools import partial
import time as delay
from datetime import datetime,timedelta

data=str(input("Escolha a data do arquivo (DIA-MES)"))


# Permanencia x Numero de carros
arq=open(data)
x=1
t=0
pn=[]
for i in arq.readlines():
    if x==3:
        j=int(i[:len(i)-8])
        j=int(j/60)
        pn.append(j)
    x=x+1
    if x==5:
        x=1
        t=t+1
print(t)
pn.sort()
print(pn)

arq.close()
perm=open("PxN_"+data,"w")
for i in range(t):
    perm.write(",["+str(i+1)+","+str(pn[i])+"]")
perm.close()

# Quantidade x Hora do dia
arq=open(data)
x=1
qt=[]
ent=[]
sai=[]
for i in arq.readlines():
    if x==2:
        j=datetime.strptime(i,'%d-%m-%Y %H:%M:%S\n')
        je=j.hour+j.minute/60
        qt.append(je)
        ent.append(je)
    if x==4:
        j=datetime.strptime(i,'%d-%m-%Y %H:%M:%S\n')
        js=j.hour*60+j.minute
        qt.append(js)
        sai.append(js)
```

```
    x=x+1
    if x==5:
        x=1
qt.sort()
ent.sort()
sai.sort()
print(qt)
print(ent)
print(sai)

arq.close()

n=0
qty=[]
for i in range(len(qt)):
    for o in range(t):
        if ent[o]<=qt[i]:
            #print(qt[i])
            #print(ent[o])
            n=n+1
        if sai[o]<=qt[i]:
            n=n-1
    qty.append(n)
    n=0
print(qty)

arq=open("QxH_"+data,"w")
for i in range(len(qt)):
    arq.write(",["+str(qt[i])+","+str(qty[i])+"]")
arq.close()

# Quantidade de carros x Cada vaga (Frequencia de uso de determinada vaga)
arq=open(data)
x=1
vagas=[]
for o in range(16):
    vagas.append(0)
for i in arq.readlines():
    if x==1:
        vagas[int(i)-1]=vagas[int(i)-1]+1
    x=x+1
    if x==5:
        x=1
arq.close()

arq=open("QxV_"+data,"w")
for i in range(16):
    arq.write(",["+str(i+1)+","+str(vagas[i])+"]")
arq.close()
print(vagas)
```

ANEXO A

Código em JavaScript para gerar gráficos de linha.

```
<!DOCTYPE html>
<head>
        <title>Gráfico</title>
        <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
        <div id="chart_div"></div>
<script>
                google.charts.load('current', {packages: ['corechart', 'line']});
                google.charts.setOnLoadCallback(drawBackgroundColor);
function drawBackgroundColor() {
    var data = new google.visualization.DataTable();
    data.addColumn('number', 'X');
    data.addColumn('number', 'Carros');
    data.addRows([
      [616,1],[736,0],[776,1],[846,0],[900,1],[996,0],[1036,1],[1186,0]
    ]);
    var options = {
      hAxis: {
        title: 'Tempo(Minutos)'
      },
      vAxis: {
        title: 'Quantidade'
      },
      backgroundColor: '#f1f8e9'
    };
    var chart = new
google.visualization.LineChart(document.getElementById('chart_div'));
    chart.draw(data, options);
  }
        </script>
```