RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
VOL. 3 NRO. 2 2022

MODELO DE PLACA DE PARE LUMINOSO PARA PREVENÇÃO DE SITUAÇÕES DE RISCO EM CRUZAMENTOS

Pitter Meira de Oliveira Camargo
Graduando em Ciência da Computação
pittermeira@hotmail.com

Yasmin Oliveira Xavier
Graduando em Ciência da Computação
yass.o.xavier@gmail.com

Carlos Eduardo de França Roland
Mestre em Desenvolvimento Regional – Uni-FACEF
roland@facef.br

Resumo

Esse trabalho acadêmico tem o objetivo de propor uma alternativa, simples e barata, baseada em sistema microcontrolado, para melhorar a segurança nas vias urbanas. Nos cruzamentos entre vias sem sinalização semafórica, frequentemente os condutores se deparam com a dificuldade de visualizar o movimento da via a ser cruzada devido, por exemplo, a veículos estacionados irregularmente. Diante deste cenário foi realizada pesquisa bibliográfica sobre os componentes de hardware e software existentes para propor um projeto de automação que pudesse operar em placas sinalizadoras de Pare para complementar a sinalização de trânsito. A hipótese a ser testada se refere a um dispositivo com uma câmera que identifica veículos em um cruzamento urbano e gera sinais de controle para leds instalados na placa reforçando a sinalização de trânsito urbano. As tecnologias consideradas para implementar a solução foram assim estabelecidas: coleta de imagens com microcâmera e processamento com *script* Python para armazenamento dos resultados em nuvem baseado em serviços MQTT, e acesso às publicações em nuvem por dispositivo microcontrolado NodeMCU com ESP8266 para controle da operação dos leds na placa. O sistema deveria ser mantido em operação por geração fotoelétrica durante o dia e baterias recarregáveis à noite. Para implementação e testes da solução foi desenvolvido um protótipo funcional como prova de conceito. A proposta de hardware e software pode ser testada com resultado satisfatório melhorando a visão da solução para ser, em projeto futuro, transformado em protótipo funcional para testes em campo.

Palavras-chave: Arduino. MQTT. NodeMCU ESP8266. Processamento de imagens. Python. Segurança no trânsito. Sistemas microcontrolados.

Abstract

This academic work aims to propose an alternative, simple and cheap, based on a microcontrolled system, to improve safety on urban roads. At intersections between roads without traffic lights, drivers are often faced with the difficulty of visualizing the movement of the road to be crossed due, for example, to illegally parked vehicles. In view of this scenario, a bibliographical research was carried out on the existing hardware and software components to propose an automation project that could operate on stop signs to complement the traffic signaling. The hypothesis to be tested

refers to a device with a camera that identifies vehicles at an urban intersection and generates control signals for LEDs installed on the license plate, reinforcing urban traffic signs. The technologies considered to implement the solution were as follows: image collection with a microcamera and processing with a Python script to store the results in the cloud based on MQTT services, and access to publications in the cloud via a NodeMCU microcontrolled device with ESP8266 to control the operation of the LEDs on the board. The system was to be kept in operation by photoelectric generation during the day and rechargeable batteries at night. For implementation and testing of the solution, a functional prototype was developed as a proof of concept. The hardware and software proposal can be tested with a satisfactory result, improving the vision of the solution to be, in a future project, transformed into a functional prototype for field tests.

Keywords: *Arduino. Image processing. Microcontrolled systems. MQTT. NodeMCU ESP8266. Python. Traffic Safety.*

1 Introdução

Ao fazer travessias de ruas preferenciais, muitas vezes o motorista tem a visão obstruída por outros veículos estacionados do mesmo lado da via que trafega, fazendo com que ele precise ultrapassar a linha de pare, que está a uma distância segura da via transversal, correndo o risco de colidir com outros veículos. Para evitar esse tipo de perigo, uma solução alternativa pode ser proposta através de um sistema luminoso formado por uma placa de sistema embarcado, que se conecta à internet via redes móveis, led e placa de energia solar para seu funcionamento ininterrupto, e que seria adicionado à placa de pare avisando o motorista se ele pode ou não fazer a travessia. Na grande maioria das vezes essas placas são instaladas de forma a garantir visibilidade a uma distância segura na sua própria via.

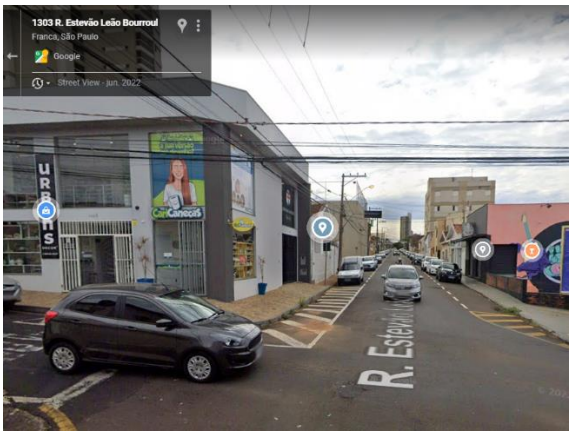
2 Revisão teórica

A questão norteadora deste projeto de Trabalho de Conclusão de Curso foi definida a partir da percepção de um problema de trânsito comum hoje em dia que, no entanto, passa despercebido para a maioria dos motoristas, colocando-os em riscos desnecessários e possíveis chances de acidentes, como mostra a Figura 1. A proposta do projeto envolve um circuito eletrônico digital microcontrolado, de fácil aplicação, podendo ser instalado em placas de Pare em cruzamentos onde seja necessário prevenir eventuais acidentes.

2.1 Engenharia de Software

De acordo com Ian Sommerville (2018), a Engenharia de Software se caracteriza por ser a área da tecnologia que dá suporte ao desenvolvimento de software, principalmente em áreas mais específicas de análise do problema do projeto, assim como também na evolução dele.

Figura 1 - Código de programação do NodeMCU - parte 2



Fonte: Google Street View

2.2 Qualidade de Software (QS)

Para que um software seja entregue com a eficácia solicitada é necessário que o projeto seja acompanhado de perto, de modo que, caso exista algum problema no decorrer do desenvolvimento, ele possa ser corrigido o mais cedo possível. Esse processo é chamado de Gerenciamento de Qualidade de Software.

Além da eficiência do produto, o gerenciamento de qualidade é responsável por fazer com que o projeto seja entregue dentro do prazo e custo previamente definidos. De modo geral, a qualidade prevê que o software seja confiável, funcione de modo correto e atenda às necessidades do cliente, segundo Sommerville (2018).

QS é uma parte importante do processo de desenvolvimento de software e é abrangente de acordo com o projeto em execução. Existem requisitos que são contemplados em um software, porém divergem em resultados de uma implementação para outra. Desse modo a equipe responsável pela QS define quais são os parâmetros mais eficazes para os testes no projeto para analisar os resultados de acordo com o esperado.

2.3 Sistemas Embarcados

Diferente dos sistemas para computadores, os sistemas embarcados são um somatório de hardware e software com o objetivo de realizar tarefas específicas baseadas em requisitos e alocado dentro de um aparelho com automação tecnológica. Sua composição inclui microcontroladores, a peça-chave para a construção de um sistema embarcado, pois eles são os dispositivos que comandam os processos do equipamento.

Sua primeira aparição foi no projeto Apollo da NASA, o *Apollo Guidance Computer*, um computador de bordo com atualizações em tempo real que guiava os voos da nave espacial. Entretanto o sistema embarcado que foi reproduzido mais de uma vez, foi um computador de guia de míssil norte-americano na década de 60 (UFPR, 2015).

Desde então, a evolução tecnológica trouxe melhorias e as tarefas manuais e analógicas puderam ser automatizadas, pois o acesso aos componentes para a construção de um sistema ficou mais fácil. Hoje, sistemas embarcados são usados em vários equipamentos do dia a dia, facilitando a vida das pessoas sem que elas percebam.

2.3.1 Microcontroladores

Um microcontrolador é um computador de programa armazenado em um único circuito integrado (do inglês *System On a Chip* - SOC) que nele há um processador e espaço de memórias volátil e não volátil, para armazenamento de instruções e dados. Os microcontroladores contêm portas de comunicação com o meio externo que podem ser configuradas para as funções de entrada ou saída, para receber dados de sensores e enviar comandos para atuadores.

Ele foi criado nos anos 70 pela Intel Corporation, empresa atuante no ramo até os dias atuais como a maior fornecedora de microprocessadores (OFICINA DA NET, 2020) e possui clientes como Samsung, Acer, Lenovo, entre outros. Sua importância vem da necessidade dos microcontroladores para tarefas específicas em sistemas compactos.

A vantagem que os microcontroladores têm sobre os outros sistemas é que todos os componentes necessários para seu funcionamento estão juntos em um único componente, fazendo assim com que ele seja mais fácil de instalar nos sistemas embarcados.

2.3.2 Plataforma Arduino

Arduino é uma plataforma de prototipagem versátil e ampla para usar em projetos de automação e controle. As placas Arduino têm o objetivo de facilitar o acesso à prototipagem de produtos digitais, tendo sido desenvolvida uma plataforma de uso simplificado para a programação do sistema e a eletrônica foi implementada em placas de baixo custo e flexíveis para uma grande gama de projetos.

As placas Arduino foram criadas em 2005 por cinco professores pesquisadores do programa de pós graduação IVREA *(Interaction Design Institute* Ivrea): David Mellis, David Cuartielles, Gianluca Martino, Massimo Banzi e Tom Igoe. A primeira placa criada tinha um microcontrolador ATmega128, podia ser programada por uma IDE usando a linguagem C++ em um microcomputador e com um cabo USB recebia o programa compilado (*sketch*) para ser armazenado em uma memória não volátil do microcontrolador. O projeto Arduino adotou o conceito de *Open Hardware*, onde pode-se criar, editar, montar, personalizar, melhorar, e modificar o sistema todo.

2.3.3 ESP8266

De acordo com Oliveira (2021), ESP8266 é um modelo de microcontrolador feito pela empresa Espressif, da China, com especificações de fabricação de 50 kb de memória disponível para dados e 4 MB disponíveis para o programa. Ele funciona com um processador de 80MHz a 160MHz, tensão de 3,3 V e corrente de 20μA em modo ocioso, e varia de 50mA a 170mA quando em operação, desse modo, com uma bateria recarregável ele pode funcionar até 20 horas ou mais.

Ele conta com 17 portas de entrada ou saída, podendo ser configuradas para operar em modo digital, e uma em modo analógico. Possui interfaces de

comunicação síncronas e assíncronas, além da interface *Wifi,* possibilitando que a atualização do componente seja feita através da rede *Wifi*, ao invés de apenas pelo cabo USB, facilitando a operação do sistema.

As aplicações embarcadas podem ser desenvolvidas com o ESP8266 integrado à programação pela IDE Arduino, desse modo facilitando o desenvolvimento do projeto lógico de automação.

2.3.4 NodeMCU

Criado em 2014, NodeMCU é uma placa microcontrolada concorrente da Arduíno e muito usada para desenvolvimento de projetos IoT. Seu diferencial é ser uma plataforma *open source* que não precisa de um componente externo para sua aplicação, por embutir no hardware o ESP8266 para comunicação *wifi*.

2.3.5 Redes *Wifi* e Móveis

Wifi, abreviação para *Wireless Fidelity*, foi criado por volta do ano de 1997 e tem como objetivo disponibilizar a internet por meio de frequências de rádio via protocolo Ethernet, sem a necessidade da conexão por cabos. No início, a transferência de dados era de, no máximo, 11 Mb/s que foi evoluindo com o tempo, melhorando segurança, eficiência e velocidade de modo que, em 2019, ela era de 9,6 Gb/s, sendo 87 vezes mais potente do que quando foi criada (GARRET, 2021).

Para que se faça a conexão à rede é necessário que tenha um intermediador chamado roteador, que distribui as ondas *Wifi* dentro da área de alcance e faz com que os aparelhos compatíveis se conectem à internet. Essa conexão é feita por um *microchip* existente dentro dos processadores e aparelhos, alguns com frequência de 2.4 GHz e outros com uma frequência de 5 GHz (TECHTUDO, 2015).

2.3.6 Linguagem Python

Aproximadamente nos anos 90 um matemático holandês criou uma linguagem de programação com o objetivo de melhorar a criação e leitura dos códigos por programadores e trabalhadores da área. Desse modo surgiu a linguagem Python.

Por ser uma linguagem criada para descomplicar o trabalho de programação, ela é multiplataforma, dinâmica e orientada a objetos, assim se tornando uma linguagem de alto nível. Um de seus diferenciais também é a quantidade de bibliotecas disponíveis, possibilitando a criação de sistemas simples ou complexos, além de ser usada também para análise de dados e programação de sistemas embarcados (KRIEGER, 2022)

2.4 Internet das Coisas

Magrani (2018) define Internet das Coisas (IoT do termo em inglês *Internet of Things*) como uma conexão entre objetos e a própria internet através de aparelhos captadores do mundo ao redor, os sensores, e que transmitem dados para centrais de armazenamento para processamento e tomadas de decisão, enviando comandos para atuadores. De acordo com o autor a “sigla refere-se a um mundo onde objetos e pessoas, assim como dados e ambientes virtuais, interagem uns com os outros no espaço e no tempo” (MAGRANI, 2018, p. 44).

O termo foi criado por Kevin Ashton em 1999, pouco depois de Bill Joy ter pensado em possibilidades *device-to-device* (D2D, dispositivo para dispositivo em tradução livre). Ashton (*apud* Magrani, 2018) ainda escreveu um artigo sobre como as pessoas, na correria do dia a dia, precisam se conectar à internet através de novos meios. Com IoT essa possibilidade se torna cada vez mais viável.

Atualmente ela está presente em nossas vidas em locais que muitas vezes passam despercebidos, como por exemplo em eletrodomésticos, que, com comandos simples, executam suas tarefas. Assim, quando Ashton desenvolvia seu artigo, ele pensava que “essa revolução será maior do que o próprio desenvolvimento do mundo online” (Magrani, 2018, p. 45).

2.4.1 Cidades Inteligentes

Farias *et al.* (2011) apresentam como vem crescendo o número de pessoas nas cidades e que por consequência os problemas aumentam. Para ele, o conceito de Cidade Inteligente baseia-se em um ambiente com tecnologia inteligente para otimização dos recursos básicos e ambiente capaz de fornecer a todos uma boa qualidade de vida.

Quando se fala sobre tecnologia inteligente para otimização, estamos falando de aparelhos eletrônicos com propósito de coleta e análise de dados. No caso das cidades tem-se exemplos de sensores de poluição do ar ou água, umidade do ar, temperatura, luminosidade, dentre outros sensores voltados para monitoramento ambiental.

Pelo adensamento populacional nas cidades também surge a questão do aumento de tráfego de veículos nas vias urbanas, gerando a necessidade de monitorar a malha viária. Bucker e Souza (2018), discutem a questão da qualidade da mobilidade urbana em cidades inteligentes.

No estudo apresentam que numa pesquisa feita em 2017, havia uma diferença significativa entre a taxa de mortalidade no trânsito entre os países de baixa e de alta renda devido ao mau planejamento urbano. No ano de 2021 foram registradas mais de 11 mil mortes no trânsito e mais de 600 mil ocorrências. Foram aproximadamente 32 mortes por dia e 72 acidentes por hora.

Em cidades inteligentes como Amsterdã, Berlim e Copenhague, as apostas de melhorias estão na diminuição de veículos particulares e melhorias no transporte público ou não automotivo. Muitas delas implementaram projetos para que os cidadãos usem bicicletas alugáveis como meio de transporte, de modo a diminuir a quantidade de veículos e melhorar a saúde dos habitantes.

2.5 Gestão de energia em IoT

Todo e qualquer dispositivo eletrônico necessita de uma fonte de energia para seu funcionamento recarregável ou não. Em IoT, geralmente o funcionamento é contínuo, necessitando, assim, de uma fonte que seja duradoura e econômica.

Para esses sistemas, é preciso uma fonte primária (em tensão e corrente alternadas), que é a energia distribuída por empresas do ramo de eletricidade, com tensões de 127 V ou de 220 V, que precisam ser convertidas em corrente contínua para o adequado funcionamento de circuitos digitais. Já em sistemas IoT remotos, é

recomendado fontes de energia alternativas como energia solar e eólica, dependendo do projeto.

2.5.1 Energia Solar Fotovoltaica

Como o próprio nome sugere, energia solar é a energia gerada pela luminosidade e temperatura do sol sobre a Terra, que é convertida em energia elétrica e energia térmica, dependendo de como ela é captada. Para a conversão em energia elétrica, a solar é captada por painéis fotovoltaicos, usinas heliotérmicas e painéis de aquecimento (SOUZA, 2022).

A energia solar fotovoltaica é a energia que recebe luminosidade e a converte para energia elétrica, utilizando painéis fotovoltaicos. Nos últimos anos, ela vem sendo uma fonte de energia alternativa para as residências, pois sua implementação faz com que o consumo de energia fornecida pelas companhias elétricas seja menor, consequentemente baixando o valor da conta no final do mês (SOUZA, 2022).

Os painéis solares são feitos de placas de metais semicondutores, geralmente sendo usado o silício, e seu funcionamento se dá por meio do chamado Efeito Fotovoltaico. Quando um fóton emitido pela luz do sol entra em contato com esses metais, ele cria um deslocamento dos elétrons livres dos elementos, e esse deslocamento é o responsável por gerar a corrente elétrica que transforma a energia solar em elétrica (SOUZA, 2022).

3 Desenvolvimento

Para o desenvolvimento do protótipo funcional, foi criada uma prova de conceito, com o objetivo de mostrar e testar a hipótese de solução. Para isso, foi necessário uma *protoboard*, uma placa NodeMCU - ESP8266, cabos elétricos, um led e um microcomputador para o desenvolvimento da lógica de programação do sistema nas linguagens Wiring (para o microcontrolador, pela IDE Arduino) e Python para processamento de imagem e envio dos dados coletados para um repositório em nuvem.

3.1 Programação do Arduíno

A programação do NodeMCU foi feita em linguagem C++, no programa Visual Studio Code (VS Code), como descrito a seguir.

A Figura 2 retrata o código para a importação das bibliotecas. A biblioteca ESP82GGWifi serve para que o sistema embarcado se conecte à uma rede *Wifi*. A biblioteca PubSubClient é necessária para que sejam enviadas e recebidas mensagens para o armazenamento em nuvem pelo protocolo MQTT.

Nas definições, o TOPICO_SUBSCRIBE é voltado para ler as mensagens enviadas ao *broker* pelo protocolo MQTT, enquanto o TOPICO_PUBLISH é usado para enviar as mensagens para o *broker*, que é o serviço em nuvem que hospeda os dados comunicados via MQTT. Por fim, o ID_MQTT é o que define a sessão de MQTT.

Figura 2 - Código de programação do NodeMCU - parte 1

```
#include <ESP8266WiFi.h> // Importa a Biblioteca ESP8266WiFi
#include <PubSubClient.h> // Importa a Biblioteca PubSubClient

//defines:
//defines de id mqtt e tópicos para publicação e subscribe
#define TOPICO_SUBSCRIBE "iottccsub"     //tópico MQTT de escuta
#define TOPICO_PUBLISH   "iottccpub"    //tópico MQTT de envio de informações para Broker
                                        //IMPORTANTE: recomendamos fortemente alterar os nomes
                                        //            desses tópicos. Caso contrário, há grandes
                                        //            chances de você controlar e monitorar o NodeMCU
                                        //            de outra pessoa.
#define ID_MQTT  ""     //id mqtt (para identificação de sessão)
                        //IMPORTANTE: este deve ser único no broker (ou seja,
                        //            se um client MQTT tentar entrar com o mesmo
                        //            id de outro já conectado ao broker, o broker
                        //            irá fechar a conexão de um deles).
```

Fonte: os autores

Na Figura 3, são mostradas as definições dos pinos usados para operação do NodeMCU.

Figura 3 - Código de programação do NodeMCU - parte 2

```
//defines - mapeamento de pinos do NodeMCU
#define D0    16
#define D1    5
#define D2    4
#define D3    0
#define D4    2
#define D5    14
#define D6    12
#define D7    13
#define D8    15
#define D9    3
#define D10   1
```

Fonte: os autores

A Figura 4 mostra os códigos necessários para que o sistema se conecte ao WIFI e ao *brokerf* que foi instalado na máquina usada para teste e conectada ao MQTT *explorer*, na internet para verificação.

Figura 4 - Código de programação do NodeMCU - parte 3

```
// WIFI
const char* SSID = "Moto G (5S) Plus 1797"; // SSID / nome da rede WI-FI que deseja se conectar
const char* PASSWORD = "b1b1297a9ec5"; // Senha da rede WI-FI que deseja se conectar

// MQTT
const char* BROKER_MQTT ="192.168.43.227"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQ TT
```

Fonte: os autores

Na Figura 5, é criada a conexão com o MQTT, sob nome de “espClient”, e a variável “EstadoSaida” recebe o último retorno que o MQTT enviou.

Figura 5 - Código de programação do NodeMCU - parte 4

```
//Variáveis e objetos globais
WiFiClient espClient; // Cria o objeto espClient6
PubSubClient MQTT(espClient); // Instancia o Cliente MQTT passando o objeto espClient
char EstadoSaida = '0';  //variável que armazena o estado atual da saída
```

Fonte: os autores

Na Figura 6 são mostradas as chamadas às funções, que são definidas em seguida. A função “initSerial”, responsável por monitorar a taxa de transferência em bits por segundo no terminal e a função “init*Wifi*”, responsável por fazer a conexão *Wifi* com o sistema.

A função “initMQTT” é configurada, indicando qual *broker* e porta serão conectados, além de chamar a função “mqtt_callback”.

A função “mqtt_callback” é chamada quando recebe alguma informação no *broker*. Essa função é responsável por enviar ao Led o dado de que ele precisa ser acionado, isso apenas quando necessário.

A função “reconectMQTT” é a responsável por conectar o *broker* caso a conexão ainda não tenha sido feita ou tenha caído. Do mesmo modo, a função “reconectWifi” realiza a conexão com a rede Wifi caso essa não tenha sido realizada, ou a conexão tenha caído. A função “VeirifcaConexaoWifiEMQTT” é responsável por vigiar as duas conexões, tanto Wifi quanto do Broker.

A Figura 7 mostra o código necessário para que o *broker* receba o estado de saída, 0 ou 1, do sistema, e a Figura 8 mostra a função inicial do processo que define que o *output* inicial é 1 e o Led, operado em estado lógico contrário, inicia-se apagado.

Figura 6 - Código de programação do NodeMCU - parte 5

```
//Prototypes
void initSerial();
void initWiFi();
void initMQTT();
void reconectWiFi();
void mqtt_callback(char* topic, byte* payload, unsigned int length);
void VerificaConexoesWiFIEMQTT(void);
void InitOutput(void);

/*
 *  Implementações das funções
 */
void setup()
{
    //inicializações:
    InitOutput();
    initSerial();
    initWiFi();
    initMQTT();
}
```

Fonte: os autores

Figura 7 - Código de programação do NodeMCU - parte 6

```
//Função: envia ao Broker o estado atual do output
//Parâmetros: nenhum
//Retorno: nenhum
void EnviaEstadoOutputMQTT(void)
{
    if (EstadoSaida == '0')
      MQTT.publish(TOPICO_PUBLISH, "D");

    if (EstadoSaida == '1')
      MQTT.publish(TOPICO_PUBLISH, "L");

    Serial.println("- Estado da saida D0 enviado ao broker!");
    delay(1000);
}
```

Fonte: os autores

Figura 8 - Código de programação do NodeMCU - parte 7

```
//Função: inicializa o output em nível lógico baixo
//Parâmetros: nenhum
//Retorno: nenhum
void InitOutput(void)
{
    //IMPORTANTE: o Led já contido na placa é acionado com lógica invertida (ou seja,
    //enviar HIGH para o output faz o Led apagar / enviar LOW faz o Led acender)
    pinMode(D0, OUTPUT);
    digitalWrite(D0, HIGH);
}
```

Fonte: os autores

A Figura 9 mostra o código principal do sistema, aquele que executa em laço infinito para automação do processo.

Figura 9 - Código de programação do NodeMCU - parte 8

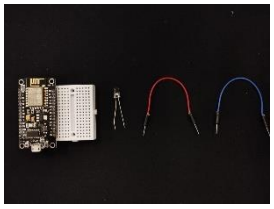
```
//programa principal
void loop()
{
    //garante funcionamento das conexões WiFi e ao broker MQTT
    VerificaConexoesWiFIEMQTT();

    //envia o status de todos os outputs para o Broker no protocolo esperado
    EnviaEstadoOutputMQTT();

    //keep-alive da comunicação com broker MQTT
    MQTT.loop();
}
```

Fonte: os autores

3.1.1 Montagem do hardware

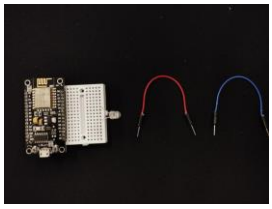
A montagem se iniciou conectando a placa NodeMCU a um dos lados do *protoboard* (Figura 10). Do outro lado foi conectado o anodo e catodo, considerados os lados positivo e negativo, em espaços diferentes (Figura 11). Em seguida, foi conectado um cabo no D0 e no anodo (Figura 12), e outro cabo no GND e no catodo para fechamento do circuito (Figura 13).

Figura 10 - Passo um: montagem da prova de conceito



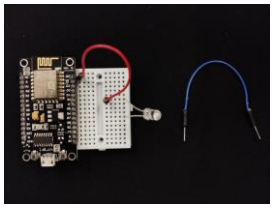
Fonte: os autores

Figura 11 - Passo dois: montagem da prova de conceito



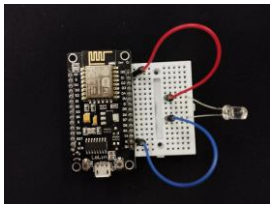
Fonte: os autores

Figura 12 - Passo três: montagem da prova de conceito



Fonte: os autores

Figura 13 - Passo quatro: montagem da prova de conceito



Fonte: os autores

3.2 Programação da Câmera e análise de imagens

A câmera foi programada em linguagem Python para identificar os carros, enviar um sinal ao *broker* e, aí sim, ligar o led, sinalizando que o motorista da via secundária não poderia ultrapassar a pista preferencial.

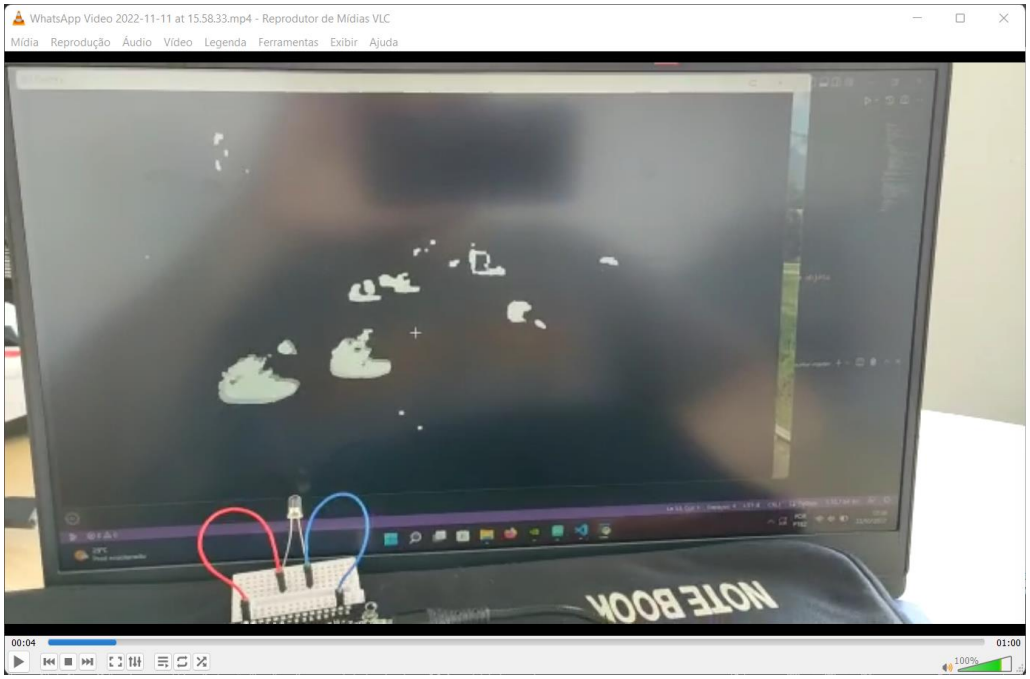
Na Figura 14 é mostrado o código detalhado do processo utilizado pela câmera para a identificação dos veículos. Seu funcionamento tem como princípio a diferenciação de objetos na tela e, para isso, é necessário que a câmera separe em frames as imagens recebidas e as transforme em preto e branco (Figura 15). Feito isso, ela cria uma matriz para preencher os buracos criados na imagem, onde ficavam as tais imperfeições.

Figura 14 - Código de programação da câmera - parte 1

```
while True:
    ret, frame1 = cap.read()  # Pega cada frame do vídeo
    tempo = float(1 / delay)
    sleep(tempo)  # Dá um delay entre cada processamento
    grey = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)  # Pega o frame e transforma para preto e branco
    blur = cv2.GaussianBlur(grey, (3, 3), 5)  # Faz um blur para tentar remover as imperfeições da imagem
    img_sub = subtracao.apply(blur)  # Faz a subtração da imagem aplicada no blur
    dilat = cv2.dilate(img_sub, np.ones((5, 5)))  # "Engrossa" o que sobrou da subtração
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (
        5, 5))  # Cria uma matriz 5x5, em que o formato da matriz entre 0 e 1 forma uma elipse dentro
    dilatada = cv2.morphologyEx(dilat, cv2.MORPH_CLOSE, kernel)  # Tenta preencher todos os "buracos" da imagem
    dilatada = cv2.morphologyEx(dilatada, cv2.MORPH_CLOSE, kernel)

    contorno, img = cv2.findContours(dilatada, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    cv2.line(frame1, (25, pos_linha), (1200, pos_linha), (255, 127, 0), 3)
    for (i, c) in enumerate(contorno):
        (x, y, w, h) = cv2.boundingRect(c)
        validar_contorno = (w >= largura_min) and (h >= altura_min)
        if not validar_contorno:
            continue

        cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)
        centro = pega_centro(x, y, w, h)
        detec.append(centro)
        cv2.circle(frame1, centro, 4, (0, 0, 255), -1)

    set_info(detec)
    show_info(frame1, dilatada)

    if cv2.waitKey(1) == 27:
        break
```

Fonte: os autores

Figura 15 – Imagem transformada em preto e branco



Fonte: os autores

A função mostrada na Figura 16 é a principal para a detecção do veículo, pois ela é a responsável por identificar, localizar e enviar os dados de que o objeto em

foco foi registrado. Para que isso ocorra, é traçada uma linha de referência na imagem, e essa linha faz a identificação do objeto (Figura 17).

Por fim, a figura 18 mostra o funcionamento do sistema. Recebidas as informações de que o veículo está no frame, a primeira coisa que essa função faz é enviar ao broker a informação, com atraso de cerca de 1 segundo (Figuras 19 e 20). Assim que é recebido, ele inicia o processo em que o estado de saída que está desligado é ligado, ficando por 4 segundos aceso, e em seguida desligado, após esse processo, ele finaliza o processo com uma contagem de carros.

Figura 16 - Código de programação da câmera - parte 2

```
def pega_centro(x, y, largura, altura):
    """
    :param x: x do objeto
    :param y: y do objeto
    :param largura: largura do objeto
    :param altura: altura do objeto
    :return: tupla que contém as coordenadas do centro de um objeto
    """
    x1 = largura // 2
    y1 = altura // 2
    cx = x + x1
    cy = y + y1
    return cx, cy
```

Fonte: os autores

Figura 17 – Linha traçada na imagem

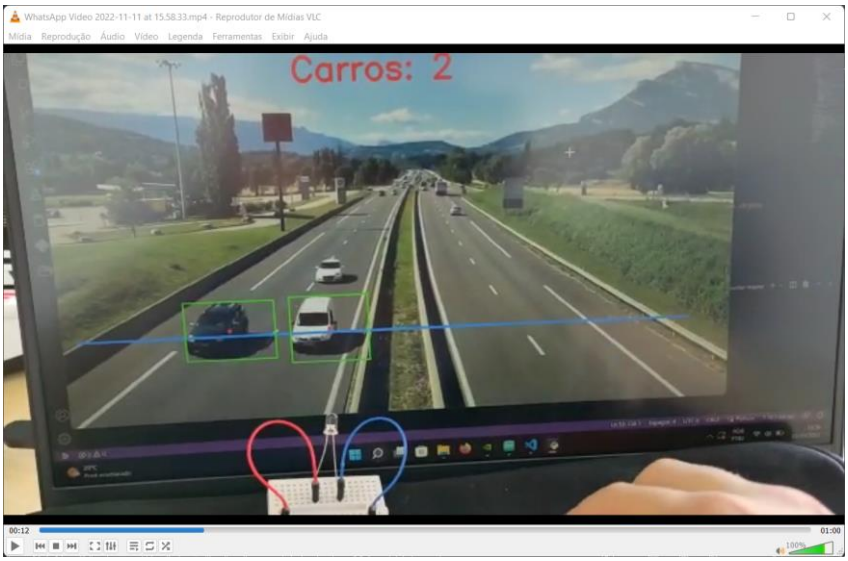
Fonte: os autores

Figura 18 - Código de programação da câmera - parte 3

```
def set_info(detec):
    global carros
    for (x, y) in detec:
        if (pos_linha + offset) > y > (pos_linha - offset):
            carros += 1
            broker="192.168.43.227"
            broker="192.168.43.227"
            #define callback
            def on_message(client, userdata, message):
                time.sleep(1)
                print("received message =",str(message.payload.decode("utf-8")))

            client= paho.Client("client-001") #create client object client1.on_publish = on_pul
            ######Bind function to callback
            client.on_message=on_message
            #####
            print("connecting to broker ",broker)
            client.connect(broker)#connect
            client.loop_start() #start loop to process received messages
            client.publish("iottccsub","L")#publish
            client.publish("iottccsub","D")#publish
            time.sleep(4)
            client.publish("iottccsub","L")#publish
            client.disconnect() #disconnect
            client.loop_stop() #stop loop
            cv2.line(frame1, (25, pos_linha), (1200, pos_linha), (0, 127, 255), 3)
            detec.remove((x, y))
            print("Carros detectados até o momento: " + str(carros))
```

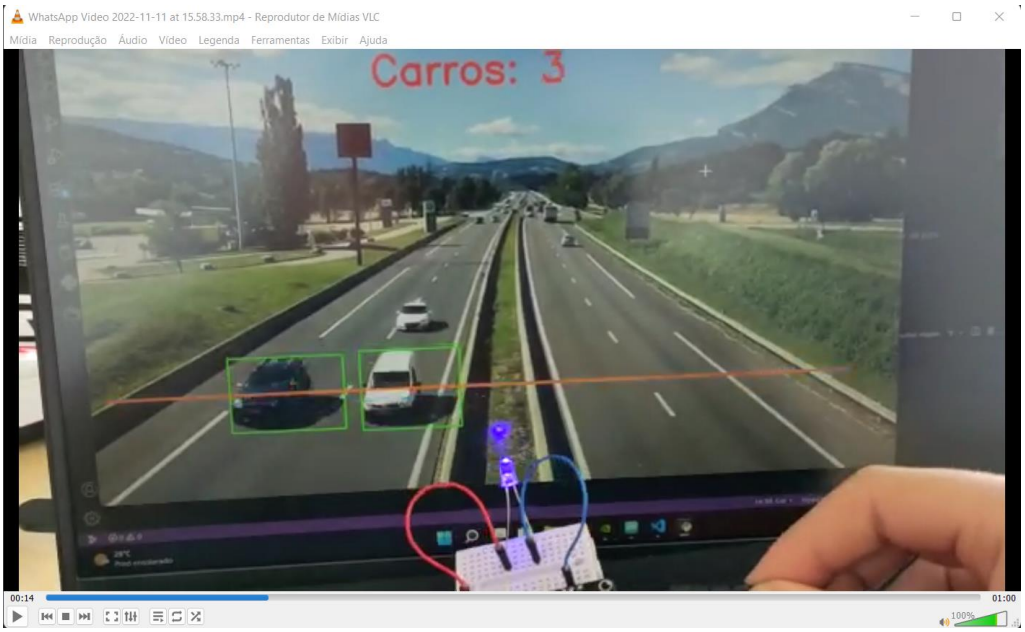
Fonte: os autores

Figura 19 – travamento da imagem em 1 segundo



Fonte: os autores

Figura 20 – led aceso por análise de imagem



Fonte: os autores

4 Resultados

Foi realizada uma prova de conceito utilizando um computador e um vídeo de uma via urbana movimentada em que se passam carros, motos e caminhões, assim provando o funcionamento do teste como o esperado.

5 Projetos futuros

Em um momento posterior, o projeto poderia ser recriado de modo a conter todos os componentes para a implementação, incluindo a placa solar fotovoltaica, que não era necessária num primeiro teste. Além de mais leds conectados tanto ao protótipo e posicionados na placa, assim finalizando uma primeira montagem e implementação do trabalho.

Também pode ser estudada a possibilidade de implementação de um sistema voltado para a contagem de veículos para estudo de tráfego das vias implementadas para a prefeitura da cidade, podendo ajudar a área de planejamento e melhorar a mobilidade urbana.

6 Considerações Finais

O trabalho realizado tinha como objetivo a prevenção de colisões em vias urbanas por conta da má visibilidade em paradas obrigatórias e a solução proposta foi um modelo autônomo de placa de Pare que se iluminaria ao perceber um veículo na via principal.

Para isso seria necessário um sistema embarcado feito com uma câmera, utilizada para identificar os veículos, uma placa microcontrolada e um led de sinalização. Para que o sistema funcionasse, ele viria com um painel solar que o alimentaria e manteria a bateria carregada para operação noturna.

Mesmo não realizando testes de campo, o sistema implementado e testado se mostrou funcional como esperado, e a prova de conceito ainda mostrou

que ele é de fácil implementação pois seu tamanho diminuto garante sua aplicação prática.

7 Referências

BUCKER, B. S.; SOUZA, I. C. Mobilidade urbana com segurança através da tecnologia das Cidades Inteligentes. 2018. Niterói: Universidade Federal Fluminense - Escola de Engenharia. Disponível em: https://app.uff.br/riuff/bitstream/handle/1/8065/TCC_Bianca_e_Ingrid%20(vers%E3o%20com%20ficha%20catalogr%E1fica).pdf;jsessionid=78DF5254699ACE66C32833EAE0972542?sequence=1. Acesso em: 14.set.2022.

COSTA, Y. G. (Março de 2008). SISTEMAS EMBARCADOS. Pet.com. Fonte: Pet Comunicações: http://www.petccufpb.com.br/wp-content/uploads/2008/03/Sistemas-Embarcados.pdf

FARIAS, J.; ALENCAR, M.; LIMA, I.; ALENCAR, R. Cidades Inteligentes e Comunicações. 2011. Disponível em: http://rtic.com.br/index.php/rtic/article/view/7. Acesso em: 14.set.2022.

GARRET, F. Tudo sobre Wi-Fi: entenda os diferentes padrões das redes wireless. 2021. Disponível em: https://www.techtudo.com.br/noticias/2021/02/tudo-sobre-wi-fi-entenda-os-diferentes-padroes-das-redes-wireless.ghtml. Acesso em: 14.set.2022.

KRIEGER, D. O que é Python, para que serve e por que aprender? 2022. Disponível em: https://kenzie.com.br/blog/o-que-e-python/. Acesso em: 29.set.2022.

MAGRANI, E. A internet das coisas. 2018. Rio de Janeiro: FGV Editora.

MAPS, Google. Disponível em: https://www.google.com.br/maps/@-20.5429436,-47.3984885,3a,75y,356.05h,80.5t/data=!3m6!1e1!3m4!1sWhWswiVMfpT2UO5wVDkWQw!2e0!7i16384!8i8192. Acesso em: 20.out.2022

OFICINA DA NET. As 10 maiores empresas de tecnologia do mundo. 2020. Disponível em: https://www.oficinadanet.com.br/tecnologia/26650-as-10-maiores-empresas-de-tecnologia-do-mundo. Acesso em: 9.set.2022.

OLIVEIRA, S. Internet das Coisas com ESP8266, Arduino e Raspberry Pi. 2a ed. 2021. São Paulo: Novatec Editora.

SOMMERVILLE, I. Engenharia de Software. 2018. São Paulo: Pearson.

SOUZA, R. Energia solar: como funciona, tipos, vantagens e desvantagens. 2022. Disponível em: https://brasilescola.uol.com.br/geografia/energia-solar.htm. Acesso em: 29.set.2022.

TECHTUDO. Como um Wi-Fi funciona? Entenda a tecnologia. 2015. Disponível em: https://www.techtudo.com.br/noticias/2015/02/como-um-wi-fi-funciona-entenda-tecnologia.ghtml. Acesso em: 14.set.2022.

UFPR. Curso de Engenharia Elétrica Ênfase em Engenharia de Sistemas Eletrônicos Embarcados. Universidade Federal do Paraná. 2015. Acesso em 16 de September de 2022, disponível: http://www.eletrica.ufpr.br/graduacao/noturno/embarcados.html