



UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

FACULDADE DE TECNOLOGIA - FT

ENGENHARIA DA COMPUTAÇÃO

# Sistema de rastreamento de tempo qualificatório de corridas usando sensor PIR e microcontrolador ESP32

Jimmy Villalaz dos Santos

Manaus - AM

Julho/2023

Jimmy Villalaz dos Santos

# Sistema de rastreamento de tempo qualificatório de corridas usando sensor PIR e microcontrolador ESP32

Monografia de Graduação submetida à avaliação para a Coordenação de Engenharia da Computação pela Faculdade de Tecnologia da Universidade Federal do Amazonas como requisito para a obtenção do título de Engenheiro da Computação.

Orientador(a)

Thiago Brito Bezerra, Prof. Dr.

Universidade Federal do Amazonas - UFAM

Faculdade de Tecnologia - FT

Manaus - AM

Julho/2023

## Ficha Catalográfica

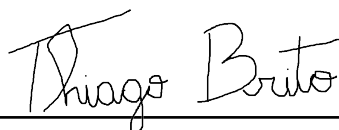
Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

S237s Santos, Jimmy Villalaz dos  
Sistema de rastreamento qualificatório de corridas usando sensor PIR e microcontrolador ESP32 / Jimmy Villalaz dos Santos . 2023  
76 f.: il. color; 31 cm.

Orientador: Thiago Brito Bezerra  
TCC de Graduação (Engenharia da Computação) - Universidade Federal do Amazonas.

1. Tempo Qualificatório. 2. Rastreamento. 3. Microcontrolador. 4. Tempo de Volta. 5. Sensor de movimento. I. Bezerra, Thiago Brito. II. Universidade Federal do Amazonas III. Título

Monografia de Graduação sob o título *Sistema de rastreamento qualificatório de corridas usando sensor PIR e microcontrolador ESP32* apresentada por Jimmy Villalaz dos Santos e aceita pela Coordenação do Curso de Engenharia da Computação da Universidade Federal do Amazonas para conceder o grau de Engenheiro da Computação como parte dos requisitos, sendo aprovada por todos os membros da banca examinadora abaixo especificada:



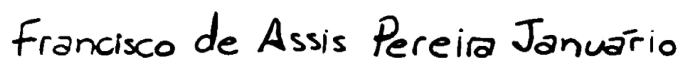
---

Prof. Dr. Thiago Brito Bezerra

Orientador(a)

Departamento de Eletrônica e Computação - DTEC

Universidade Federal do Amazonas




---

Prof. Dr. Francisco de Assis Pereira Januário

Departamento de Eletrônica e Computação - DTEC

Universidade Federal do Amazonas



---

Prof. Dr. Frederico da Silva Pinagé

Departamento de Eletrônica e Computação - DTEC

Universidade Federal do Amazonas

Manaus - AM, 06 de Julho de 2023.

À minha família e amigos.

---

# AGRADECIMENTOS

É impossível começar uma seção de agradecimentos sem antes falar daquele que faz com que a gratidão tenha significado em minha vida. Ao Eterno, criador do céu e da terra, dono de todo o universo, meu Rei. Sem a graça de D'us, Sua infinita misericórdia, bondade e amor, eu não seria nada. A Ele, toda a gratidão por tudo: desde o ar que respiro, o deitar e o levantar, e principalmente pelo cuidado que me concedeu.

Não poderia continuar falando sobre gratidão sem citar também aqui os meus maiores incentivadores, os exemplos que eu sempre tive na vida e meu maior motivo para concluir a graduação: meus pais e irmãos. Obrigado por nunca me deixarem desistir, por cuidarem de mim em todas as etapas da minha vida, por me amarem muito mais do que sempre me achei merecedor e por não medirem esforços para que eu me tornasse quem sou hoje. Saibam que de tudo que conquistei, foram vocês quem garantiram que eu conseguisse, e de tudo que eu quero alcançar e me tornar, é por vocês e para vocês que pretendo conseguir. Uma vida nunca será suficiente para demonstrar o quanto os amo e tentar retribuir um pouco de tudo que tão deliberadamente recebi.

Aos meus professores, mestres e guias nessa jornada, o tempo é a melhor maneira de descobrir quão valiosos são os ensinamentos que recebemos, e o que recebi de vocês não tem preço. Direta ou indiretamente, vocês me ajudaram a ser melhor, a aprender com os erros e a conquistar mais. A não me contentar com pouco e a estar sempre disposto a ouvir e aprender. Vocês, em especial os membros desta banca avaliadora, são excelentes profissionais e os tenho na mais alta estima. Espero que meu trabalho um dia alcance pessoas e transforme vidas da mesma maneira como o de vocês, com tanto empenho, transformou a minha. Deixo aqui meu mais sincero 'muito obrigado!'.

Por fim, dedico esta última parte aos meus, não menos importantes, amigos. Não tenho como não citá-los por tudo que já passamos juntos e que, nesse longo período de 8 anos, dividimos. Luís, Evaldo e Lucas, em todos os momentos que precisei de ajuda, vocês estiveram lá por mim. Em cada dia que me senti menos capaz, vocês também me ajudaram a enxergar um "eu" e um caminho que não imaginei que existia. Foram anos com pessoas tão questionáveis e de índole duvidosa como a de vocês. Pelos muitos trabalhos, provas e listas, horas de desespero e refeições compartilhadas, obrigado. Não estaria formando se não fosse por vocês, e serei sempre grato! (Vamos tomar um xurepinho).

E aos amigos mais recentes, quero citá-los aqui porque, sem vocês, este último ano não teria sido nem de longe tão feliz e empolgante como foi. Caroline, Jakeline, Bruno, Dávilon, João, Karen, Maria, Gabriel, Arllem e alguns outros que, se eu fosse seguir citando, ainda esqueceria e provavelmente seria injusto. Não se preocupem, tenho um lugar guardado aqui. A vocês, obrigado por estarem comigo. Cada instante que dividimos neste último ano foi muito importante para mim como profissional, como pessoa e como amigo. Ensinei e aprendi com vocês, e espero que vocês sejam para outros o que eu fui para cada um de vocês. Obrigado pela aventura que foram estes últimos períodos. O tempo vai passar, mas por meio deste, deixarei registrado meu carinho e apreço por cada um e espero que também não esqueçam. Por tudo, muito obrigado!

*“Tudo tem o seu tempo determinado, e há tempo para todo propósito debaixo do céu.”*

*Eclesiastes 3:1, Salomão.*



# Sistema de rastreamento de tempo qualificatório de corridas usando sensor PIR e microcontrolador ESP32

Autor: Jimmy Villalaz dos Santos

Orientador: Thiago Brito Bezerra, Prof. Dr.

## Resumo

Este trabalho apresenta um sistema de rastreamento desenvolvido para cronometrar os tempos qualificatórios em eventos amadores de automóveis do Kartódromo da Vila Olímpica de Manaus. Esse sistema visa garantir um melhor controle pela organização do evento, monitorando os tempos de volta de cada um dos pilotos e classificando-os de acordo com cada categoria automaticamente. Além disso, o próprio piloto pode ter acesso em tempo real as informações do seu desempenho e o público do evento também pode acompanhar mais ativamente a competição.

O projeto é baseado na integração de um dispositivo IoT (Internet of things) de baixa potência, especificamente o microcontrolador ESP32, e um sensor de presença PIR(Pyroelectric Infrared) posicionado na pista, à uma aplicação web desenvolvida em NodeJS e Angular. Através desta ocorre o cadastro do evento, dos carros, pilotos e categorias que serão disputadas. Os tempos recebidos pelo dispositivo são mostrados numa outra página da interface, onde podem ser analisados posteriormente.

*Palavras-chave:* Tempo Qualificatório, Corridas, Rastreamento, Tempo de Volta, Microcontrolador, Sensor de movimento.

# Sistema de rastreamento de tempo qualificatório de corridas usando sensor PIR e microcontrolador ESP32

Autor: Jimmy Villalaz dos Santos

Orientador: Thiago Brito Bezerra, Prof. Dr.

## Abstract

This work presents a tracking system developed to time qualifying laps in amateur car events at the Kartódromo da Vila Olímpica de Manaus. This system aims to provide better control for event organizers by monitoring the lap times of each driver and automatically classifying them according to their respective categories. Additionally, drivers themselves can access real-time information about their performance, and the event audience can actively follow the competition.

The project is based on the integration of a low-power IoT (Internet of Things) device, specifically the ESP32 microcontroller, and a PIR (Pyroelectric Infrared) presence sensor positioned on the track, with a web application developed in NodeJS and Angular. Through this application, event registration, car and driver information, and the categories to be competed in are recorded. The times received by the device are displayed on another page of the interface, where they can be analyzed later on.

*Keywords:* Qualifying Time, Races, Tracking, Lap Time, Microcontroller, Motion Sensor.

---

## LISTA DE ILUSTRAÇÕES

Figura 1 – Dados coletados manualmente do Hotlap na categoria de carros 1.0. Fonte: Hotlap da Villa . . . . .	19
Figura 2 – Dados coletados manualmente do Hotlap na categoria de carros de Dianteira Turbo. Fonte: Hotlap da Villa . . . . .	20
Figura 3 – Representação do Microcontrolador - ESP32. Fonte: GrabCAD . . . . .	29
Figura 4 – Diagrama de pinagem do ESP32. Fonte: DEVBOARD . . . . .	30
Figura 5 – Sensor PIR - Representação do sensor. Fonte: GrabCAD . . . . .	32
Figura 6 – Sensor PIR - Diagrama. Fonte: Autor . . . . .	33
Figura 7 – Tela para cadastro de carro na interface web . . . . .	36
Figura 8 – Tela para cadastro de campeonato na interface web . . . . .	37
Figura 9 – Tela para verificar os carros cadastrados na interface web . . . . .	38
Figura 10 – Tela para enviar os carros cadastrados para a pista e verificar os tempos de volta deste na interface web . . . . .	39
Figura 11 – Tela para cadastrar cada bateria, uma vez que exista um campeonato.	40
Figura 12 – Diagrama de classes com as relações entre as estruturas do backend. Fonte: Autor . . . . .	41
Figura 13 – Arquitetura do Sistema - Representação em blocos. Fonte: Autor . . . . .	42
Figura 14 – Circuito do racetracker. Fonte: Autor . . . . .	43
Figura 15 – Legenda do circuito do racetracker. Fonte: Autor . . . . .	43
Figura 16 – Arquivos utilizados para o firmware . . . . .	44
Figura 17 – Visão lateral do racetracker montado em uma protoboard . . . . .	45
Figura 18 – Visão superior do racetracker montado em uma protoboard . . . . .	45

Figura 19 – Estacionamento da Faculdade de Tecnologia. Fonte: Google Maps. . . . .	51
Figura 20 – Comparativo estatístico das baterias no UP em segundos. ME: 0.3ms. Fonte: Autor . . . . .	53
Figura 21 – Comparativo estatístico das baterias no HB20 em segundos. ME: 0.3ms. Fonte: Autor . . . . .	55
Figura 22 – Circuito Kartódromo da Vila Olímpica. Fonte: Google Maps . . . . .	56
Figura 23 – Dados coletados da categoria Dianteira Turbo na tela da Interface do sistema. Fonte: Autor . . . . .	56
Figura 24 – Dados coletados da categoria 1.0 na tela da Interface do sistema. Fonte: Autor . . . . .	57
Figura 25 – Dados coletados em classificação geral na tela da Interface do sistema. ME: 0.3ms. Fonte: Autor . . . . .	57
Figura 26 – Dados coletados distribuídos em uma tabela para tratamento estatís- tico (categorias DT e 1.0) ME: 0.3ms. Fonte: Autor . . . . .	58
Figura 27 – Dados avaliados de acordo com as métricas estabelecidas (categorias DT e 1.0) ME: 0.3ms. Fonte: Autor . . . . .	58
Figura 28 – Código de estimação da margem de erro medida manualmente feito no Matlab®Fonte: Autor . . . . .	59
Figura 29 – Resultado da estimativa da estimação da margem de erro medida manualmente Fonte: Autor . . . . .	60
Figura 30 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando a margem de erro de medição do cronômetro. Fonte: Autor	61
Figura 31 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando a margem de erro de medição do Racetracker. Fonte: Autor	61
Figura 32 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando ambas as margens de erro de medição. Fonte: Autor . . . . .	62
Figura 33 – Parte do código da estrutura do formulário em HTML do cadastro do carro . . . . .	65
Figura 34 – Parte do código da estrutura do formulário em TS do cadastro do carro	66

Figura 35 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações do carro . . . . .	66
Figura 36 – Parte do código da estrutura do formulário em HTML do campeonato	67
Figura 37 – Parte do código da estrutura do formulário em TS do campeonato . .	67
Figura 38 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações do campeonato . . . . .	67
Figura 39 – Parte do código da estrutura do formulário em HTML da home . . . .	68
Figura 40 – Parte do código da estrutura do formulário em TS da home . . . . .	68
Figura 41 – Parte do código da estrutura do formulário em HTML do painel . . . .	69
Figura 42 – Parte do código da estrutura do formulário em TS do painel . . . . .	69
Figura 43 – Parte do código do serviço da categoria dos carros, arquivo do tipo TS	70
Figura 44 – Parte do código da estrutura do formulário em HTML da bateria . . . .	70
Figura 45 – Parte do código da estrutura do formulário em TS da bateria . . . . .	71
Figura 46 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações da bateria atual, os carros e os tempos ordenados . . . . .	72
Figura 47 – Parte do código embarcado na placa para a medição dos tempos de volta . . . . .	73

---

## LISTA DE TABELAS

Tabela 1 – Tempos de volta do UP recebidos pela placa em milisegundos. ME: 0.3ms . . . . .	51
Tabela 2 – Tempos de volta do UP em segundos medidos pelo cronômetro digi- tal. ME: 0.3ms. . . . .	52
Tabela 3 – Tempos de volta do HB20 recebidos pela placa em milisegundos. ME: 0.3ms. . . . .	53
Tabela 4 – Tempos de volta do HB20 em segundos medidos pelo cronômetro digital. ME: 0.3ms . . . . .	54

---

# LISTA DE ABREVIATURAS E SIGLAS

**ACID** (Atomicidade e Consistência e Isolamento e Durabilidade)

**API** Application Programming Interface

**BD** Banco de Dados

**CPU** Central Processing Unit

**DOM** Document Object Model

**GPIO** General Purpose Input/Output

**HTML** Linguagem de Marcação de HiperTexto

**HTTP** Hypertext Transfer Protocol

**I2C** Inter-Integrated Circuit

**I/O** Input/Output

**IoT** Internet of Things

**JS** Java Script

**LED** Light Emitting Diode

**MCLR** Master Clear

**MVC** Model View Controler

**NPM** Node Package Manager

**PIR** Pyroelectric Infrared

**RC** Resistor/Capacitor

**SGBDR** Sistema de gerenciamento de banco de dados relacional

**SPA** Single Page Application

**SPI** Serial Peripheral Interface

**SQL** Structured Query Language

**UART** Universal Asynchronous Receiver/Transmitter

**URL** Uniform Resource Locator



---

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>17</b>
<b>1.1</b>	<b>Definição do problema</b>	<b>18</b>
<b>1.2</b>	<b>Objetivos</b>	<b>20</b>
1.2.1	Objetivo geral	20
1.2.2	Objetivos específicos	20
<b>1.3</b>	<b>Organização do Trabalho</b>	<b>21</b>
<b>2</b>	<b>EMBASAMENTO TEÓRICO</b>	<b>22</b>
<b>2.1</b>	<b>Aplicação Web</b>	<b>22</b>
2.1.1	JavaScript	22
2.1.2	Angular	23
2.1.3	NodeJs	25
2.1.4	PostgreSQL	26
<b>2.2</b>	<b>Sistema Embarcado</b>	<b>27</b>
2.2.1	O Microcontrolador	27
2.2.1.1	ESP32	28
2.2.2	Sensores	30
2.2.2.1	O sensor PIR	31
<b>3</b>	<b>METODOLOGIA DO TRABALHO</b>	<b>34</b>
<b>3.1</b>	<b>Sistema de rastreamento qualificatório de Corrida</b>	<b>34</b>
3.1.1	A interface web	35
3.1.1.1	Cadastro do Carro	36
3.1.1.2	Cadastro do Campeonato	37
3.1.1.3	Home (Início)	38

3.1.1.4	Painel . . . . .	38
3.1.1.5	Baterias . . . . .	39
3.1.1.6	API . . . . .	40
3.1.2	RaceTracker . . . . .	41
3.1.2.1	Modulação do circuito . . . . .	43
3.1.2.2	Firmware . . . . .	44
<b>4</b>	<b>RESULTADOS . . . . .</b>	<b>46</b>
<b>4.1</b>	<b>Preparando o ambiente . . . . .</b>	<b>46</b>
4.1.1	Iniciando o servidor local . . . . .	46
4.1.2	Conectando com o racetracker . . . . .	47
4.1.3	Rodando a aplicação web . . . . .	48
<b>4.2</b>	<b>Apresentando o modelo do teste . . . . .</b>	<b>49</b>
<b>4.3</b>	<b>Resultados do teste - Análise . . . . .</b>	<b>50</b>
4.3.1	UP Move . . . . .	51
4.3.2	HB20 Comfort Plus . . . . .	53
<b>4.4</b>	<b>Resultados no Hotlap da Villa . . . . .</b>	<b>55</b>
<b>5</b>	<b>CONSIDERAÇÕES FINAIS . . . . .</b>	<b>63</b>
<b>APÊNDICE A</b>	<b>CÓDIGO FONTE . . . . .</b>	<b>65</b>
<b>Referências</b>	<b>. . . . .</b>	<b>74</b>

# 1

---

## INTRODUÇÃO

**E**ventos de track day são realizados em autódromos, kartódromos ou pistas fechadas, nos quais os motoristas levam seus carros ao limite de desempenho, mantendo-se dentro das normas de segurança e da lei. Esses eventos proporcionam uma excelente oportunidade para a formação de novos pilotos de competição, tanto profissionais quanto amadores. No kartódromo da Vila Olímpica de Manaus, destaca-se o evento Hotlap da Villa, que pertence à categoria conhecida como Time Attack. Nessa modalidade, o piloto e seu carro correm contra o relógio, com o objetivo de conquistar o menor tempo de volta possível. Não se trata de uma corrida convencional, pois não há largada; ao invés disso, os participantes entram individualmente na pista, buscando alcançar sua melhor performance em tempo de volta. ([Regulamento Hot Lap da Villa, 2023](#)).

A cada uma delas, distribuídas em baterias, os pilotos precisam otimizar a velocidade máxima e manter o controle nas curvas, onde cada segundo e milésimo importa como também uma análise detalhada dos dados coletados é fundamental para identificar setores mais lentos, pontos específicos da pista ou técnicas de direção que podem ser aprimorados, resultando em uma redução no tempo de cada volta. Essas informações também ajudam a definir ajustes no veículo e estratégias de corrida mais eficientes, contribuindo para o desenvolvimento contínuo das habilidades dos condutores e um melhor desempenho nos próximos eventos.

## 1.1 Definição do problema

Atualmente são utilizados vários métodos para simular e prever a performance de veículos e pilotos de corrida numa pista e cada vez mais é importante obter o melhor tempo de volta possível. Seja usando modelos matemáticos, estudando as características mecânicas e geométricas do veículo ou realizando por meio de modelos computacionais simulações de volta completa, essas são algumas das formas encontradas para garantir que o tempo de volta numa qualificação seja sempre melhor e menor. Quais escolhas fazer, manobras e decisões o piloto deveria tomar e como ele se comportaria para ser o piloto ideal para realizar o menor tempo de volta (KELLY; SHARP, 2010). Do menor dos veículos de categoria mais amadora, o kart, até o mais famoso dos veículos competitivos profissionais que é a Formula 1(F1), a preocupação em garantir a primeira colocação é a mesma e isso está diretamente associado ao desempenho de um veículo e seu piloto no evento qualificatório (LOT; BIANCO, 2016).

Numa corrida real a posição de chegada dos pilotos é diretamente influenciada pela sua ordem de largada, como foi citado anteriormente. A pole position, no caso, o melhor tempo classificatório, dá a vantagem de duas posições a mais na linha de chegada, de acordo com uma análise feita nos GPs (Grand Prix de F1) dos anos de 1950 até 2013. Esse mesmo artigo comparativo mostra que a chance do pole position ganhar a corrida é em torno de 10 pontos percentuais maior do que as demais posições qualificatórias, resultando numa probabilidade de vitória estatisticamente significativa, observado também em similar magnitude em outras várias situações prováveis do GP, destacando assim a importância da posição de largada para o resultado final da corrida (WESSELBAUM; OWEN, 2021).

Contudo o melhor desempenho não necessariamente só depende das habilidades do piloto durante a volta, à velocidade, resultado da qualidade e quantidade de tecnologia do veículo (JENKINS, 2010) ou a constância de ambos numa bateria de provas. A precisão da medição do tempo, desde a largada até a conclusão de cada volta, juntamente com a forma de armazenar essas informações ao longo de várias baterias, é crucial, podendo destacar que essa medição de tempo independe do piloto, veículo e equipe. Considerando todos esses fatores e outros elementos, ter precisão na medição

do tempo é fundamental na análise e comparação de desempenho entre competidores ao longo do evento.

Em eventos que não são corridas, como o Hotlap da Villa, é crucial ter uma medição precisa do tempo. Os instantes que separam os competidores são por poucos milésimos e são muito importantes para o resultado da competição. No entanto, esses eventos ainda dependem de cronometragem manual, como podemos ver nas figuras 1 e 2, o que resulta em tempos imprecisos, e como cada milésimo de segundo é extremamente valioso, isso tem um impacto direto no resultado final do campeonato, podendo interferir na posição e classificação.

HOT LAP					
CAT. 1.0	1ª VLT	2ª VLT	3ª VLT	4ª VLT	MELHOR
Leandro	1.05.87	1.05.25	1.04.01	1.04.26	1.04.01
	1.07.10	1.07.99	1.04.86	1.04.03	1.04.03
Márcio	3.28.99	X-X-X	X-X-X	X-X-X	3.28.99
	X-X-X	X-X-X	X-X-X	X-X-X	X-X-X
Abel	1.03.29	1.06.25	1.05.33	1.02.49	1.02.33
	1.08.23	1.06.03	1.05.72	1.04.05	1.04.05
Tiago	1.02.72	1.01.85	1.01.88	1.01.25	1.01.25
	1.01.46	1.01.24	1.00.80	1.01.58	1.00.80
Ritor	1.07.70	1.08.90	1.02.61	1.08.82	1.02.61
	1.06.13	1.05.47	1.04.93	1.03.18	1.03.18
Tiago B	1.03.15	1.07.62	1.06.22	1.06.67	1.02.75
	1.04.31	1.04.54	1.04.73	1.04.47	1.04.31
Roberto	1.05.77	1.04.01	1.03.85	1.02.80	1.02.80
	1.01.55	1.03.55	1.01.93	1.04.55	1.01.55

Figura 1 – Dados coletados manualmente do Hotlap na categoria de carros 1.0. Fonte: Hotlap da Villa

	1º VOLT	2º VOLT	3º VOLT	4º VOLT	MELHOR
SILVIO	1.09.77	1.07.65	1.02.64	1.02.87	1.02.64
	1.13.10	1.12.45	1.11.31	1.02.12	1.11.31
AMELI	1.02.57	1.06.53	1.06.32	1.07.25	1.06.57
	1.31.04	1.29.64	1.12.67	1.13.29	1.12.67
BARCEL	1.02.55	1.02.25	1.02.71	1.02.04	1.02.04
	1.10.46	1.10.89	1.11.22	1.12.03	1.10.46
THALLO	1.00.38	58.80	57.19	59.42	58.80
	1.11.15	1.09.25	1.10.03	1.08.94	1.08.94
KAVIA					

Figura 2 – Dados coletados manualmente do Hotlap na categoria de carros de Dianteira Turbo. Fonte: Hotlap da Villa

## 1.2 Objetivos

### 1.2.1 Objetivo geral

Com base nesse projeto, objetiva-se o desenvolvimento de uma solução satisfatória que facilite a medição de tempo em eventos de circuito e suas modalidades de competição. Através desse sistema a lista de tempos de cada piloto e categoria no campeonato estará disponível e de fácil acesso e não haverá quaisquer interferências que possam gerar dúvidas sobre os valores medidos.

### 1.2.2 Objetivos específicos

Nesse contexto, este trabalho apresenta um projeto que utiliza de um dispositivo IoT de baixa potência com conectividade Wi-fi que consiste em um microcontrolador, o ESP32,

associado a um sensor de movimento PIR que, integrados a uma aplicação web desenvolvida em Angular e Node.js, se propõe a solucionar esse desafio da cronometragem precisa e eficaz dos tempos de volta.

Essas tecnologias permitem capturar os dados de movimento do veículo de forma mais acurada e automática, tratando os dados e os transmitindo para um banco de dados. Então, esses dados são organizados pela interface e apresentados de forma atrativa e rodando num servidor local, como em (MITROVIĆ et al., 2021) e (MACHESO et al., 2021), gerando resultados mais confiáveis e proporcionando uma análise mais detalhada de desempenho.

### 1.3 Organização do Trabalho

O trabalho foi dividido de forma que o segundo capítulo foi separado para a introduzir fundamentos e conceitos que algumas das tecnologias utilizadas no projeto se baseiam e seu funcionamento, apresentando um breve resumo de como foram usadas na resolução do problema apresentado.

O terceiro capítulo é dedicado a mostrar a construção do projeto em si, dividido entre a parte do desenvolvimento do software, seu funcionamento, e do hardware, com a montagem e suas especificações, as características e restrições de uso.

No quarto capítulo serão apresentados os resultados obtidos com o uso do projeto do kartodromo em testes reais feitos e através de simulação em um ambiente controlado.

Por fim, no quinto e último capítulo, serão apresentados os objetivos alcançados, os empecilhos observados durante a aplicação do projeto e os trabalhos futuros que serão desenvolvidos para a melhoria do mesmo.

## 2

---

# EMBASAMENTO TEÓRICO

**N**este capítulo, serão apresentados os aspectos teóricos que fundamentam este trabalho. Serão abordados os conceitos das tecnologias utilizadas, por meio de uma revisão da literatura sobre o assunto. Além disso, serão resumidos os resultados de estudos realizados por outros autores, cujas obras foram citadas e consultadas e estão listadas nas referências.

## 2.1 Aplicação Web

### 2.1.1 JavaScript

O destaque do Java Script está na sua versatilidade e adaptabilidade. Ele é utilizado não apenas no desenvolvimento web, da parte do cliente, mas também encontra aplicação na programação do lado do servidor com frameworks como o Node.js. Além disso, o JS ampliou sua presença em áreas diversas, como aplicativos de desktop, aplicativos para dispositivos móveis, rastreadores de atividades físicas, robôs e principalmente em sistemas embarcados. Destaca-se ainda o seu uso no projeto do Telescópio Espacial James Webb, onde é parte do software de controle embarcado, com a versão incorporada do JS ES1 da Nombas ([DASHEVSKY; BALZANO, 2008](#)). Logo, é indispensável este projeto utilizar frameworks que se baseiem nessa linguagem.

Segundo ([BORGES; HORA; VALENTE, 2016](#)), considerando os arquivos da base de dados do estudo realizado, o JavaScript (JS) aparece como a linguagem mais



usada nos repositórios, sendo encontrados cerca de 855 deles, algo em torno de 34,2%. A popularidade do JavaScript, entre as 10 mais usadas linguagens de programação, também é a maior de acordo com a avaliação em estrelas do mesmo, com 3.697 estrelas. De acordo com a análise feita por (WIRFS-BROCK; EICH, 2020), a importância do JavaScript pode ser atribuída ao seu crescimento inesperado e à sua ampla adoção em várias áreas. Inicialmente foi concebido como um complemento ao Java para o desenvolvimento básico de páginas da web, o JS rapidamente superou o Java e se tornou a linguagem principal para páginas interativas na web. Ao longo dos primeiros 20 anos, apesar de várias tentativas de aprimorá-lo ou substituí-lo, o JS emergiu como a linguagem de programação mais amplamente utilizada no mundo, estendendo sua presença para além das páginas da web.

Ainda com base em (WIRFS-BROCK; EICH, 2020), o surgimento do JavaScript pode ser atribuído a vários fatores. Em primeiro lugar, os requisitos de interoperabilidade da web favoreceram uma linguagem de programação dominante, e o JavaScript se mostrou adequado para esse propósito. Além disso, a evolução da teoria do cenário dos jogos em navegadores pode ter contribuído para o sucesso do JavaScript. Embora não houvesse uma necessidade intrínseca de que o JavaScript se tornasse a linguagem dominante, pela sua adaptabilidade, o amplo suporte e capacidades em constante evolução o posicionaram como uma ferramenta versátil e essencial no mundo da programação.

### 2.1.2 Angular

Seguindo o mesmo contexto de popularidade e capacidade de desenvolvimento, o Angular, um framework de desenvolvimento criado pela Google em 2010, também é uma das ferramentas mais populares para em aplicações web por todo o mundo. Se destacando por ser um framework completo e ter seu código aberto, permite criar aplicações web de página única (SPA) complexas e interativas. Ele oferece uma estrutura para desenvolver componente reutilizáveis e tem suporte à arquitetura Model View Controller.

De acordo com (SUNARDI; SUHARJITO, 2019), o MVC é um padrão arquitetural

que divide uma aplicação em três componentes principais: o modelo, a visualização e o controlador. O modelo lida com os dados e a lógica de negócio, a visualização apresenta os dados ao usuário e o controlador coordena a interação entre o modelo e a visualização. Esse padrão promove a modularidade, a reutilização de código e facilita a manutenção e o teste da aplicação e além disso, ajuda na organização do código e na colaboração entre equipes de desenvolvimento.

O principal objetivo do uso deste framework foi o de facilitar o desenvolvimento da aplicação web, fornecendo uma estrutura confiável e que pudesse ser escalável para criar a interface de usuário. Uma aplicação que fosse moderna, de boa performance e segura, permitindo uma melhor manutenção do código e facilitando a produtividade durante o desenvolvimento. ([Angular Community, 2023](#))

As características de funcionamento em que se baseiam esse framework são: a segmentação em **componentes**, encapsulando a lógica de uma parte específica da interface e garantindo a manutenção do seu próprio modelo de dados, comportamento e estilo fazendo com que cada um desses componentes continuem reutilizáveis; os **templates** definem a estrutura e aparência da aplicação ao combinar elementos HTML com diretivas do Angular mesmo o que cria uma interface dinâmica e interativa; o recurso de **Data-Binding** permite que os dados modelados pelo usuário sejam atualizados automaticamente com a visualização do projeto e assim, qualquer alteração dos dados reflita automaticamente na interface e vice-versa; o sistema de **Injeção de Dependência** torna mais fácil gerir e compartilhar dependências entre os componentes do aplicativo; suas **Diretivas**, marcadores HTML são usadas para estender o comportamento dos elementos da interface do usuário, permitindo adicionar comportamentos específicos como iterações, eventos e mesmo manipulações do DOM; o **Roteamento** possibilita navegar para diferentes partes da aplicação sem recarregar a página inteira, facilitando criar aplicações de várias páginas dentro de uma SPA e, por fim, nos **Serviços**, classes reutilizáveis que fornecem funções específicas para os componentes e são usados para compartilhar dados, realizar chamadas de API, autenticação e outras operações. ([Angular Community, 2023](#))

A curiosidade é que, a princípio quando foi criado, a finalidade do AngularJS,

como era conhecido, era apenas para uso interno da empresa para fazer aplicativos web envolventes nos diversos projetos que fossem necessários. Posteriormente que se decidiu disponibilizar o AngularJS como software de código aberto para o público, permitindo que todos criassem aplicativos online flexíveis, e que acabou se popularizando tanto quanto é visto atualmente. (GEETHA et al., 2022)

### 2.1.3 NodeJs

Como citado anteriormente, por se fundamentar em JavaScript, o Node.js consiste em um ambiente de execução de código do lado do servidor, construído com base na engine V8 do Google Chrome. Ele permite criar servidores escaláveis e de alto desempenho, característica muito interessante que o destaca como tecnologia. O custo para a sua operação com o tráfego entre 100 a 10.000 usuários é quase equivalente, variando muito pouco, como quando com acessos de até 100.000 usuários efetivamente com acesso ao servidor. (SUN et al., 2018)

Ao contrário do JavaScript no navegador, que é executado de forma assíncrona e orientada a eventos, o Node.js introduz o conceito de I/O (E/S - Entrada/Saída, ou em inglês, I/O - Input/Output) não bloqueante. Diferentemente da forma síncrona tradicional, onde uma operação bloqueia a execução do programa até que ela seja concluída, o Node.js pode lidar com várias solicitações de forma eficiente sem bloquear a execução. Ele usa um loop de eventos, orientado a eventos, para manipular as solicitações de forma assíncrona, permitindo que uma única thread gerencie várias conexões, garantindo eficiência e rapidez. (TILKOV; VINOSKI, 2010)

Ou seja, usar o NodeJs para construir um servidor web e API's, assegura que ele seja capaz de lidar com um grande número de solicitações simultâneas de forma simples com algumas poucas linhas de código. Também fornece uma API muito poderosa de rede que permite gerenciar as solicitações HTTP, o roteamento, a manipulação de arquivos e uma vasta biblioteca de módulos NPM que facilitam tanto criar quanto integrar a API com outros sistemas e serviços. Esses pacotes acrescentam funcionalidades como autenticação, manipulação de banco de dados, manipulação de documentos,

comunicação com serviços externos e outras ([Node Package Manager \(NPM\), 2023](#)).

Quem desenvolve em JavaScript pode escolher tanto no lado do cliente quanto no lado do servidor, e é o que torna o desenvolvimento de aplicativos web mais consistente e eficiente quando comparado com outras tecnologias ([LEI; MA; TAN, 2014](#)). Então aqui se tem um ambiente de execução rápido e que é escalável, podendo ser utilizado tanto por um sistema único num único lugar como até por vários numa mesma cidade ou país sem se incrementar muito seu custo como já citado, com recursos avançados para lidar com tarefas assíncronas e de rede, motivos pelos quais esta tecnologia está presente neste projeto.

#### 2.1.4 PostgreSQL

Um sistema de gerenciamento de banco de dados relacional (SGBDR) de código aberto, muito utilizado em aplicações modernas, oferece recursos avançados e é conhecido por ser confiável, ter um bom desempenho e ter conformidade com os padrões SQL ("Structured Query Language", que podem ser traduzidos para o português como "Linguagem de Consulta Estruturada"). Basicamente um SGBDR é um tipo de Banco de Dados que organiza dados em tabelas relacionadas entre si, baseado no modelo relacional proposto em meados de 1970.

O PostgreSQL funciona seguindo o modelo cliente-servidor. O servidor relacional PostgreSQL é responsável por armazenar e gerenciar dados, enquanto os clientes se conectam ao servidor para fazer as consultas e manipular estes. Quando um cliente envia uma consulta SQL para o servidor PostgreSQL, a consulta é processada pelo otimizador de consultas, que decide a melhor forma de executar a consulta com base nas estatísticas e índices disponíveis. O servidor então executa a consulta e retorna os resultados para o cliente. ([The PostgreSQL Global Development Group, 2023](#))

Por possuir uma arquitetura flexível, é possível a extensão através de módulos adicionais, chamados pelo mesmo nome "extensões", que fornecem funções extras. O PostgreSQL suporta várias características avançadas, como transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), controle de concorrência, gatilhos,

funções armazenadas, visões, herança de tabelas, replicação, entre outros. E também oferece suporte a vários tipos de dados, inclusive tipos de dados personalizados que foram definidos pelo usuário (DOUGLAS; DOUGLAS, 2003).

A garantia de um forte sistema de gerenciamento de banco de dados relacional, que foi projetado para armazenar, gerenciar e recuperar grandes quantidades de dados, justifica a escolha do mesmo para este projeto pelas suas características de confiabilidade, documentação e no conhecimento prévio já existente com um SGBDR.

## 2.2 Sistema Embarcado

### 2.2.1 O Microcontrolador

A vantagem de usar um microcontrolador é que este ocupa pouco espaço integrando muitas funcionalidades numa porção muito pequena de peça. São feitos pra operar com baixo consumo de energia, ideais para dispositivos que serão alimentados por bateria ou que dependam de requisitos de eficiência energética. Tem o custo mais acessível do que outros sistemas embarcados, ou computadores inteiros, e é uma opção mais econômica pra ser usada em alguns sistemas. Podem ser programados para uma grande variedade de aplicações, e permitem desenvolver soluções personalizadas de acordo com os requisitos específicos. Simplifica o design, já que muitos dos circuitos estão integrados num chip, reduzindo a complexidade do hardware e acelerando o desenvolvimento do projeto. (GüVEN et al., 2017)

De acordo com (GüVEN et al., 2017) estes são os principais elementos de arquitetura dos microcontroladores:

- *Alimentação* - funcionam em uma faixa de tensão de alguns volts, sendo comum alimentar microcontroladores de 8 bits com 3,3V ou 5V.
- *Reset* - um pino que serve para reiniciar a placa a partir de um sinal de tensão que é aplicado no pino. Reiniciar é fazer o programa interno do microcontrolador voltar ao início. Este pino pode receber o nome de RESET ou de Master Clear (MCLR).

- *CPU* - o cérebro do microcontrolador. Tem a função idêntica a um processador de um computador desktop doméstico ou notebook.
- *Memórias* - Existem três tipos de memórias: memória de dados, a RAM e registradores de CPU, memória de programa, comumente representada pela memória FLASH, e EEPROM, memória para o programa armazenar qualquer tipo de dado que precise ser retido.
- *Oscilador* - Esse é o circuito que gera o clock do microcontrolador. Ele pode ser interno (normalmente um circuito RC) ou externo (normalmente um circuito com cristal). Dependendo, o oscilador interno pode operar na mesma velocidade do externo, mas não necessariamente possui a mesma precisão.
- *GPIO* - periférico dos pinos de propósito geral(entradas e saídas digitais), sendo o mais comum dos microcontroladores. Este periférico é o responsável pela CPU conseguir, por exemplo, acionar um LED ligado em um pino do microcontrolador. Os pinos são divididos em grupos (quantidade que pode variar) e os grupos são indicados por letras: A, B, C, D... E a designação do pino dentro de um grupo é feita por um número.
- *Timers* - um circuito que pode ser utilizado pelo programa para criar contadores e temporizadores que rodam no hardware e deixam o programa livre para executar outras tarefas em paralelo.

Essas características são importantes a se ponderar dependendo do projeto que será construído. Assim, para conseguir cumprir as necessidades do projeto justifica-se o uso do mesmo.

#### 2.2.1.1 ESP32

Um microcontrolador de baixo custo e baixo consumo de energia baseado no processador Tensilica Xtensa LX6. Amplamente utilizado em projetos de IoT(Internet das Coisas) e em sistemas embarcados, o ESP32 possui uma arquitetura de dois núcleos permitindo que ele execute múltiplas tarefas simultaneamente e aumente o desempenho

do dispositivo. Cada núcleo tem uma velocidade de clock de até 240 MHz e suporta o conjunto de instruções Xtensa LX6.

Fabricado pela Espressif Systems, o ESP32 (ver fig.?? tem como uma das principais características a sua conectividade. Ele possui um módulo Wi-Fi integrado, possibilitando a conexão a redes sem fio e troca de dados pela internet. Além disso, também tem um módulo Bluetooth integrado, que permite a comunicação sem fio com outros dispositivos compatíveis.(Espressif Systems, 2023)

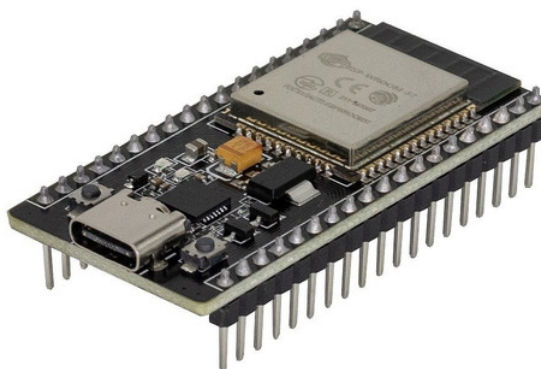


Figura 3 – Representação do Microcontrolador - ESP32. Fonte: GrabCAD

Com uma ampla variedade de pinos de entrada e saída, pode se conectar a sensores, atuadores e outros componentes eletrônicos e suportar várias interfaces, como GPIO (Entrada/Saída de Propósito Geral), UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface) e I2C (Inter-Integrated Circuit).(MAIER; SHARP; VAGAPOV, 2017)

É possível programar o ESP32 utilizando o ambiente de desenvolvimento Arduino IDE, que oferece uma interface fácil de usar e uma vasta biblioteca de funções assim como também utilizar a linguagem C ou MicroPython, como foi usado neste trabalho pra implementar o firmware da placa. Versatil, tem um baixo custo e alta conectividade sendo uma boa escolha para desenvolver um trabalho de automação e sistema eletrônico integrado.

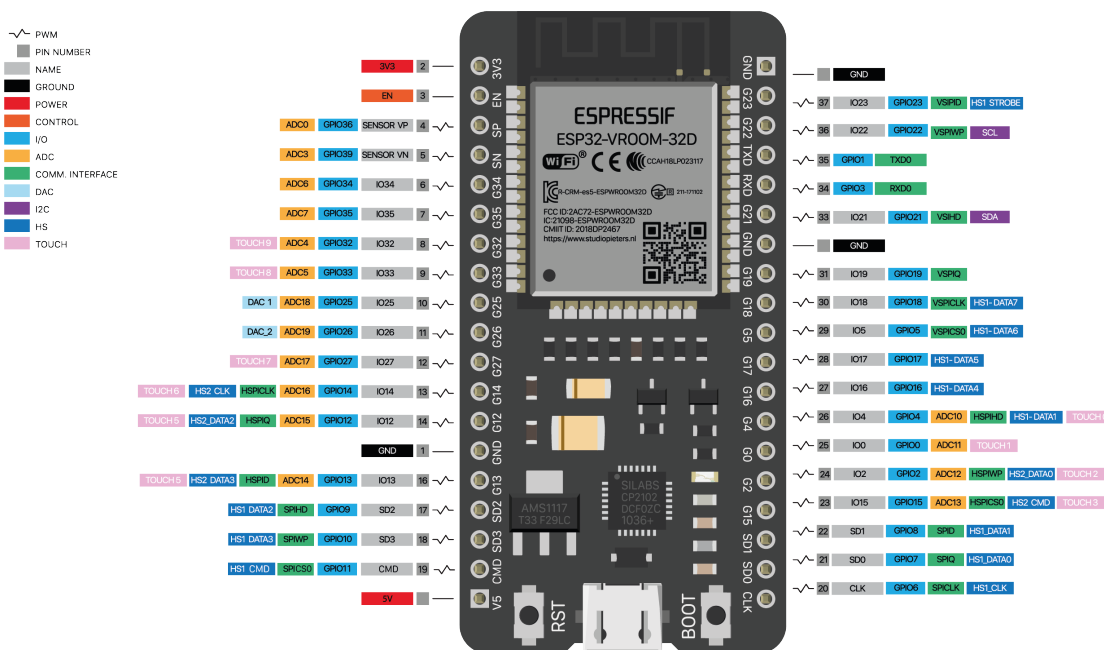


Figura 4 – Diagrama de pinagem do ESP32. Fonte: DEVBOARD

### 2.2.2 Sensores

Dispositivos que tem o trabalho de detectar e medir os diferentes estímulos ou variáveis físicas presentes no ambiente como temperatura, pressão, luz, movimento e umidade. Eles convertem essas grandezas físicas em sinais elétricos para serem processados por outros componentes eletrônicos, como microcontroladores, citados anteriormente.

Um estudo feito por (SEHRAWAT; GILL, 2019) relata sobre os diferentes princípios físicos em que os sensores eletrônicos se baseiam, dependendo do tipo de grandeza medida, e seus usos em aplicações para criar soluções com IoT, em cada um dos casos relacionando ao sensor equivalente. Como um exemplo de princípios físicos, sensores de temperatura utilizam o efeito termoelétrico, enquanto sensores de luz utilizam fotodetectores como fotodiodos ou fototransistores. Após captar o estímulo, o sensor converte a informação em um sinal elétrico proporcional à grandeza que foi medida e esse sinal pode ser analógico, variando continuamente, ou digital, com valores discretos.

Os sinais gerados pelo sensor são processados por outros componentes eletrônicos, como microcontroladores, que fazem os cálculos, tomam decisões ou acionam ações com base nas informações que foram fornecidas. Por exemplo, em um sistema de controle de temperatura, o sensor de temperatura envia um sinal ao microcontrola-



dor, que analisa o valor recebido e ativa dispositivos de aquecimento ou resfriamento conforme necessário.

Amplamente utilizados em automação industrial (MAT et al., 2016), monitoramento ambiental (ULLO; SINHA, 2020), segurança (DOU; NAN, 2017), saúde, dispositivos portáteis e veículos autônomos, entre outras aplicações como em sistemas embarcados, eles desempenham um papel crucial na coleta de dados do ambiente físico e no controle de processos, permitindo que os sistemas interajam e respondam às condições do mundo real de maneira eficiente e precisa (JAMSHED et al., 2022). E nessa característica, os sensores são uma solução viável para solucionar o problema proposto por este trabalho.

#### 2.2.2.1 O sensor PIR

Também conhecido como Passive Infrared, o sensor PIR é um dispositivo eletrônico utilizado para detectar a presença de seres vivos ou objetos em movimento com base nas emissões de calor infravermelho (GAMI, 2018). Ele consiste em um elemento piroelétrico dividido em duas metades, cada uma com uma lente que focaliza a radiação infravermelha em direção ao elemento. Quando um objeto em movimento é detectado, ocorre uma mudança no padrão de radiação infravermelha, gerando uma diferença de tensão entre as metades do sensor. (Ver Figura 5)

Esse sinal é convertido em um pulso elétrico pelo circuito interno do sensor e enviado para um dispositivo eletrônico, como um microcontrolador, que pode acionar ações com base nesse sinal, como ativar uma luz, um alarme de segurança ou indicar uma interrupção num registrador de tempo. O sensor PIR é passivo, ou seja, não emite radiação infravermelha, apenas detecta a radiação emitida pelos objetos em movimento. Devido à sua alta precisão e baixo consumo de energia, o sensor PIR é amplamente utilizado em sistemas de segurança (AKBAS; EFE; OZDEMIR, 2014), iluminação automática e controle de acesso, assim como também pode detectar a direção do movimento, entre outros (YUN; SONG, 2014).

O PIR possui dois ajustes físicos no seu módulo de sensor como visto na fig.6, o



Figura 5 – Sensor PIR - Representação do sensor. Fonte: GrabCAD

ajuste de delay de reconhecimento e medição do tempo, e o ajuste de sensibilidade do sensor, podendo ser programável e ajustável para distâncias maiores ou mais próximas e poder resolver problemas de interferências de sinais infravermelhos que não os carros que seriam o objetivo do uso do mesmo, de acordo com (ADAfruit, 2022), o ajuste fino de sensibilidade do sensor varia para até 110° de abertura, cobrindo do alcance mínimo até 10 metros de distância para a detecção de movimento. A margem de erro do tempo de medição do PIR varia de acordo com o ajuste do tempo de medição, podendo ir entre 0.3 milissegundo até 5 minutos. Para o uso neste foi usado a mínima variação de 0.3ms. Por essas características e pela sua fácil disponibilidade e baixo custo, o uso deste foi empregado neste trabalho.

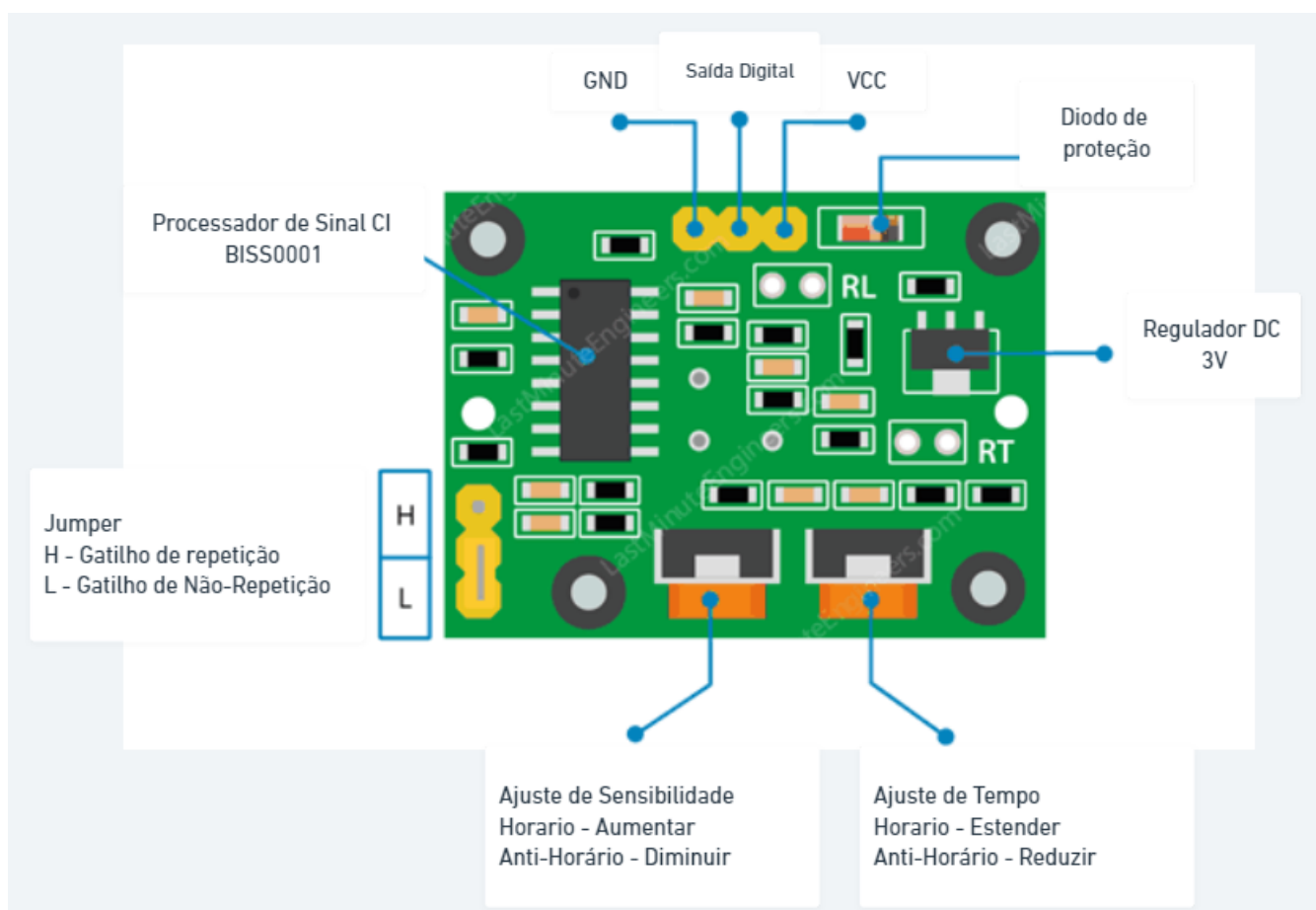


Figura 6 – Sensor PIR - Diagrama. Fonte: Autor

## 3

---

# METODOLOGIA DO TRABALHO

### 3.1 Sistema de rastreamento qualificatório de Corrida

Este trabalho apresenta um sistema de rastreamento desenvolvido para atender às necessidades específicas do Kartódromo da Vila Olímpica de Manaus e projetado especialmente para eventos organizados pelo Hot Lap da Villa, uma entidade apoiada e regulamentada pela LAM (Liga Amazonense de Motociclismo).

O foco é proporcionar uma melhor medição e organização dos tempos qualificatórios dos competidores nos eventos conhecidos como HotLaps. Os Hotlaps são etapas dos campeonatos que ocorrem ao longo do ano, nesses eventos, os pilotos participam individualmente de até duas baterias, dependendo da organização, completando 5 voltas em cada uma delas, com o intuito de alcançar o menor tempo possível, modalidade conhecida como "Time Attack", no percurso traçado. Ao final do ano, e do campeonato, ocorre a soma dos pontos acumulados durante os Hotlaps e são declarados os campeões de cada categoria marcando o final da temporada. ([Regulamento Hot Lap da Villa, 2023](#))

O sistema de rastreamento busca assegurar os tempos qualificatórios dos competidores garantindo que sejam registrados com o máximo de precisão e eficiência garantindo uma competição mais justa e transparente. Podendo assim auxiliar o melhor desempenho e desenvolvimento dos pilotos no decorrer do campeonato e por fim, também ajudar a promover e engajar o interesse do público, uma vez que os resultados e as classificações vão estar disponíveis durante a competição em tempo real e isso possibilitará que estes estejam mais envolvidos no decorrer da competição.

Com essa finalidade o projeto se baseou na integração das seguintes tecnolo-

gias: uma aplicação web desenvolvida utilizando as tecnologias NodeJS e Angular, e a integração com um microcontrolador ESP32 e um sensor de presença PIR, funcionando como o rastreador dos tempos de volta. Neste capítulo será retratado como foi construído o projeto e como ele se integra para resolver o problema apresentado.

### 3.1.1 A interface web

A aplicação web consiste na criação de uma arquitetura cliente-servidor, em que o frontend é responsável pela interface do usuário e interações no navegador, enquanto o backend cuida da lógica, manipulação de dados e integração com outros sistemas.

A arquitetura SPA do angular possui um componente responsável pela lógica e manipulação dos dados enquanto um arquivo `template` é responsável pela representação visual do formulário.

O `form.component.html` possui o código HTML juntamente com as expressões diretivas do Angular que são usadas para criar os componentes e elementos do formulário. Ele é o responsável por definir campos de entrada de texto, botões, rótulos e outros elementos viruais, podendo ter também mensagens de erro, validações de entrada, formatação condicional e outros elementos HTML.

Responsável pela lógica, o `form.component.ts` é uma classe que define o comportamento do formulário, os métodos para manipular os eventos, funções de validação e outras funções. Arquivo em TypeScript que armazena a lógica do comportamento em um arquivo Angular.

Por fim, o `service.ts` encapsula as interações com as APIs externas. É o responsável por promover a reutilização do código e separar os componentes e as camadas de acesso de dados.

Cada um dos arquivos foram construídos para cada formulário e serão descritos e apresentados a seguir.

### 3.1.1.1 Cadastro do Carro

Para criar a interface, primeiramente foi desenvolvida uma estrutura básica com um formulário em HTML. Esse formulário é responsável por receber as informações necessárias que caracterizam um carro, como modelo, nome, dono, placa e categoria. Algumas dessas informações podem não estar definidas no momento, mas serão apresentadas posteriormente.

A estrutura do formulário em HTML está representada na figura 33 do Apêndice A. Após a criação do formulário, é necessário formatar a parte lógica do sistema. Nessa etapa, são definidas as características do objeto do tipo carro e essas características são associadas aos campos previamente definidos na estrutura do formulário em HTML.

Na figura 34 do mesmo apêndice A, estão apresentadas as características lógicas do carro que será cadastrado. Essa representação visualiza de forma clara as propriedades e funcionalidades do objeto carro.

Dessa forma, o resultado exibido na tela será uma interface [7] que permite ao usuário preencher o formulário com as informações do carro e, por meio da lógica implementada, esses dados serão armazenados de acordo com as características definidas.

Figura 7 – Tela para cadastro de carro na interface web

Para efetuar o armazenamento dos dados recebidos, o serviço "car.service.ts" é utilizado para estabelecer a comunicação com a API e gerenciar as informações. Esse

serviço desempenha um papel crucial ao lidar com as operações de envio e recuperação de dados do banco de dados.

Por meio deste, é possível enviar os dados coletados do formulário para a API, onde serão processados e armazenados no banco de dados correspondente. Essa comunicação é essencial para garantir a persistência dos dados e permitir o acesso futuro a eles.

Detalhes adicionais sobre a implementação e funcionalidades do "car.service.ts" podem ser encontrados na figura 35 do Apêndice A, fornecendo uma visão mais aprofundada sobre a sua importância no contexto do sistema de gerenciamento de carros.

### 3.1.1.2 Cadastro do Campeonato

De forma semelhante, foi desenvolvida a estrutura visual do formulário para o cadastro de um campeonato [8]. Essa estrutura consiste em uma única descrição e um botão, onde uma parte do código correspondente pode ser observada na figura 36 do Apêndice A.

Após a criação da estrutura em HTML, é necessário implementar a lógica em TypeScript para o formulário do campeonato. O código correspondente, incluindo os métodos relacionados, é apresentado na devida figura 37 do Apêndice A.

A tela resultante é a interface de cadastro do campeonato, na qual o usuário pode preencher as informações necessárias por meio do formulário.

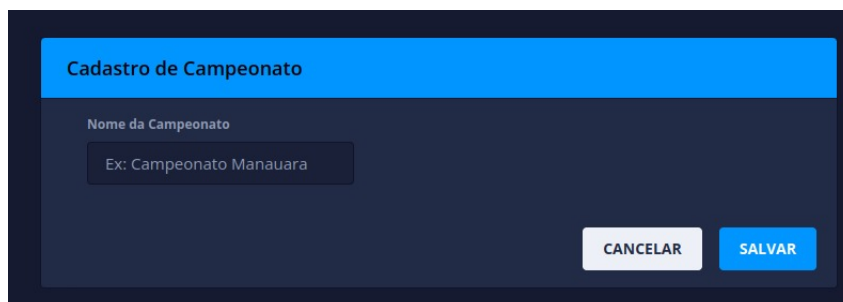


Figura 8 – Tela para cadastro de campeonato na interface web

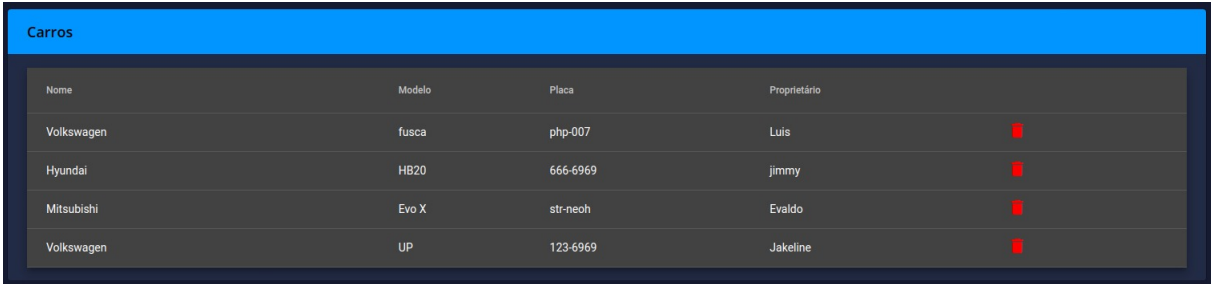
A cada campeonato, são atribuídas informações essenciais, como a quantidade de baterias, os competidores participantes, seus respectivos carros e os tempos de volta

registrados. Esses dados permitem a filtragem dos resultados, tanto por competidor individualmente quanto por categorias específicas. Além disso, é possível visualizar um quadro geral dos resultados, que será explorado em etapa posterior da análise.

Essa funcionalidade é fundamental para oferecer aos usuários uma visão personalizada dos resultados do campeonato, permitindo que eles obtenham informações específicas sobre competidores e categorias de interesse. Na figura 38 do Apêndice A, é possível consultar detalhes adicionais sobre a implementação dessa funcionalidade, proporcionando uma compreensão mais abrangente de como os resultados são filtrados e apresentados aos usuários.

### 3.1.1.3 Home (Início)

A página Home [9], ou início, tem uma tabela com os veículos cadastrados, nela pode-se ver as informações de cada competidor, organizados por ordem de cadastro. Também é possível apagar entradas ou visualizar os tempos individuais de um piloto. Para conseguir mais detalhes de implementação basta ver as figuras 39 e 40 do Apêndice A.



Nome	Modelo	Placa	Proprietário	
Volkswagen	fusca	php-007	Luis	■
Hyundai	HB20	666-6969	jimmy	■
Mitsubishi	Evo X	str-neoh	Evaldo	■
Volkswagen	UP	123-6969	Jakeline	■

Figura 9 – Tela para verificar os carros cadastrados na interface web

Ela pede os carros cadastrados, os ordena e mostra numa tabela com todas as informações elencadas.

### 3.1.1.4 Painel

O painel [10] mostra o atual carro que está correndo na pista, seus tempos de voltas na bateria e permite que se altere o carro que vai pra pista. Após cumprir todas as voltas



preestabelecidas pela bateria o tempo deixa de ser contado e os dados são exibidos até que outro carro vá para a pista. (Ver figuras 41 e 42 do Apêndice A)

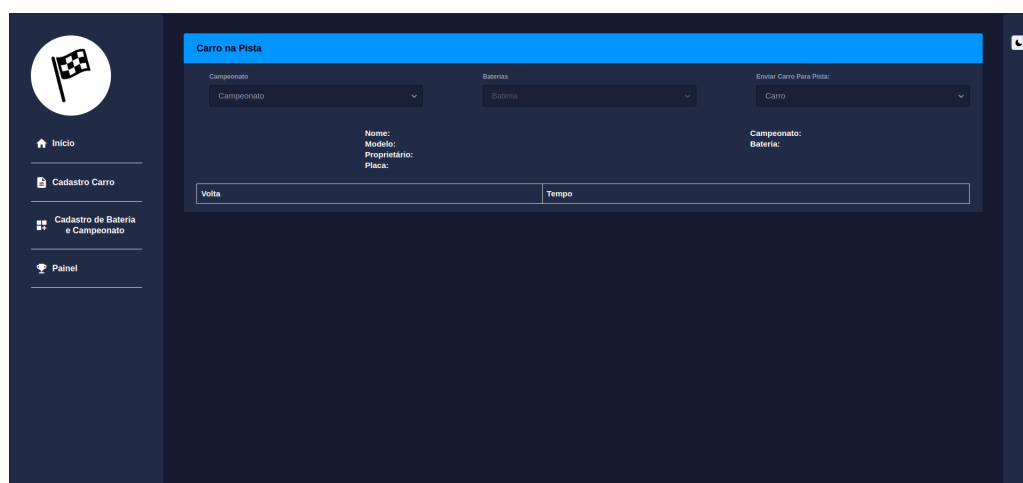


Figura 10 – Tela para enviar os carros cadastrados para a pista e verificar os tempos de volta deste na interface web

Como serviço, "Category.service.ts" é responsável por facilitar a comunicação com o backend e permitir a filtragem dos melhores tempos de volta, tanto por competidor como por categorias específicas, além de possibilitar a visualização dos melhores tempos gerais. No entanto, os caminhos de interação entre o frontend e o serviço ainda não foram implementados em um formulário HTML. É importante ressaltar que a tecnologia necessária para estabelecer essa conexão está disponível e pode ser utilizada para futuras implementações. Ver Figura 43 do Apêndice A.

### 3.1.1.5 Baterias

Cadastrar cada bateria que será realizada é importante pro esquema relacional dos dados apresentados. Os arquivos HTMS e TS estão disponíveis no Apêndice A, nas figuras 44 e 45, e mostram a estrutura tanto do formulário quanto da lógica que envolve os campeonatos e algumas outras características para poder ser criada com êxito.[11]

Por fim, o serviço "round.service.ts" é responsável por caso existam os carros na pista ele estabelecer a relação entre o campeonato e as informações do piloto, ele adquire e gerencia as informações relacionadas às outras rotas do sistema, e associa aos



Figura 11 – Tela para cadastrar cada bateria, uma vez que exista um campeonato.

tempos de volta adquiridos. Ver Apêndice A, figura 46.

#### 3.1.1.6 API

Para permitir a comunicação entre os diferentes componentes do software, a construção da API (Application Programming Interface) ocorreu em etapas. A primeira etapa foi a definição das funções e informações que serão acessíveis por meio dela, indicando quais dados serão fornecidos, manipulados e quais operações serão permitidas.

Na segunda etapa, definiu-se os endpoints (URLs) que serão utilizados para acessar esses recursos. Cada endpoint corresponde a uma determinada ação ou operação que pode ser realizada na API.

A terceira etapa consistiu na implementação da lógica que irá tratar as requisições feitas e fornecer as respostas adequadas. Após essa implementação, os endpoints são testados para verificar se a estrutura está consistente. No caso deste projeto, a API foi construída utilizando o Node.js e a sua estrutura está apresentada no diagrama de classes encontrado na figura.12.

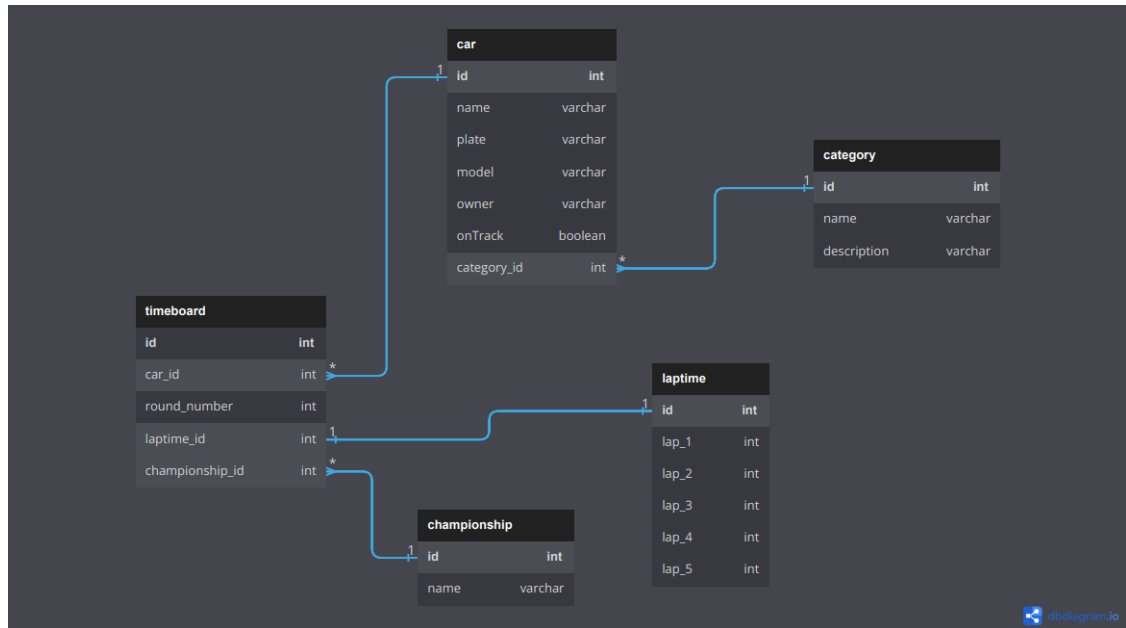


Figura 12 – Diagrama de classes com as relações entre as estruturas do backend. Fonte: Autor

Neste diagrama são apresentadas as relações entre as classes que fazem parte da lógica do backend. Cada categoria possui sua própria identificação, nome e descrição, se relacionando com um atributo da classe carro, que possui identificação nome, placa, modelo, dono e um atributo que indica se está correndo na pista atualmente ou não, o que é utilizado pela tela de painel para obter as informações necessárias para disposição dos resultados. A classe laptime possui sua identificação e armazena os tempos de voltas feitos, requeridos como atributo da classe principal, a timeboard que recebe os atributos da classe carro, os tempos de volta de laptime e o nome e identificação do campeonato, e além disso tem o número da bateria que está sendo competida para consultas futuras deste campeonato, diferenciando uma da outra.

### 3.1.2 RaceTracker

A seguir, na fig.13, pode-se observar um diagrama que representa a arquitetura do sistema funcional e que apresenta o fluxo dos dados desde o piloto até a interface do sistema.

Os pilotos correm na pista, onde o racetracker está posicionado. Quando o carro

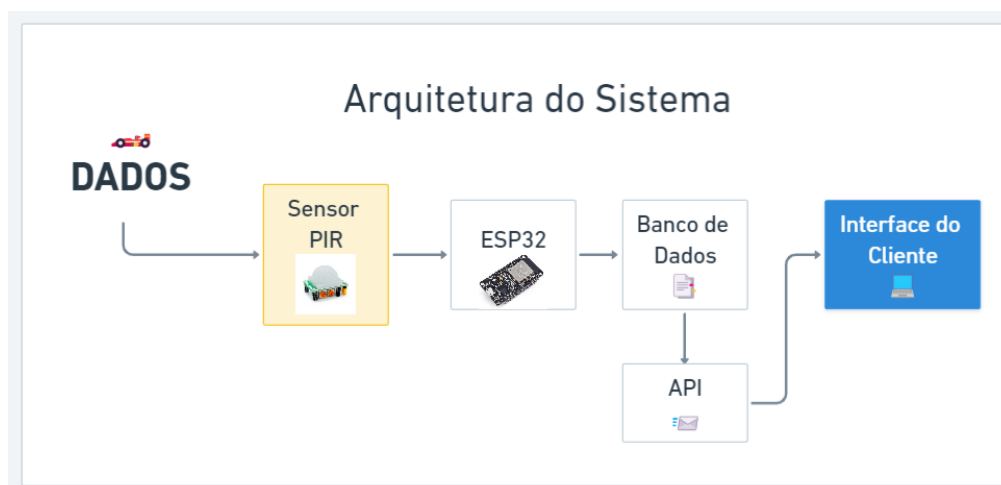


Figura 13 – Arquitetura do Sistema - Representação em blocos. Fonte: Autor

passa em frente ao racetracker o sensor PIR capta o sinal infravermelho e sinaliza um tempo de volta percorrida. Esse dado é enviado para o microcontrolador que o associa a um tempo de volta específico programado pelo firmware. Após esse dado ter sido coletado ele é atualizado através do websocket diretamente para o banco de dados da aplicação, que recebe este dado e o associa à classe laptime com uma identificação de volta recebida do ESP32. Então, a API recebe a requisição dos dados do banco, e repassa essa informação do banco de dados para a interface do cliente, mostrando em tempo real a informação recebida diretamente da pista.

Para o plano de prototipagem do circuito do racetracker, foram utilizados os seguintes componentes: Esp32, Sensor PIR, LED Vermelho, LED Brando, Push-Button e 3 resistores de 220ohm. Ele foi projetado num software online e foi testado através de simulações antes de ser montado no protótipo.

### 3.1.2.1 Modulação do circuito

O modelo do protótipo esquematizado no software Fritzing[14].

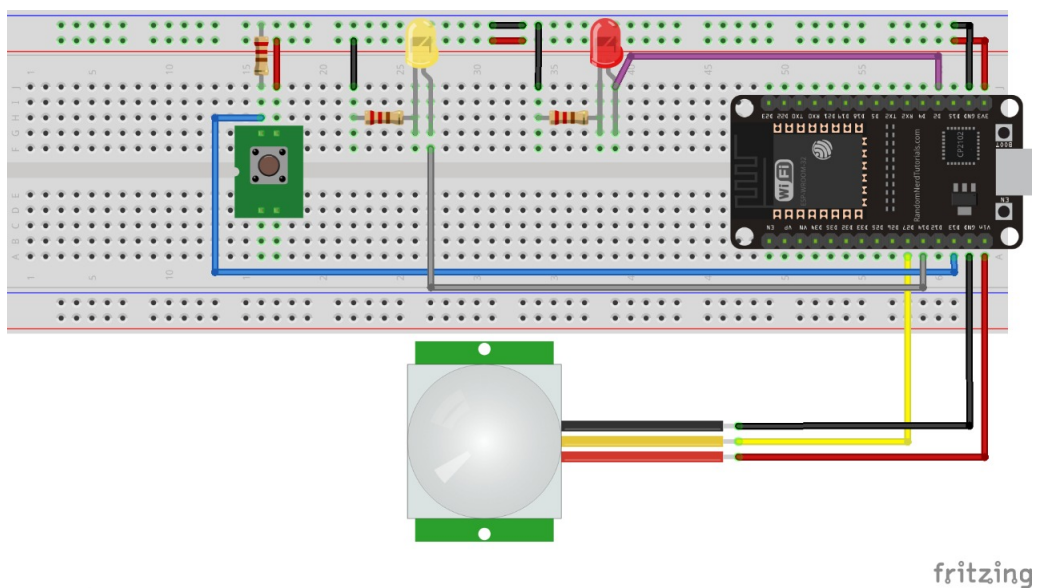


Figura 14 – Circuito do racetracker. Fonte: Autor

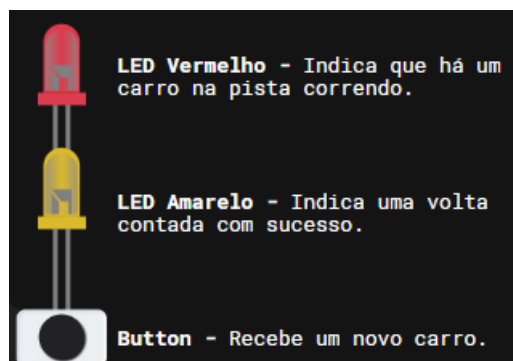


Figura 15 – Legenda do circuito do racetracker. Fonte: Autor

### 3.1.2.2 Firmware

Para executar o projeto localmente [16], é necessário instalar o MicroPython na placa, e na primeira vez que for necessária a instalação deve-se seguir os respectivos passos. Os passos para realizar essa instalação podem ser encontrados no link fornecido pela ESPRESSIF.

- Instalar a ferramenta *esptool* para gravar o firmware do MicroPython na placa.
- Integração através do Wi-fi, empregando o primeiro comando no boot da placa de se conectar com o wifi local.
- Contato com o backend e notificar se está ou não recebendo um carro na pista.

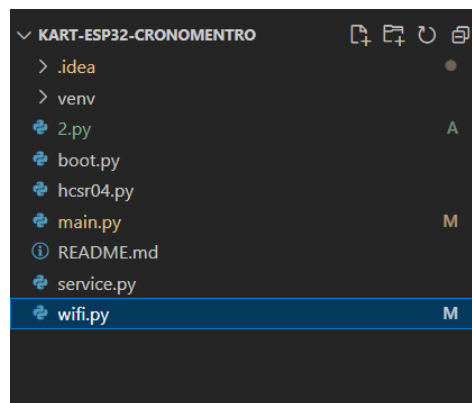


Figura 16 – Arquivos utilizados para o firmware

Parte do código da main do firmware embarcado na placa está na figura 47 do Apêndice A. Por fim, após a embarcação, o dado circuito é montado para testes reais, podendo finalmente ser utilizado em algum evento. Segue a seguir fotos do mesmo montado pouco antes de realizar os testes:

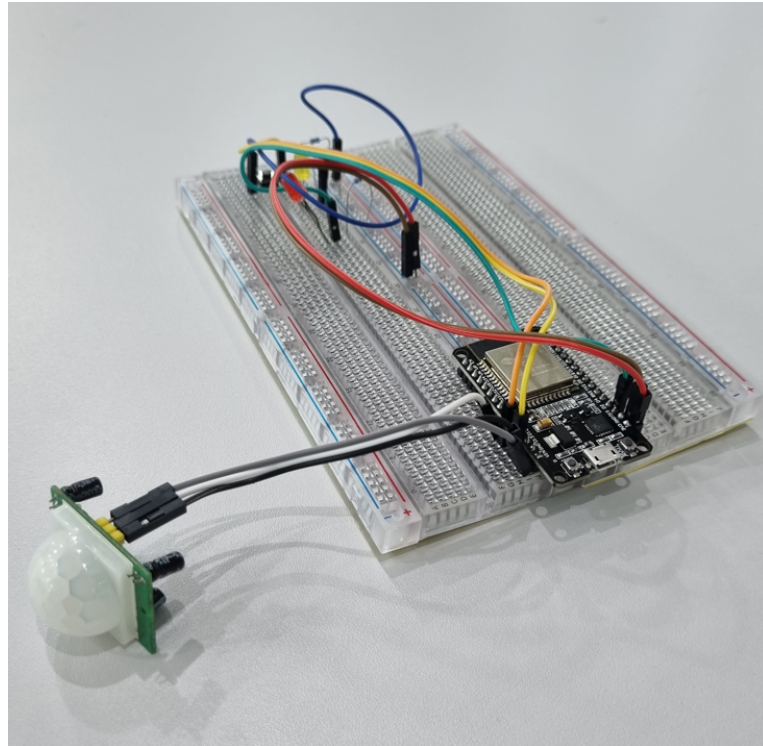


Figura 17 – Visão lateral do racetracker montado em uma protoboard

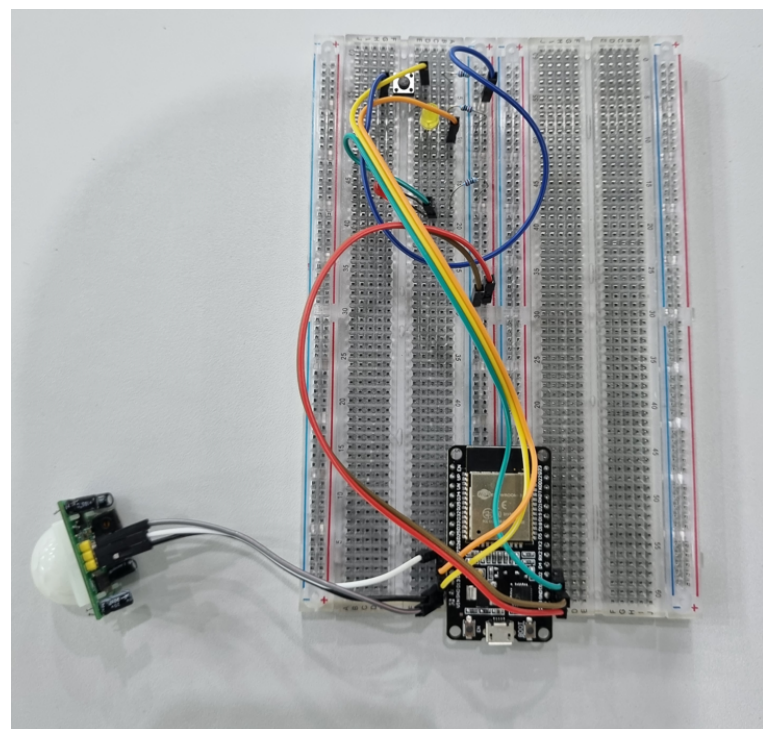


Figura 18 – Visão superior do racetracker montado em uma protoboard

## 4

---

# RESULTADOS

**P**ara realizar os testes uma série de passos foi seguida para garantir uma melhor aproximação científica do resultado desejado. O circuito a ser percorrido foi demarcado numa área específica, um sentido e orientações foram estabelecidas e foram colocadas contenções para evitar a maior interferência de outros veículos ou pessoas. Além do sistema, uma pessoa ficou responsável por marcar o tempo de volta do veículo com um cronometro digital, enquanto outra anotava os resultados dos tempos nos testes. Testes estes que aconteceram em duas oportunidades no estacionamento da Universidade Federal do Amazonas, onde um foram feitas um total de 10 baterias, cada uma com 5 voltas. O computador utilizado se conectou a placa via Wifi e os tempos de volta foram medidos via dispositivo e cronômetro ocorrendo simultaneamente.

### 4.1 Preparando o ambiente

#### 4.1.1 Iniciando o servidor local

Com o repositório baixado na maquina que irá hospedar o banco de dados, é necessário se certificar de:

- Antes de abrir o projeto ou tentar instalar as dependências do mesmo.
- Instalar o NodeJs, o PostgreSQL e o Yarn.
- Abrir a pasta do projeto via linha de comando e executar o comando:



```
$ yarn install
```

- Criar um arquivo chamado `.env` na pasta raiz do projeto para gerenciar as variáveis de ambiente. Adicione nesse arquivo a variável:

```
$ DATABASE_URL=postgres://<dbuser>:<dbpassword>@127.0.0.1:5432  
/racertrack
```

substituindo o `<dbuser>` e o `<dbpassword>` pelo usuário e senha que foi criado quando foi instalado o PostgreSQL.

- Numa janela do terminal aberta no caminho da pasta do projeto, basta executar o comando:

```
$ yarn start
```

### 4.1.2 Conectando com o racetracker

Após toda a instalação necessária, usando o Pycharm, é feito o upload dos arquivos na placa por ter um plugin embutido para micropython.

- Clonar o repositório do projeto.
- Abrir a pasta do repositório no terminal e rodar os seguintes comandos:

```
$ source venv/bin/activate'
```

```
$ cd venv/lib/python3.9/site-packages/serial/tools/
```

```
$ python3 -m miniterm /dev/ttyUSB0 115200
```

Dessa forma o microcontrolador pode se conectar a mesma rede de internet que o host do backend, assim eles fazem a comunicação, então configurar o arquivo de Wi-fi com as informações da rede e senha é uma das coisas que precisam ser observadas para que este se conecte assim que iniciar o ciclo.

### 4.1.3 Rodando a aplicação web

Por fim, para rodar o visual da aplicação web, é necessário:

- Instalar o NodeJs e o Angular.
- Se conectar na mesma rede que as requisições. O acesso pela API vai funcionar, independente do computador que roda o backend.
- Clonar o repositório na máquina.
- Abrir a pasta deste pela linha de comando e executar o código:

```
$ npm install
```

para instalar as dependências necessárias para a execução.

- Ainda numa janela do terminal aberta no caminho da pasta do projeto, executar o comando:

```
$ npm start
```

e o estará projeto rodando localmente na máquina. A aplicação específica foi gerada com Angular CLI version 11.2.10.

## 4.2 Apresentando o modelo do teste

Basicamente os testes foram feitos num ambiente pouco controlado a interrupções do que o desejado, visto que a área não era devidamente uma pista para a corrida. Entretanto, visto a natureza do projeto, não configurou-se como fator determinante para invalidar os resultados.

Os carros que foram utilizados foram dois, um HB20 Comfort Plus 1.6cc com tração dianteira aspirado, e um UP Move 1.0cc também com tração dianteira aspirado. Dois pilotos fizeram o percurso durante as baterias e os protocolos de coleta de dados foram padronizados, com instruções sobre como realizar as medições, buscando evitar ao máximo vieses muito fortes ou variações desnecessárias na coleta de dados.

Buscando avaliar os resultados obtidos foi feito um levantamento estatístico para avaliar os tempos em cada bateria. O objetivo deste estudo estatístico é de verificar o desempenho individual de cada carro nas baterias que ele competiu e avaliar comparativamente o tempo medido pela placa e o medido pelo cronômetro, com o intuito de verificar a diferença no método de medição e após isso apresentar os resultados obtidos.

O tratamento dos dados vai se basear numa análise de alguns parâmetros estatísticos definidos que são: média, mediana, o desvio padrão e o intervalo entre o maior e menor tempo.

O Cálculo da média mostra aproximadamente quanto é o tempo, no total das voltas de cada bateria, que cada piloto demora para completar o circuito. Melhorar o tempo na média indica que seu desempenho em geral no trajeto especificado também melhorou por consequência. Ele se dá pela seguinte fórmula:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (4.1)$$

A mediana é o valor que separa a metade inferior e a metade superior de um conjunto de dados ordenados. A formula é, para um conjunto de dados ordenado, dada por:

$$\text{Mediana} = \begin{cases} x_{\left(\frac{n+1}{2}\right)}, & \text{se } n \text{ é ímpar} \\ \frac{x_{\left(\frac{n}{2}\right)} + x_{\left(\frac{n}{2}+1\right)}}{2}, & \text{se } n \text{ é par} \end{cases} \quad (4.2)$$

Através da mediana pode-se comparar o tempo de cada volta e inferir tanto para um tempo mais rápido ou mais lento segundo qual valor ocupa a posição central da amostragem de dados. Num dia ou numa bateria específica, pode se verificar se o seu desempenho está melhorando ou piorando com o decorrer das voltas e avaliar o cansaço ou alguma outra característica que possa servir de indicador para a melhora ou piora do tempo.

Para o desvio padrão(amostral) a equação que determina seu comportamento estatístico é escrita por:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (4.3)$$

O desvio padrão é uma medida de dispersão que indica o quanto os valores estão afastados da média, no caso, o quanto o conjunto de dados dispersa da média, se os resultados são mais próximos ou afastados.

Por fim, o intervalo calculando a diferença entre o maior valor e o menor, indica geralmente alguma forte discrepância de resultados. A formula que será usada é:

$$\text{Intervalo} = x_{\text{máx}} - x_{\text{mín}} \quad (4.4)$$

### 4.3 Resultados do teste - Análise

Demarcado de amarelo, na figura 19 está o circuito em que se realizaram os testes feitos para este trabalho.



Figura 19 – Estacionamento da Faculdade de Tecnologia. Fonte: Google Maps.

#### 4.3.1 UP Move

Os resultados coletados estão apresentados ordenados pelas tabelas. A primeira tabela, assim como as posteriores, mostra em quanto tempo o respectivo carro fez cada volta no decorrer das baterias. Essa apresenta os tempos em milissegundo porque é como a placa faz a medição. O resultado apresentado em cada tabela é a subtração do tempo total pelo tempo anterior, uma vez que o microcontrolador quanto o cronômetro digital medem o tempo em continuamente apenas marcando o instante que o carro faz cada volta, seja no botão ou pelo sensor.

Tabela 1 – Tempos de volta do UP recebidos pela placa em milissegundos. ME: 0.3ms

Bateria	Volta 1	Volta 2	Volta 3	Volta 4	Volta 5
1	27120	30560	31230	29140	28210
2	30350	29380	32250	31290	30180
3	29470	33140	28890	29750	32460
4	31530	29720	32680	28450	30860
5	30940	30120	28560	33210	31890
6	32780	28940	31820	29530	31150
7	28860	33450	29410	32470	29820
8	33120	28880	31380	29790	34250
9	32010	31760	30450	32960	29180
10	30260	32690	29130	33640	29750

Cada um dos tempos foi bem próximo uma vez que o percurso não apresentava muita diferença de relevo ou facilidade para manobrar e é parte do que foi padronizado

como orientações do teste de de não adquirir muita velocidade, visto o ambiente do estacionamento e as restrições por ser um local público.

Tabela 2 – Tempos de volta do UP em segundos medidos pelo cronômetro digital. ME: 0.3ms.

Bateria	Volta 1	Volta 2	Volta 3	Volta 4	Volta 5
1	28.85	32.61	33.24	31.08	29.97
2	32.35	31.45	34.32	33.41	32.08
3	31.45	35.29	30.84	31.96	34.61
4	33.58	31.74	34.84	30.20	32.86
5	33.03	32.10	30.54	35.36	33.91
6	35.04	31.17	33.96	31.44	33.17
7	31.03	36.08	31.45	34.66	31.79
8	35.26	31.18	33.68	31.99	36.46
9	33.97	34.02	32.46	35.05	31.17
10	32.31	35.03	31.06	35.88	31.79

Aparentemente os resultados apresentados já se configuram com maiores ao se comparar com os medidos pelo sensor da placa. O ângulo de abertura do Sensor PIR ocupa um cone de graus que pode ser definido no mesmo ajustando tanto o alcance como a sensibilidade, provavelmente um dos fatores que justifiquem a diferença de valores também seja o tempo de largada e a diferença da resposta.

Pode-se ver que a diferença varia muito pouco em alguns casos, tomando a média por campo de análise, mas em outras baterias é de quase 3 segundos. No intervalo entre os menores tempos e os maiores, a disparidade também aumenta quando comparando ao que foi medido pelo sensor. É possível observar que desvio padrão aumenta no decorrer das baterias, o que indica que os dados estão se apresentando mais dispersos do que agrupados em torno da média, o que significa que temos dados mais heterogêneos com uma distribuição mais ampla, o que justifica também o intervalo maior quando o desvio padrão aumenta. O que isso quer dizer é que pode ser inferido que a acurácia e precisão dos resultados pelo sensor é maior, mostrando mais tempos dispersos e distintos entre si, retratando de maneira interessante e fidedigna a alta variabilidade das causas nas pistas de corrida. Algo que vai poder ser visto também nos resultados do HB20.

		UPI				
		Bateria	Média	Mediana	Desvio Padrão	Intervalo
Cronômetro	1	29,97	29,97	1,576848122	2,23	
	2	32,72	32,35	1,13967978	2,87	
	3	32,83	31,96	1,990062813	4,45	
	4	32,64	32,86	1,770276815	4,64	
	5	32,99	33,03	1,820678445	4,82	
	6	32,96	33,17	1,649615107	3,87	
	7	32,00	31,45	2,419167212	6,42	
	8	33,69	33,68	2,217990081	5,28	
	9	34,12	33,97	3,003869172	7,93	
	10	33,50	33,23	1,965189558	4,82	
		Bateria	Média	Mediana	Desvio Padrão	Intervalo
Racetracker	1	29,252	29,14	1,678323568	4,11	
	2	30,69	30,35	1,105147049	2,87	
	3	30,742	29,75	1,919236827	4,25	
	4	30,648	30,86	1,63127864	4,23	
	5	30,944	30,94	1,760122155	4,65	
	6	30,844	31,15	1,592648737	3,84	
	7	30,802	29,82	2,029007146	4,59	
	8	31,484	31,38	2,235426134	5,37	
	9	31,272	31,76	1,473455123	3,78	
	10	31,094	30,26	1,961410207	4,51	

Figura 20 – Comparativo estatístico das baterias no UP em segundos. ME: 0.3ms. Fonte: Autor

### 4.3.2 HB20 Comfort Plus

Os testes agora foram realizados com outro piloto num carro diferente. Tem-se resultados equivalentes devido ao percurso ter permanecido o mesmo, mas a pessoa que segurava o cronômetro mudou, o que pode ter acarretado nas diferenças maiores.

Tabela 3 – Tempos de volta do HB20 recebidos pela placa em milissegundos. ME: 0.3ms.

Bateria	Volta 1	Volta 2	Volta 3	Volta 4	Volta 5
1	30540	28980	30120	32250	27560
2	29870	31020	29230	30290	31510
3	29460	31970	29750	28890	28150
4	30210	29780	31290	32650	30520
5	28190	29050	29840	28760	33120
6	31420	30590	29710	28840	30360
7	30140	28630	29480	30760	28920
8	29320	30490	28670	29830	31250
9	28810	32340	29970	28940	31980
10	30060	29890	31020	32470	30330

O resultado apresentado também é a subtração do tempo total pelo tempo anterior, pelo motivo anteriormente citado. Era esperado que por ser um carro razoavelmente mais potente este tivesse um tempo muito melhor do que medidos pelo UP, contudo por ser outro piloto isso provavelmente contribuiu para uma série de baterias mais lentas do que o esperado.

Tabela 4 – Tempos de volta do HB20 em segundos medidos pelo cronômetro digital. ME: 0.3ms

Bateria	Volta 1	Volta 2	Volta 3	Volta 4	Volta 5
1	33.17	31.96	33.12	35.22	29.97
2	32.72	34.17	32.25	33.96	35.04
3	32.22	35.01	32.78	31.89	31.72
4	33.35	32.36	34.48	36.03	33.58
5	31.33	32.71	33.28	32.06	37.91
6	34.33	33.92	33.15	32.22	34.52
7	33.02	31.27	32.53	34.60	32.34
8	32.30	33.89	31.87	33.18	34.77
9	31.62	35.77	33.71	32.31	35.94
10	33.09	32.36	34.99	36.87	34.14

A confiabilidade do cronômetro não foi muito bem testada antes de iniciadas as baterias. Após o fim dos teste com o UP o novo responsável por medir os tempos das voltas precisou praticar o uso dos botões do cronometro digital visto que ele nunca havia manuseado um relógio físico. Após a primeira volta, um tempo não foi contado pelo descuido do medidor, o teste teve que ser reiniciado.

Ao avaliar os resultados nessa escala comparativa podemos notar que o sistema de qualificação é mais eficiente para a medição de tempos mais confiáveis, precisos e fiéis ao que é proposto pelo trabalho que, apesar de alguns custos, ainda apresenta muitas vantagens do que comparado ao uso do método analógico do controle do tempo. Vantagem esta constatada tanto para o UP quanto para o HB20 num mesmo circuito com 2 métodos propostos comparativos.



		HB20				
		Bateria	Média	Mediana	Desvio Padrão	Intervalo
Cronômetro	1	32,688	33,12	1,920122392	5,25	
	2	33,628	33,96	1,131490168	2,79	
	3	32,724	32,22	1,340384273	3,29	
	4	33,96	33,58	1,381285633	3,67	
	5	33,568	32,71	2,607464285	6,63	
	6	33,332	33,15	0,7721204569	1,81	
	7	32,752	32,53	1,214730423	3,33	
	8	33,202	33,18	1,174763806	2,9	
	9	33,868	33,71	1,965189558	4,32	
	10	34,29	34,14	1,755975512	4,51	
Racetracker	1	29,89	30,12	1,753710352	4,69	
	2	30,384	30,29	0,905140873	2,28	
	3	29,644	29,46	1,436655839	3,82	
	4	30,89	30,52	1,127940601	2,87	
	5	29,792	29,05	1,953194819	4,93	
	6	30,184	30,36	0,9688291903	2,58	
	7	29,586	29,48	0,873773426	2,13	
	8	29,912	29,83	1,003005484	2,58	
	9	30,408	29,97	1,666154255	3,53	
	10	30,754	30,33	1,051584519	2,58	

Figura 21 – Comparativo estatístico das baterias no HB20 em segundos. ME: 0.3ms.  
 Fonte: Autor

#### 4.4 Resultados no Hotlap da Villa

No ambiente do Kartodromo da Vila Olímpica, os eventos do HotLap possuem datas agendadas para ocorrerem durante o ano e não coincidiram com os testes feitos para o desenvolvimento deste.

Demarcado de preto, na figura 22 está o circuito em que se realizou o Hotlap de Julho.



Figura 22 – Circuito Kartódromo da Vila Olímpica. Fonte: Google Maps

Aqui estão acrescidos os resultados obtidos pelo tratamento dos dados do evento na interface de dados do sistema, nas figuras 23 e 24.

Piloto	Categoria	Carro	V.M.R.	Bateria
Thiago Brito	Tração Dianteira Turbo	1ª bateria	0:58.800	1ª bateria
Gabriel Nicolau	Tração Dianteira Turbo	1ª bateria	1:02.40	1ª bateria
Silvio Castro	Tração Dianteira Turbo	1ª bateria	1:03.650	1ª bateria
André Carvalho	Tração Dianteira Turbo	1ª bateria	1:06.120	1ª bateria
Thiago Brito	Tração Dianteira Turbo	2ª bateria	1:08.940	2ª bateria
Gabriel Nicolau	Tração Dianteira Turbo	2ª bateria	1:10.460	2ª bateria
Silvio Castro	Tração Dianteira Turbo	2ª bateria	1:11.810	2ª bateria
André Carvalho	Tração Dianteira Turbo	2ª bateria	1:12.670	2ª bateria

Figura 23 – Dados coletados da categoria Dianteira Turbo na tela da Interface do sistema. Fonte: Autor

Piloto	Categoria	Carro	V.M.R.	Bateria
Tiago Lima	1.0	2ª bateria	1:00:800	2ª bateria
Tiago Lima	1.0	1ª bateria	1:01:350	1ª bateria
Rodrigo Simões	1.0	2ª bateria	1:01:550	2ª bateria
Rodrigo Simões	1.0	1ª bateria	1:02:800	1ª bateria
Tiago Tadeu	1.0	1ª bateria	1:03:150	1ª bateria
Victor Santos	1.0	2ª bateria	1:03:180	2ª bateria
Leandro Aguiar	1.0	1ª bateria	1:04:10	1ª bateria
Leandro Aguiar	1.0	2ª bateria	1:04:30	2ª bateria
Adiel Souza	1.0	2ª bateria	1:04:50	2ª bateria
Tiago Tadeu	1.0	2ª bateria	1:04:310	2ª bateria
Adiel Souza	1.0	1ª bateria	1:05:330	1ª bateria
Victor Santos	1.0	1ª bateria	1:07:610	1ª bateria
Márcio Azevedo	1.0	1ª bateria	3:28:990	1ª bateria

Figura 24 – Dados coletados da categoria 1.0 na tela da Interface do sistema. Fonte: Autor

Aqui podemos ver os resultados obtidos ordenados por cada categoria, listando do menor ao maior tempo obtido pelos competidores reais nas baterias competidas. Então, os resultados puderam ser apresentados numa classificação geral, objetivo desse trabalho, na figura 25.

Piloto	Categoria	Carro	V.M.R.	Bateria
Thiago Brito	Tração Dianteira Turbo	1ª bateria	0:58:800	1ª bateria
Silas Souza	Tração Dianteira Aspirado	1ª bateria	0:59:520	1ª bateria
Tiago Lima	1.0	2ª bateria	1:00:800	2ª bateria
Tiago Lima	1.0	1ª bateria	1:01:350	1ª bateria
Rodrigo Simões	1.0	2ª bateria	1:01:550	2ª bateria
Gabriel Nicolau	Tração Dianteira Turbo	1ª bateria	1:02:40	1ª bateria
Rodrigo Simões	1.0	1ª bateria	1:02:800	1ª bateria
Tiago Tadeu	1.0	1ª bateria	1:03:150	1ª bateria
Victor Santos	1.0	2ª bateria	1:03:180	2ª bateria
Silvio Castro	Tração Dianteira Turbo	1ª bateria	1:03:650	1ª bateria
Leandro Aguiar	1.0	1ª bateria	1:04:10	1ª bateria
Leandro Aguiar	1.0	2ª bateria	1:04:30	2ª bateria
Adiel Souza	1.0	2ª bateria	1:04:50	2ª bateria
Tiago Tadeu	1.0	2ª bateria	1:04:310	2ª bateria

Figura 25 – Dados coletados em classificação geral na tela da Interface do sistema. ME: 0.3ms. Fonte: Autor

Os tempos coletados se apresentam de acordo com a tabela representada na figura 26. A seguir foi feito um tratamento estatístico em cima dos mesmos resultados observando-se o seguinte de cada um deles segundo a figura 27:

Dianteira Turbo				
Pilotos	Volta 1	Volta 2	Volta 3	Volta 4
THIAGO BRITO	60,36	58,8	59,19	59,42
GABRIEL NICOLAU	62,55	62,25	62,71	62,04
SILVIO CASTRO	64,77	63,65	66,32	67,35
ANDRE CARVALHO	67,57	66,53	66,12	67,35

1.0				
Pilotos	Volta 1	Volta 2	Volta 3	Volta 4
TIAGO LIMA	62,72	61,85	61,88	61,35
RODRIGO SIMÕES	65,37	64,01	63,85	62,8
TIAGO TADEU	63,15	67,68	66,22	66,64
VICTOR SANTOS	67,7	68,9	67,61	68,87
LEANDRO AGUIAR	65,87	65,25	64,01	64,76
ADIEL SOUZA	65,59	66,25	65,33	67,47
MARCIO AZEVEDO	208,99	N/C	N/C	N/C

Figura 26 – Dados coletados distribuidos em uma tabela para tratamento estatístico (categorias DT e 1.0) ME: 0.3ms. Fonte: Autor

		Piloto	Média	Mediana	Desvio Padrão	Intervalo
DT		THIAGO BRITO	59,4425	59,305	0,6630422309	1,56
		GABRIEL NICOLAU	62,3875	62,4	0,3000416638	0,67
		SILVIO CASTRO	65,5225	65,545	1,637912798	3,7
		ANDRE CARVALHO	66,8925	66,94	0,6822694971	1,45
		Piloto	Média	Mediana	Desvio Padrão	Intervalo
1.0		TIAGO LIMA	61,95	61,865	0,5679788728	1,37
		RODRIGO SIMÕES	64,0075	63,93	1,055031595	2,57
		TIAGO TADEU	65,9225	66,43	1,947551882	4,53
		VICTOR SANTOS	68,27	68,285	0,7111961755	1,29
		LEANDRO AGUIAR	64,9725	65,005	0,7861456608	1,86
		ADIEL SOUZA	66,16	65,92	0,9553358921	2,14
		MARCIO AZEVEDO	208,99	208,99	-	0

Figura 27 – Dados avaliados de acordo com as métricas estabelecidas (categorias DT e 1.0) ME: 0.3ms. Fonte: Autor

Para encontrar a margem de erro de uma pessoa real e um cronômetro digital com base nos tempos de medição manual de tempos de volta, usou-se uma abordagem de reamostragem ou bootstrap. Essa abordagem permite estimar a margem de erro a partir dos dados observados. O código a seguir, figura 28 foi feito no software Matlab®:

```
1  % Dados dos tempos de volta medidos manualmente
2  - tempos_medidos = [60.36 58.8    59.19  59.42 62.55 62.25  62.71  62.04 64.77 63.65  66.32  67.35 67.57 66.53  66.12  67.35];
3
4  % Parâmetros da estimação
5  - n_simulacoes = 1000; % Número de simulações/bootstrap
6  - n_amostras = numel(tempo_medidos); % Número de tempos de volta medidos
7
8  % Estimação/bootstrap
9  - margens_erro = zeros(n_simulacoes, 1);
10 - for i = 1:n_simulacoes
11     amostra_bootstrap = datasample(tempo_medidos, n_amostras, 'Replace', true);
12     margens_erro(i) = std(amostra_bootstrap) / sqrt(n_amostras);
13 - end
14
15 % Estatísticas das margens de erro estimadas
16 - media_margens_erro = mean(margens_erro);
17 - desvio_padrao_margens_erro = std(margens_erro);
18 - limite_inferior = median(margens_erro) - 1.96 * std(margens_erro);
19 - limite_superior = median(margens_erro) + 1.96 * std(margens_erro);
20
21 % Resultados da estimação
22 - fprintf('Margem de erro média: %.2f segundos\n', media_margens_erro);
23 - fprintf('Desvio padrão das margens de erro: %.2f segundos\n', desvio_padrao_margens_erro);
24 - fprintf('Intervalo de confiança de 95%%: [%.2f, %.2f] segundos\n', limite_inferior, limite_superior);
25
```

Figura 28 – Código de estimação da margem de erro medida manualmente feito no Matlab® Fonte: Autor

Aqui temos uma matriz **tempo\_medidos** contendo os tempos de volta medidos manualmente.

Depois, são definidos os parâmetros da estimação, incluindo então o número de simulações (**n\_simulacoes**) a serem feitas para crescer o valor estatístico das amostras e o número de amostras para serem reamostradas no bootstrap (**n\_amostras**), que é o mesmo número de tempos de volta medidos.

Essa estimação é realizada usando um loop for, em que a cada iteração são feitas reamostragens com reposição dos tempos de volta medidos usando a função *datasample*. A margem de erro é calculada como o desvio padrão da amostra bootstrap dividido pela raiz quadrada do número de amostras.

Após a estimação/bootstrap, são calculadas as estatísticas das margens de erro simuladas, incluindo a média (**media\_margens\_erro**), o desvio padrão e um intervalo de confiança de 95% (calculado como a mediana das margens de erro mais ou menos 1,96 vezes o desvio padrão).

Gerando assim os resultados incluindo a margem de erro média, o desvio padrão das margens de erro e o intervalo de confiança de 95%. Como pode ser visto aqui na figura 29:

```
Command Window
>> simMargemCronometro
Margem de erro média: 0.75 segundos
Desvio padrão das margens de erro: 0.08 segundos
Intervalo de confiança de 95%: [0.59, 0.92] segundos
fx >>
```

Figura 29 – Resultado da estimativa da estimação da margem de erro medida manualmente Fonte: Autor

Com estes dados prontos, foi possível fazer uma análise comparatória entre os tempos medidos com as margens de erro tanto do racetracker como do tempo de medição manual.

Uma matriz **tempos** armazena os tempos de volta dos pilotos, onde cada linha representa um piloto e cada coluna representa uma volta. A matriz **erros** armazena as margens de erro correspondentes do medidor utilizado.

O `for` é utilizado para iterar sobre cada piloto e plotar sua linha com a função `errorbar`. Dentro do laço, `errorbar(1:size(tempos, 2), tempos(i, :), erros(i, :), 'o-', 'LineWidth', 1.5)` é usado para traçar a linha com marcadores circulares ('o-') e largura de linha 1.5. A função `size(tempos, 2)` retorna o número de voltas para determinar a escala no eixo x.

Assim, através do software Matlab® pode-se ter os gráficos apresentados nas figuras 30 e 31. No primeiro gráfico, na figura 30, está o comparatório de tempos de volta entre os competidores da categoria DT e sua respectiva variação de erro de medição manual. No segundo gráfico, na figura 31, apresenta-se o comparatório de tempos dos mesmos competidores mas com a variação de erro de medição do racetracker.

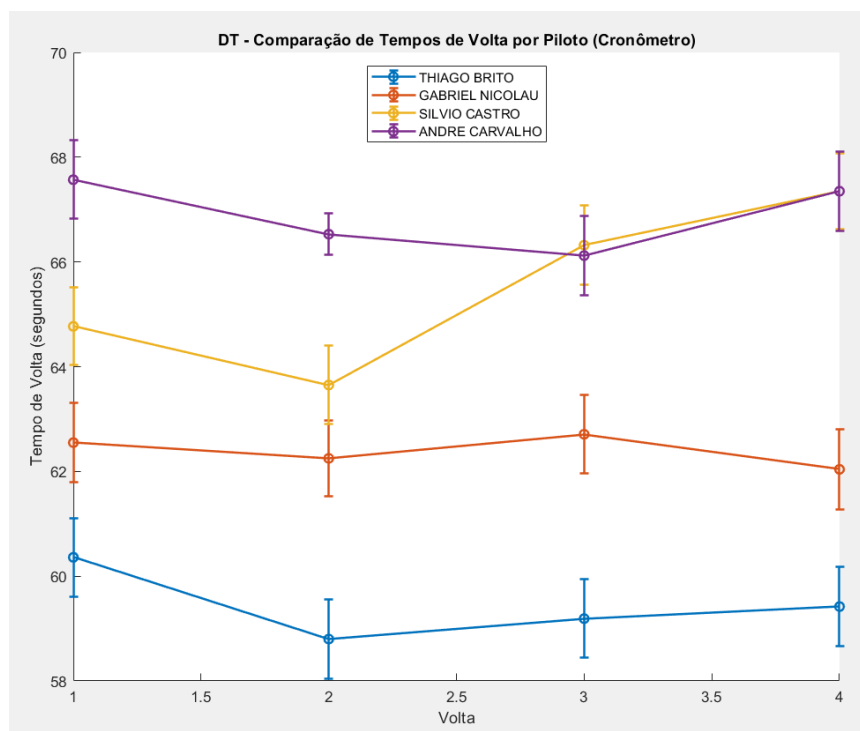


Figura 30 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando a margem de erro de medição do cronômetro. Fonte: Autor

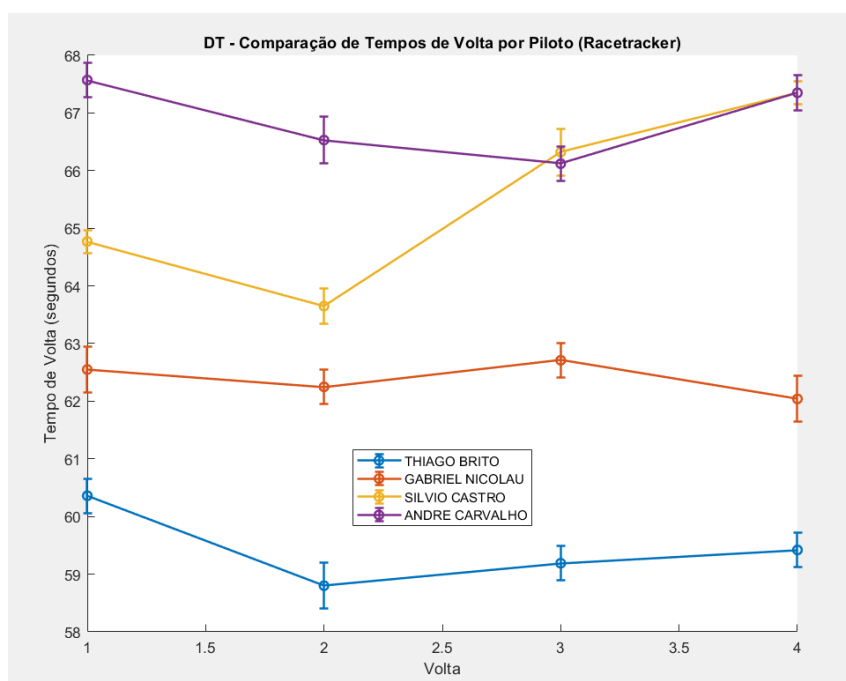


Figura 31 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando a margem de erro de medição do Racetracker. Fonte: Autor

Compilando os resultados num único gráfico comparativo em que são utilizados dois tracejados de linhas de medição identificando o conjunto de dados do racetracker e do cronômetro, pode-se perceber claramente a vantagem do uso do racetracker em detrimento do uso da medição manual. (Ver a Figura 32)

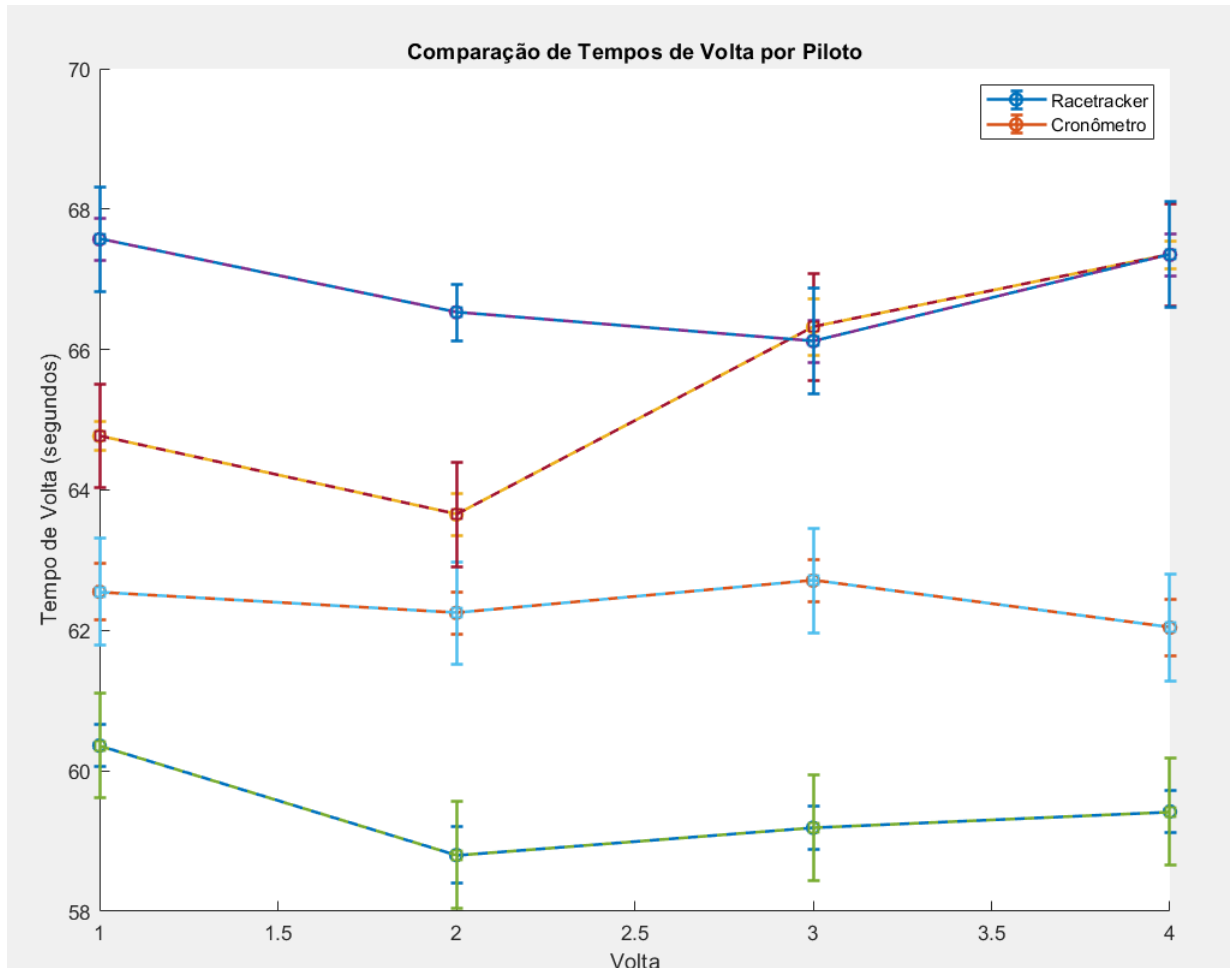


Figura 32 – Resultado comparativo dos tempos de cada piloto na categoria DT utilizando ambas as margens de erro de medição. Fonte: Autor

Assim, de acordo com a variação menor na margem de erro de medição, o racetracker interfere menos na variação dos tempos de cada piloto o que resulta em um sistema mais confiável quando comparado ao modelo de medição manual, tendo resultados obtidos mais precisos.



## 5

---

# CONSIDERAÇÕES FINAIS

O aspecto mais importante que este trabalho buscou solucionar foi medir o tempo de uma maneira confiável, rápida e concisa, organizando os resultados e tornando mais fácil a sua obtenção de maneira prática.

Dos resultados que foram medidos, através do tratamento dos dados, pode-se observar no teste do UP uma diferença de 6,54% do tempo medido através do uso do cronometro digital com interferencia humana na medição, do que foi medido através do sensor pelo racetracker, este apresentando esses valores menores.

Em comparação, o HB20 apresentou uma diferença de 7,34% em valores medidos pelo cronômetro e pelo sensor do racetracker, com mais uma vez o racetracker apresentando os menores valores. Vale ressaltar que a margem de erro dos sensores PIR variam de acordo com o ajuste do tempo que foi definido, variando de 0.3 milisegundo até 5 minutos no total. Para este teste ele foi definido no mínimo possível variando entre 0.3 ms para mais ou pra menos.

Pode-se observar que o custo do racetracker é uma alternativa de mais em conta uma vez que seu valor chega próximo a R\$90,00 reais dados os preços pesquisados em algumas lojas de eletônicos da região. Se apresentando assim como uma solução mais viável, ao invés de um sistema de telemetria utilizado em alguns kartodromos, do que o modelo *Aim Mycron5 2t com GPS integrado*, que custa em torno de R\$5.250,00 reais.

Esse sistema trás uma solução interessante que podem atender às necessidades e requisitos além de poder também melhorar o desenvolvimento do cenário automobilístico amador na região norte do Brasil e também pode trazer mais interesse e visibilidade

para o mesmo.

Apesar de ainda não ser uma solução definitiva é um passo interessante para profissionalizar a prática deste e de outros eventos competitivos cada vez mais variados, podendo tornar os campeonatos cada vez mais concorridos e desafiadores, descobrir talentos e atrair mais pilotos e entusiastas.

Algumas funcionalidades pretendidas não foram implementadas, o ajuste para uma placa definitiva juntamente com o desenho de uma estrutura para armazenar o hardware tornando-o um módulo também não pode ser feito. Houveram muitos problemas de bugs de código, e a falta de um domínio maior no uso das tecnologias também não contribuíram para que o projeto fosse feito como se esperava. O tempo foi um forte empecilho.

Contudo, há muito ainda que pode ser feito a partir deste trabalho. Como por exemplo:

- Melhorar a interface web com adição de mais telas e tabelas.
- Usar um banco de dados não relacional na estrutura.
- Armazenar o servidor na nuvem para acesso remoto.
- Usar visão computacional para incrementar a usabilidade do trabalho num escopo maior de competições e medições.
- Modularizar os Racetrackers e aplicá-los ao longo da pista para segmentalizar o percurso e avaliar o desempenho entre cada setor ao invés de cada volta.

Assim, através da realização de mais experimentos e dedicando mais tempo a este projeto, é possível alcançar resultados ainda mais promissores do que os já obtidos até o momento. Assim, este trabalho serve como um ponto de partida para avançar e explorar ainda mais essa área de estudo.

# A

## CÓDIGO FONTE

Neste Apêndice foram listadas algumas partes do código fonte utilizado neste trabalho. O código fonte completo pode ser encontrado no Github do autor<sup>1</sup>.

```

src > app > pages > car-form > car-form.component.html > nb-card
4 </no-card-body>
5
6
7 <form [formGroup]="carForm" (ngSubmit)="submitForm()">
8
9   <div class="form-row">
10
11     <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
12       <label for="model" class="label">Modelo</label>
13       <input maxLength="20" fullWidth type="text" nbInput fullWidth id="model" placeholder="Ex: Evo X"
14         formControlName="model">
15       <!-- <div class="text-danger"
16         *ngIf="processForm.get('number').invalid && processForm.get('number').touched">
17         <div *ngIf="processForm.get('number').errors.required">*Campo obrigatório</div>
18         <div *ngIf="processForm.get('number').errors.minlength">*Caracteres insuficientes</div>
19         <div *ngIf="processForm.get('number').errors.maxlength">*Caracteres excedentes</div>
20         <div *ngIf="processForm.get('number').errors.pattern">*Campo Inválido</div>
21       </div -->
22     </div>
23   <!-- <div>{{processForm.errors|json}}</div -->
24
25
26
27
28   <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
29     <label for="name" class="label">Marca</label>
30     <input maxLength="20" fullWidth type="text" nbInput fullWidth id="name"
31       placeholder="Marca ou apelido" formControlName="name">
32     <!-- <div class="text-danger"
33       *ngIf="processForm.get('documentNumber').invalid && processForm.get('documentNumber').touched"
34       <div *ngIf="processForm.get('documentNumber').errors.required">*Campo obrigatório</div>
35       <div *ngIf="processForm.get('documentNumber').errors.pattern">*Valor Inválido</div>
36       <div *ngIf="processForm.get('documentNumber').errors.minlength">*Caracteres insuficientes</div>
37     </div -->
38

```

Figura 33 – Parte do código da estrutura do formulário em HTML do cadastro do carro

<sup>1</sup> Villalaz, Jimmy. *Repositório GitHub*. 2023. Disponível em: <[https://github.com/jvillalaz/PFC\\_RaceCarTracker.git](https://github.com/jvillalaz/PFC_RaceCarTracker.git)>.

```

TS car-form.component.ts X
src > app > pages > car-form > TS car-form.component.ts > CarFormComponent > createCar > subscribe() callback
58 private buildCarForm() {
59   this.carForm = this.formBuilder.group({
60     model: [null],
61     name: [null],
62     owner: [null],
63     plate: [null],
64     category: [null],
65   });
66 }
67
68 getCategories() {
69   this.categoryService
70     .getAll()
71     .pipe(takeUntil(this.unsubscribe$))
72     .subscribe(
73       (ans: any) => {
74         this.categories = ans;
75         console.log(this.categories);
76       },
77       (error) => {
78         alert("Não foi possível obter as categorias.");
79       }
80     );
81 }
82
83 private createCar() {
84   const newCar = {
85     name: this.carForm.get("name")?.value,
86     model: this.carForm.get("model")?.value,
87     owner: this.carForm.get("owner")?.value,
88     plate: this.carForm.get("plate")?.value,
89     category: this.carForm.get("category")?.value,
90   };
91 }
    
```

Figura 34 – Parte do código da estrutura do formulário em TS do cadastro do carro

```

TS car.service.ts X
src > app > pages > shared > services > TS car.service.ts > CarService
34 }
35
36 getCurrentCarTrack(): Observable<Track> {
37   const url = `${environment.URL_API}/track/current`;
38   return this.http
39     .get(url)
40     .pipe(catchError(this.handleError), map(this.jsonDataToTrack));
41 }
42
43 create(car: Car): Observable<Car> {
44   // car.transformer = car.transformer.id;
45   return this.http
46     .post(this.apiPath, car)
47     .pipe(catchError(this.handleError), map(this.jsonDataToCar));
48 }
49
50 update(car: FormData): Observable<Car> {
51   const url = `${this.apiPath}/${car.get("id")}`;
52   return this.http
53     .put(url, car)
54     .pipe(catchError(this.handleError), map(this.jsonDataToCar));
55 }
56
57 private jsonDataToProcesses(jsonData: any): Car[] {
58   const car: Car[] = [];
59   jsonData.forEach((element: Car) => car.push(element as Car));
60   return car;
61 }
62
63 private jsonDataToCar(jsonData: any): Car {
64   return jsonData as Car;
65 }
66
67 private jsonDataToTrack(jsonData: any): Track {
68   return jsonData[0] as Track;
    
```

Figura 35 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações do carro

```
championship-form.component.html X
src > app > pages > championship-form > < championship-form.component.html > div.d-flex.flex-column.align-items-center.text-center
1 <div class="d-flex flex-column align-items-center text-center">
2   
3   <strong>Não há campeonatos</strong>
4 </div>
```

Figura 36 – Parte do código da estrutura do formulário em HTML do campeonato

```
TS championship-form.component.ts X
src > app > pages > championship-form > TS championship-form.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-championship-form',
5   templateUrl: './championship-form.component.html',
6   styleUrls: ['./championship-form.component.scss']
7 })
8 export class ChampionshipFormComponent implements OnInit {
9
10  constructor() { }
11
12  ngOnInit(): void {
13  }
14
15 }
16
```

Figura 37 – Parte do código da estrutura do formulário em TS do campeonato

```
TS championship.service.ts X
src > app > pages > shared > services > TS championship.service.ts > ...
18 constructor(private http: HttpClient) { }
19
20 getAll(): Observable<Championship[]> {
21   return this.http
22     .get(this.apiPath)
23     .pipe(catchError(this.handleError), map(this.jsonDataToChampionships));
24 }
25
26 getById(id: number): Observable<Championship> {
27   const url = `${this.apiPath}/${id}`;
28   return this.http
29     .get(url)
30     .pipe(catchError(this.handleError), map(this.jsonDataToChampionship));
31 }
32
33 create(championship: any): Observable<Championship> {
34   // championship.transformer = championship.transformer.id;
35   return this.http
36     .post(this.apiPath, championship)
37     .pipe(catchError(this.handleError), map(this.jsonDataToChampionship));
38 }
39
40 update(championship: any): Observable<Championship> {
41   const url = `${this.apiPath}/${championship.get("id")}`;
42   return this.http
43     .patch(url, championship)
44     .pipe(catchError(this.handleError), map(this.jsonDataToChampionship));
45 }
46
47 private jsonDataToChampionships(jsonData: any): Championship[] {
48   const championship: Championship[] = [];
49   jsonData.forEach((element: Championship) => championship.push(element as Championship));
50   return championship;
51 }
```

Figura 38 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações do campeonato

```

home.component.html
src > app > pages > home > home.component.html > nb-card > nb-card-body > table.mat-elevation-z8 > ng-container > td
28 <table mat-table [dataSource]="dataSource" class="mat-elevation-z8">
29
30 <!-- Note that these columns can be defined in any order.
31      The actual rendered columns are set as a property on the row definition -->
32
33 <!-- Position Column -->
34 <ng-container matColumnDef="position">
35   <th mat-header-cell *matHeaderCellDef> Nome</th>
36   <td mat-cell *matCellDef="let element"> {{element.name}} </td>
37 </ng-container>
38
39 <!-- Name Column -->
40 <ng-container matColumnDef="name">
41   <th mat-header-cell *matHeaderCellDef> Modelo </th>
42   <td mat-cell *matCellDef="let element"> {{element.model}} </td>
43 </ng-container>
44
45 <!-- Weight Column -->
46 <ng-container matColumnDef="weight">
47   <th mat-header-cell *matHeaderCellDef> Placa </th>
48   <td mat-cell *matCellDef="let element"> {{element.plate}} </td>
49 </ng-container>
50
51 <ng-container matColumnDef="category">
52   <th mat-header-cell *matHeaderCellDef> Categoria </th>
53   <td mat-cell *matCellDef="let element"> {{element?.category.name}} </td>
54 </ng-container>
55
56 <!-- Symbol Column -->
57 <ng-container matColumnDef="symbol">
58   <th mat-header-cell *matHeaderCellDef> Proprietário </th>
59   <td mat-cell *matCellDef="let element"> {{element.owner}} </td>
60 </ng-container>
61

```

Figura 39 – Parte do código da estrutura do formulário em HTML da home

```

TS home.component.ts
src > app > pages > home > TS home.component.ts > ...
64
65 ngOnDestroy(): void {
66   this.unsubscribe$.next();
67   this.unsubscribe$.complete();
68 }
69
70 getCars(): void {
71   this.carService
72     .getAll()
73     .pipe()
74     .subscribe((ans) => {
75       console.log(ans);
76       this.dataSource = ans;
77     });
78 }
79
80 updateSort(sortRequest: any): void {
81   this.sortColumn = sortRequest.column;
82   this.sortDirection = sortRequest.direction;
83 }
84
85 getSortDirection(column: string): NbSortDirection {
86   if (this.sortColumn === column) {
87     return this.sortDirection;
88   }
89   return NbSortDirection.NONE;
90 }
91
92 getShowOn(index: number) {
93   const minWithForMultipleColumns = 400;
94   const nextColumnStep = 100;
95   return minWithForMultipleColumns + nextColumnStep * index;
96 }
97 }

```

Figura 40 – Parte do código da estrutura do formulário em TS da home

```

panel-show.component.html
src > app > pages > panel-show > panel-show.component.html > nb-card
1 <nb-card>
2   <nb-card-header class="bkg-header-yellow">Carro na Pista
3   </nb-card-header>
4   <nb-card-body>
5
6
7     <div class="d-flex flex-row justify-content-between mb-4">
8       <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
9         <label for="championship" class="label">Campeonato</label>
10        <nb-select (selectedChange)="getRoundByChampionship($event)" fullWidth placeholder="Campeona
11          <nb-option *ngFor="let championship of championships" [value]="championship_id"
12            {{championship.name}}
13          </nb-option>
14        </nb-select>
15      </div>
16
17      <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
18        <label for="championship" class="label">Baterias</label>
19        <nb-select [disabled]="rounds.length == 0" [(selected)]="selected"
20          (selectedChange)="getCarsByRound($event)" fullWidth placeholder="Bateria">
21          <nb-option *ngFor="let round of rounds" [value]="round_id"
22            {{round.name}}
23          </nb-option>
24        </nb-select>
25      </div>
26
27      <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
28        <label for="championship" class="label">Enviar Carro Para Pista:</label>
29        <nb-select [disabled]="cars.length == 0" (selectedChange)="sendCarToTrack($event)" fullWidth
30          placeholder="Carro">
31          <nb-option *ngFor="let car of cars" [value]="car_id"
32            {{car.name}}
33          </nb-option>
34        </nb-select>

```

Figura 41 – Parte do código da estrutura do formulário em HTML do painel

```

TS panel-show.component.ts
src > app > pages > panel-show > TS panel-show.component.ts > Lap > time
38
39 ngOnInit(): void {
40   this.getCurrentCar();
41   this.getChampionships();
42   this.getCars();
43   this.wsConnect();
44 }
45
46 ngOnDestroy(): void {
47   this.unsubscribe$.next();
48   this.unsubscribe$.complete();
49 }
50
51
52 getCurrentCar(): void {
53   this.carService
54     .getCurrentCarTrack()
55     .pipe(takeUntil(this.unsubscribe$))
56     .subscribe(
57       (ans: any) => {
58         this.currentCar = ans;
59         console.log(this.currentCar);
60       },
61       (error) => {
62         alert("Não há car na pista!");
63       }
64     );
65 }
66
67 getCarsByRound(event: any): void {
68   this.carService
69     .getCurrentCarTrack()
70     .pipe(takeUntil(this.unsubscribe$))
71     .subscribe(

```

Figura 42 – Parte do código da estrutura do formulário em TS do painel

```
TS category.service.ts X
src > app > pages > shared > services > TS category.service.ts > ...
8  @Injectable({
9    providedIn: "root",
10 })
11 export class CategoryService {
12   private apiPath = `${environment.URL_API}/category`;
13
14   constructor(private http: HttpClient) {}
15
16   getAll(): Observable<Category[]> {
17     return this.http
18       .get(this.apiPath)
19       .pipe(catchError(this.handleError), map(this.jsonDataToCategories));
20   }
21
22   getById(id: number): Observable<Category> {
23     const url = `${this.apiPath}/${id}`;
24     return this.http
25       .get(url)
26       .pipe(catchError(this.handleError), map(this.jsonDataToCategory));
27   }
28
29   create(category: FormData): Observable<Category> {
30     // category.transformer = category.transformer.id;
31     return this.http
32       .post(this.apiPath, category)
33       .pipe(catchError(this.handleError), map(this.jsonDataToCategory));
34   }
35
36   update(category: FormData): Observable<Category> {
37     const url = `${this.apiPath}/${category.get("id")}`;
38     return this.http
39       .put(url, category)
40       .pipe(catchError(this.handleError), map(this.jsonDataToCategory));
41   }
}
```

Figura 43 – Parte do código do serviço da categoria dos carros, arquivo do tipo TS

```
round-form.component.html X
src > app > pages > round-form > round-form.component.html > nb-card
81 <nb-card-body>
82
83
84 <form [formGroup]="roundForm" (ngSubmit)="submitForm()">
85
86   <div class="form-row">
87
88     <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
89       <label for="name" class="label">Número da bateria</label>
90       <input fullWidth type="number" nbInput fullWidth id="name" placeholder="Ex: 1"
91         formControlName="name">
92       <!-- <div class="text-danger"
93         *ngIf="processForm.get('number').invalid && processForm.get('number').touched">
94         <div *ngIf="processForm.get('number').errors.required">*Campo obrigatório</div>
95         <div *ngIf="processForm.get('number').errors.minLength">*Caracteres insuficientes</div>
96         <div *ngIf="processForm.get('number').errors.maxLength">*Caracteres excedentes</div>
97         <div *ngIf="processForm.get('number').errors.pattern">*Campo Inválido</div>
98       </div> -->
99
100     </div>
101
102
103
104
105
106     <div class="form-group resp-form-row column ml-sm-4 ml-md-4 ml-lg-4">
107       <label for="championship" class="label">Campeonato</label>
108       <nb-select fullWidth id="championship" placeholder="Campeonato" formControlName="champio
109         <nb-option *ngFor="let championship of championships" [value]="championship_id"
110           {{championship.name}}
111       </nb-option>
112     </nb-select>
113     <!-- <div class="text-danger"
114       *ngIf="processForm.get('interested').invalid && processForm.get('interested').touch
```

Figura 44 – Parte do código da estrutura do formulário em HTML da bateria



```
TS round-form.components.ts X
src > app > pages > round-form > TS round-form.components.ts > ...
54 }
55
56 submitChampionshipForm() {
57   this.createChampionship();
58 }
59
60 private setCurrentAction() {
61   // if (this.route.snapshot.url[0].path == "new") this.currentAction = "new";
62   // else this.currentAction = "edit";
63 }
64
65 private buildRoundForm() {
66   this.roundForm = this.formBuilder.group({
67     name: [null],
68     championship: [null],
69   });
70 }
71
72 private buildChampionshipForm() {
73   this.championshipForm = this.formBuilder.group({
74     name: [null],
75   });
76 }
77
78 getChampionships() {
79   this.championshipService
80     .getAll()
81     .pipe(takeUntil(this.unsubscribe$))
82     .subscribe(
83       (ans: any) => {
84         this.championships = ans;
85         console.log(this.championships);
86       },
87       (error) => {
88         alert("Não foi possível obter os campeonatos.");
89       }
89     );
90 }
```

Figura 45 – Parte do código da estrutura do formulário em TS da bateria

```
TS round.service.ts X
src > app > pages > shared > services > TS round.service.ts > ...
 8  @Injectable({
 9    providedIn: "root",
10  })
11  export class RoundService {
12    private apiPath = `${environment.URL_API}/rounds`;
13    private apiPathTrack = `${environment.URL_API}/track`;
14    constructor(private http: HttpClient) { }
15
16
17
18    sendCarTrack(car: any): Observable<Round> {
19      // round.transformer = round.transformer.id;
20      return this.http
21        .post(this.apiPathTrack, car)
22        .pipe(catchError(this.handleError), map(this.jsonDataToRound));
23    }
24
25    getAll(): Observable<Round[]> {
26      return this.http
27        .get(this.apiPath)
28        .pipe(catchError(this.handleError), map(this.jsonDataToRounds));
29    }
30
31    getById(id: string): Observable<Round> {
32      const url = `${this.apiPath}/${id}`;
33      return this.http
34        .get(url)
35        .pipe(catchError(this.handleError), map(this.jsonDataToRound));
36    }
37
38    getRoundsByChampionship(id: number): Observable<Round> {
39      const url = `${this.apiPath}/byChampionship/${id}`;
40      return this.http
41        .get(url)
```

Figura 46 – Parte do código do serviço em TS se comunicando com a API através de requisições para conseguir informações da bateria atual, os carros e os tempos ordenados

```
Extension: Python  main.py M X
main.py
18
19 def getCurrent():
20     global currentCar
21     print('Buscando carro!')
22     req = urequests.get(URL_API + "/track/current")
23     if req.status_code == 200:
24         res = req.json()
25         currentCar = {
26             "car": res[0]["car"]["_id"],
27             "round": res[0]["round"]["_id"]
28         }
29         carLed.value(1)
30         print(currentCar)
31
32
33 def registerLapTime(lapTime, currentCar):
34     global counterLap
35     lap = currentCar.copy()
36     lap["time"] = lapTime
37     req = urequests.post(URL_API + "/laptime/registerLap", json=lap)
38     if req.status_code != 201:
39         registerLapTime(lapTime, currentCar)
40     else:
41         print(req.json())
42         if counterLap == 5:
43             currentCar = dict
44             carLed.value(0)
45
46
47
48 def handle_interrupt(pin):
49     global counterLap
50     global initialLapTime
51     global motion
```

Figura 47 – Parte do código embarcado na placa para a medição dos tempos de volta

---

## REFERÊNCIAS

ADAFruit. *PIR Motion Sensor DataFile*. 2022. Disponível em: <<https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>>. 32

AKBAS, S.; EFE, M. A.; OZDEMIR, S. Performance evaluation of pir sensor deployment in critical area surveillance networks. p. 327–332, 2014. 31

Angular Community. *Understanding Angular*. 2023. Disponível em: <<https://angular.io/guide>>. 24

BORGES, H.; HORA, A.; VALENTE, M. T. Understanding the factors that impact the popularity of github repositories. *IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, USA*, p. 334–344, 2016. 22

DASHEVSKY, I.; BALZANO, V. James webb space telescope ground to flight interface design. *2008 IEEE Aerospace Conference, IEEE*, v. 1, n. 7, p. 100–120, 2008. 22

DOU, R.; NAN, G. Optimizing sensor network coverage and regional connectivity in industrial iot systems. *IEEE Systems Journal*, v. 11, n. 3, p. 1351–1360, 2017. 31

DOUGLAS, K.; DOUGLAS, S. . *PostgreSQL: a comprehensive guide to building, programming, and administering PostgreSQL databases*. 1. ed.. ed. São Paulo: SAMS publishing, 2003. v. 1. ed. 27

Espressif Systems. *Especificações ESP32-WROOM*. 2023. Disponível em: <[https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e\\_esp32-wroom-32ue\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf)>. 29

GAMI, H. Movement direction and distance classification using a single pir sensor. *IEEE Sensors Letters*, v. 2, n. 1, p. 1–4, 2018. 31

GEETHA, G. et al. Interpretation and analysis of angular framework. *International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India*, p. 1–6, 2022. 25

GÜVEN, Y. et al. Understanding the concept of microcontroller based systems to choose the best hardware for applications. *Research Inventy: International Journal of Engineering and Science*, v. 7, n. 9, p. 38–44, 2017. 27

JAMSHED, M. A. et al. Challenges, applications, and future of wireless sensors in internet of things: A review. *IEEE Sensors Journal*, v. 22, n. 6, p. 5482–5494, 2022. 31

- JENKINS, M. Technological discontinuities and competitive advantage: A historical perspective on formula 1 motor racing 1950–2006. *International Journal of Management Studies*, v. 47, n. 1, p. 884–910, 2010. 18
- KELLY, D. P.; SHARP, R. S. Time-optimal control of the race car: a numerical method to emulate the ideal driver. *International Journal of Vehicle Mechanics and Mobility*, v. 48, n. 12, p. 1461–1474, 2010. 18
- LEI, K.; MA, Y.; TAN, Z. Performance comparison and evaluation of web development technologies in php, python, and node.js. *IEEE 17th International Conference on Computational Science and Engineering, Chengdu, China*, p. 661–668, 2014. 26
- LOT, R.; BIANCO, N. D. Lap time optimisation of a racing go-kart, vehicle system dynamics. *International Journal of Vehicle Mechanics and Mobility*, v. 54, n. 2, p. 210–230, 2016. 18
- MACHESO, P. et al. Design of standalone asynchronous esp32 web-server for temperature and humidity monitoring. *7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, p. 635–638, 2021. 21
- MAIER, A.; SHARP, A.; VAGAPOV, Y. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. *2017 Internet Technologies and Applications (ITA), Wrexham, UK*, p. 143–148, 2017. 29
- MAT, I. et al. Iot in precision agriculture applications using wireless moisture sensor network. p. 24–29, 2016. 31
- MITROVIĆ, N. et al. Implementation of websockets in esp32 based iot systems. *15th International Conference on Advanced Technologies, Systems and Service in Telecommunications (TELSIKS)*, p. 261–264, 2021. 21
- Node Package Manager (NPM). NPM. 2023. Disponível em: <<https://github.com/npm/npm>>. 26
- Regulamento Hot Lap da Villa. *Regulamento do evento Hot Lap da Villa*. 2023. Disponível em: <<https://drive.google.com/file/d/1IBnbgizZ76509UXRzrecmkqCZptOC05U/view>>. 17, 34
- SEHRAWAT, D.; GILL, N. S. Smart sensors: Analysis of different types of iot sensors. *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India*, p. 523–528, 2019. 30
- SUN, H. et al. Efficient dynamic analysis for node.js. *International Conference on Compiler Construction*, v. 18, n. 11, p. 134–141, 2018. 25
- SUNARDI, A.; SUHARJITO, J. Mvc architecture: A comparative study between laravel framework and slim framework in freelancer project monitoring system web based. *Procedia Computer Science*, v. 157, n. 1, p. 134–141, 2019. 23
- The PostgreSQL Global Development Group. *PostgreSQL Documentation*. 2023. Disponível em: <<https://www.postgresql.org/docs/>>. 26
- TILKOV, S.; VINOSKI, S. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, v. 14, n. 6, p. 80–83, 2010. 25

ULLO, S. L.; SINHA, G. R. Advances in smart environment monitoring systems using iot and sensors. *MDPI Sensors Journal*, v. 20, n. 11, p. 3113, 2020. 31

Villalaz, Jimmy. *Repositório GitHub*. 2023. Disponível em: <[https://github.com/jvillalaz/PFC\\_RaceCarTracker.git](https://github.com/jvillalaz/PFC_RaceCarTracker.git)>. 65

WESSELBAUM, D.; OWEN, P. The value of pole position in formula 1 history. *The Australian Economic Review*, v. 54, n. 1, p. 164–173, 2021. 18

WIRFS-BROCK, A.; EICH, B. Javascript: the first 20 years. *Proceedings of the ACM on Programming Languages and History of Programming Languages*, v. 4, n. 77, 2020. 23

YUN, J.; SONG, M.-H. Detecting direction of movement using pyroelectric infrared sensors. *IEEE Sensors Journal*, v. 14, n. 5, p. 1482–1489, 2014. 31