VITOR MORILHA DUARTE E PEDRO GALICIA DE LIMA

PROJETO E IMPLEMENTAÇÃO DE UMA INTERFACE DE MONITORAÇÃO E CONTROLE REMOTO DE UM VEÍCULO AUTOMOTOR

Monografia apresentada a Escola Politécnica da Universidade de São Paulo para a conclusão de curso.

São Paulo
2022

VITOR MORILHA DUARTE E PEDRO GALICIA DE LIMA

PROJETO E IMPLEMENTAÇÃO DE UMA INTERFACE DE MONITORAÇÃO E CONTROLE REMOTO DE UM VEÍCULO AUTOMOTOR

Monografia apresentada a Escola Politécnica da Universidade de São Paulo para a conclusão de curso.

Curso de Graduação:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Fabrício Junqueira

São Paulo
2022

Catalogação-na-publicação

Morilha Duarte, Vitor; Galicia de Lima, Pedro.
Projeto e implementação de uma interface de monitoração e controle remoto de um veículo automotor / Morilha Duarte, Vitor; Galicia de Lima, Pedro. – São Paulo, 2022

Monografia (Graduação em Engenharia Mecatrônica) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR).

1. Sistema ciber-físico. 2. Internet das coisas. 3. Cloud computing. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos (PMR).

AGRADECIMENTOS

Ao nosso orientador Prof. Dr. Fabrício Junqueira pela sua constante orientação, paciência durante o período da realização do trabalho e pela ajuda nos momentos difíceis.

Aos professores da Escola Politécnica da USP, por terem nos passado seu conhecimento e nos ensinado a enfrentar desafios.

Aos nossos familiares pelo apoio recebido.

A todos aqueles que contribuíram direta ou indiretamente na produção deste trabalho.

RESUMO

O presente trabalho aborda um veículo de motor à combustão como um sistema ciber-físico (CPS). Para isso, é necessário entender não só aspectos fundamentais do funcionamento de motores a combustão e a comunicação dentro de um veículo via CAN, mas também aspectos extremamente novos no campo da engenharia, como CPS.

Um CPS relaciona-se intimamente com conceitos basilares da Indústria 4.0 como: Internet das Coisas, Computação em nuvem, *BigData*, etc. Em suma esse tipo de sistema coleta informações do ambiente e é capaz de se adaptar de acordo com essas informações.

Ao estudar a abordagem de um veículo de motor a combustão como CPS, foi possível entender a vastidão de possibilidades e aplicações deste conceito nos veículos. Neste trabalho, o foco foi propor uma interface de monitoração e controle remoto de um veículo visando a melhora do consumo de combustível.

A fim de validar a proposta foi construído um modelo de veículo conectado à internet enviando seus dados a uma nuvem. Auxiliado por um *dashboard* desse provedor de nuvem, pôde-se estabelecer um fluxo de monitoração e controle.

ABSTRACT

The present work seeks to approach a combustion engine vehicle with a cyber-physical system (CPS). For this, it is necessary to understand not only fundamental aspects of the functioning of combustion engines and communication within a vehicle via CAN, but also extremely new aspects in the field of engineering, such as CPS.

A CPS is closely related to core industry 4.0 concepts such as: Internet of Things, Cloud Computing, BigData, etc. In short, this type of system will collect information from the environment and will be able to adapt according to this information collected.

By studying the approach of a combustion engine vehicle as a CPS, it was possible to understand the vastness of possibilities and applications of this concept in vehicles. In this work, the focus was to propose an interface for monitoring and remote control of a vehicle in order to improve fuel consumption.

In order to validate the proposal, a vehicle model connected to the internet was built, sending its data to a cloud. Aided by a dashboard from this cloud provider, a monitoring and control flow could be established.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

A 4ª Revolução Industrial inaugurou dois importantes movimentos, a diminuição das dimensões de componentes eletrônicos e a melhora da comunicação. Com o passar dos anos, *hardwares* que antes tomavam muito espaço e possuíam um elevado valor agregado, tornam-se cada vez menores, com capacidade de processamento maior e menor custo. Além disso, para Rawung e Putrada (2014) a internet facilita a troca de informações através do mundo digital, diminuindo drasticamente restrições de hora e local.

Historicamente, sempre houve a perspectiva de se atrelar cada vez mais um veículo a sistemas cibernéticos. A revista *Automotive Engineering* (1989), por exemplo, previa a união entre sistemas inteligentes e automóveis. Na época não havia tecnologia suficiente para expandir o assunto e trazer soluções, mas hoje já é possível refinar a proposta.

Um sistema ciber físico (CPS) abarca os conceitos de comunicação, computação e armazenamento combinados, com intuito de monitorar ou controlar entidades do mundo físico. O trabalho, portanto, unirá tecnologias já existentes a fim de que a abordagem seja possível (SHI e WAN, 2011).

Do ponto de vista das fábricas, a abordagem permite que elas obtenham informações de seus produtos em tempo real e possam entender como eles se comportam ao longo do tempo, oferecendo cada vez mais produtos confiáveis e, inclusive, prever falhas.

Ao usuário, maior conforto, confiabilidade e conhecimento sobre suas máquinas não são as únicas vantagens da abordagem. Conhecendo a forma com que o motor se comporta, ele será capaz de ter maior autonomia na direção. Outro ponto forte de encarar um veículo como CPS é a possibilidade de melhorar o seu consumo e o seu comportamento técnico diante de diferentes situações climáticas e geográficas.

Para o proprietário, a eficiência energética vem a ser um relevante fator econômico devido à redução da demanda por combustível. Já do ponto de vista social, a redução das emissões de gases tóxicos proporciona um ambiente mais agradável e saudável.

1.1. Objetivo do trabalho

O objetivo do presente trabalho é demonstrar a viabilidade de realizar o monitoramento e controle remoto de um veículo de motor a combustão a fim de dar escalabilidade e conhecimento do comportamento do veículo sob diversos ambientes e motoristas distintos. Para isso, foi decidido abordá-lo como um CPS.

O veículo coletará dados de seus componentes eletrônicos e enviará a uma base remota para que observadores ao longe tenham conhecimento do seu desempenho técnico. Isso porque um armazenamento fora do veículo permite a agregação de informações de diversos veículos e preserva a integridade física da base de dados em caso de acidentes. Esse fato justifica a abordagem a medida em que fornecerá ao usuário de um determinado veículo conhecimento sobre o comportamento dele diante de diversos climas, pressões atmosféricas, terrenos e formas de direção.

Com esse tipo de informação, pode-se partir para o próximo estágio de trabalho, que permitirá a alteração de parâmetros em tempo real. Ao final, será possível uma comunicação bidirecional, lendo e escrevendo dados do veículo auxiliados por um microcontrolador. Esse modelo de fluxo de informação aplicado em veículos da indústria pode ser usado para prever falhas e denunciar um mau uso do equipamento.

1.2. Requisitos

Para a implementação da solução proposta, é imperativo que o veículo esteja em uma zona de cobertura com sinal de internet. Atualmente isso ainda restringe o projeto, porém o advento do 5G e outras tecnologias que vêm tendo apoio massivo da indústria (como *starlink*) tendem a ampliar essa zona para o mundo todo.

Será necessário que se realize uma simulação de forma simples que gerará dados compatíveis com os de um veículo a fim de que se cumpra o mesmo papel. Sob a ótica deste trabalho, o foco não é a origem, mas sim a transmissão desses dados.

Como processamento será utilizado um *hardware* que consiga entender as informações transmitidas pelo mesmo protocolo já utilizado em veículos da indústria, o *Controller Area Network* (CAN), e como o intuito é que seja feita a distância, o *hardware* precisa ter capacidade de longo alcance, já que ele ficará junto ao veículo.

A comunicação à distância pode ser feita através de sinais de rádio ou também pela internet.

Avaliando uma aplicação comercial, os dados precisam ser enviados para um servidor. A base de dados abrigada nesse servidor deve abarcar a possibilidade de escalabilidade, ser robusta e altamente confiável a ponto de receber um grande volume de dados sem prejudicar o seu armazenamento. Para tudo isso, uma solução comercial que já foi testada e é aplicada em massa é mais indicada a fim de atender os requisitos citados. Atualmente, esse tipo de serviço já é oferecido por grandes empresas e será discutido alguns deles ao longo do trabalho.

Uma vez disponíveis, os dados do veículo ficarão à disposição de outras pessoas, longe do veículo. Para esse observador, será necessário um computador ou celular com acesso à internet e assim deverá monitorar e também alterar parâmetros via web. Seu papel não é só acompanhar os valores de variáveis, mas também alterar parâmetros a fim de tornar a experiência do motorista mais agradável. Como o intuito do projeto é que seja possível saber em tempo real as informações do veículo, a latência de resposta deve ser de modo a permitir ajustes de parâmetros do veículo sem prejudicar seu desempenho. No futuro, um algoritmo poderia ser utilizado para analisar a base de dados coletada ao longo do tempo e realizar a alteração desses parâmetros de forma automática.

1.3. Organização do trabalho

Este trabalho foi organizado em quatro grandes capítulos. O capítulo 2 busca se aprofundar nos conceitos utilizados, sem os quais não seria possível o projeto. Dessa forma o estado da arte e os trabalhos acadêmicos relacionados ao tema deste trabalho serviram de suplemento para a elaboração da proposta desenvolvida. O capítulo 3 é a proposta em si, e trata sobre a arquitetura da solução proposta.

No capítulo 4, a implementação do sistema proposto é explicada, com todos os *hardwares* e *softwares* utilizados que melhor se adequaram à proposta. Por fim, o capítulo 5 trata dos testes que foram realizados ao longo de todo o andamento do trabalho, desde os testes iniciais até a implementação final.

Por fim, o capítulo 6 encerra o trabalho com as conclusões obtidas ao longo do desenvolvimento do projeto e propõe a realização de trabalhos futuros.

2. Revisão Bibliográfica

São apresentados nessa seção os conceitos necessários e como a literatura os apresenta. Inicialmente trata-se das diferentes formas de definir um CPS, seus desafios e seus objetivos. Depois disso, torna-se necessário demonstrar como as pesquisas modernas realizaram a aplicação dos conceitos de CPS em diferentes áreas. Para finalizar, traz referências nas áreas de monitoramento remoto e funcionamento de componentes eletrônicos de um motor a combustão.

2.1. Sistema ciber-físico

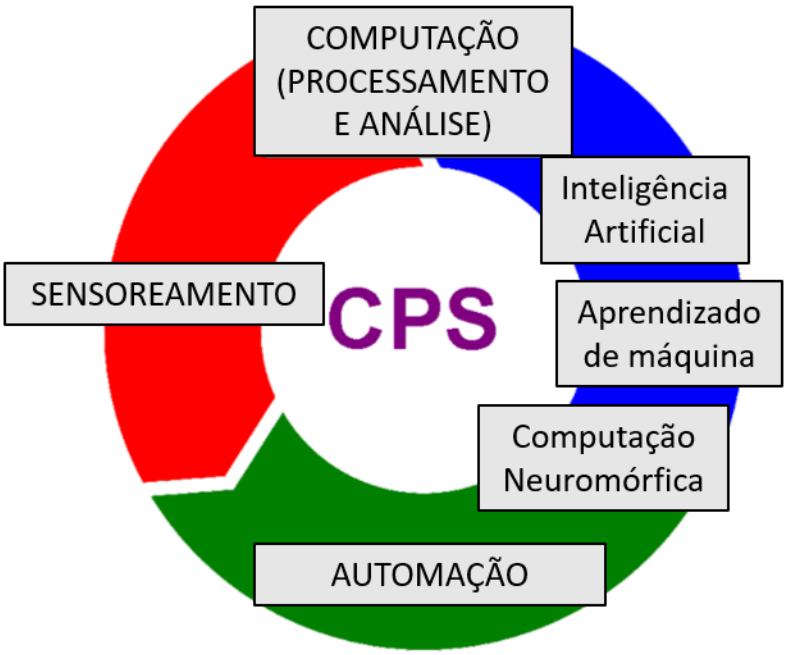
O fato de ser um conceito muito novo dificulta encontrar uma definição. Por esse motivo, duas diferentes definições complementares entre si serão apresentadas.

Para definir esse sistema, inicia-se com análise daquilo que ele não é. O CPS não é um *Personal Computer* (PC), não é um sistema embarcado e não é uma rede de sensores. Sua peculiaridade está mais ligada ao mundo físico, o que traz consigo restrições de controle (malha fechada) e um desafio de relacionar o mundo real, que ocorre em paralelo, e um processo computacional sequencial. O sistema, portanto, pode ser caracterizado por um ou mais dispositivos que detectam o ambiente e, integrado a habilidades computacionais, executam ações do mundo físico ou se comunicam com outros sistemas, permitindo cooperação e interação em tempo real (RAWUNG e PUTRADA, 2014) e (WANG e ZHANG, 2016).

Já Shi e Wan (2011) adotam a definição mais conceitual. Eles dizem: “CPS integra dinâmicas de processos físicos com *softwares* e comunicação, usando abstrações, modelagem e técnicas de análises para integrar uma rede”. O CPS, portanto, combina conceitos de computação, comunicação e sistemas embarcados. Shi e Wan (2011) ainda acrescentam com algumas características: forte integração, capacidade cibernética em todos os componentes físicos, interligação em forte escala, alta automação, entre outros.

Um CPS, portanto, oferece os conceitos de comunicação, computação e armazenamento combinados, com intuito de monitorar ou controlar entidades do mundo físico. É claro que um sistema desse tipo requer vasto conhecimento e habilidade multidisciplinar e a Figura 1 exemplifica essa interação por meio de um ciclo. Os requisitos educacionais fundamentais para o desenvolvedor de um CPS são: ciência da computação, modelagem, colaboração, coordenação de projetos, entre outros.

Figura 1: Ciclo de automação do CPS



Fonte: Traduzido de Atat *et al* (2018)

2.1.1. Arquitetura e desenvolvimento

Do ponto de vista da arquitetura, sistemas digitais em forma de rede, que interagem tanto com o mundo físico quanto com o mundo virtual, organizam-se com uma hierarquia baseada em nós. Cada nó pode ser um CPS e conter uma ou mais funções como controle, deteção, execução e computação. Sua estrutura é baseada no modelo de árvore formando camadas, em que a comunicação é dada em um sentido ou em ambos. A comunicação em um sentido só recebe ou transmite informações. Já a comunicação em ambos os sentidos recebe e transmite a fim de que cada pequeno sistema inteligente seja capaz de colaborar e atingir um objetivo comum (RAWUNG E PUTRADA, 2014; WANG E ZHANG, 2016).

2.1.2. Conceitos relacionados

Não é incomum encontrar na literatura estudos e exemplos de aplicações que intersectam os conceitos de IoT com CPS. Para Mäkiö-Marusik e Ahmad (2018), os CPSs fazem parte da internet das coisas e podem ser desenvolvidos sem limites pela indústria, agricultura e, também, na área da saúde. Já para Atat e Liu (2018), o IoT atua como uma ponte para conectar à rede de diferentes sistemas cyber-físicos.

Para o desenvolvimento de um CPS, Shi e Wan (2011) apontam, além do IoT, outros conceitos que não podem ser deixados de lado. É necessária, também, a

familiaridade com novas tecnologias. Entre elas, as mais comuns são Big Data, Análise em Tempo Real (RTA), Computação em Nuvem, entre outras. Isso aumenta a complexidade do desenvolvimento do sistema, mas acrescenta confiabilidade (Leitão e Colombo, 2016). A partir disso, segue a listagem desses conceitos que, sem os quais, não é possível progredir no estudo dos CPSs:

- **Controle de energia gasta:** Um grande desafio do dispositivo CPS é o suprimento de energia. Diversas soluções são propostas, Parolini e Toliaz (2010), por exemplo, desenvolvem um modelo orientado ao controle da dinâmica cibernética. As pesquisas elaboram uma estratégia de controle que permite maior inteligência na dualidade entre satisfazer um comando e economizar energia. Já Tang e Gupta (2008) propõem que a minimização da temperatura do sistema ciberfísico é a melhor forma de melhorar a eficiência energética;
- **Transmissão e gerenciamento:** As informações de diferentes sensores precisam ser transmitidas e tratadas. Kong (2010) propõe duas perguntas. A primeira é como o sistema reagirá às mudanças sensíveis do ambiente. Segundo, com a variação do ambiente, o quão a ação adaptativa vai se relacionar e alterar o mesmo. Assim, ele busca estudar a distribuição espaço-temporal entre os nós de um CPS para entender qual seria a abstração ideal. Sua pesquisa abriu espaço para uma solução de distribuição espacial e temporal dos nós (KONG E JIANG, 2010);
- **Projeto de *software*:** Alguns pesquisadores desenvolvem modelos de *software* para CPS baseados em: modelos de eventos, modelos físicos e confiabilidade em tempo real. O primeiro captura a informação essencial em um acontecimento. Tan (2010) propõe um modelo de grade, que captura informações dos eventos e permite que diferentes componentes cooperem a fim de construir uma leitura cibernética do evento físico. Já em sistemas mecânicos, Zhu expõe como modelos analíticos de sistemas físicos podem ser mapeados automaticamente (Zhu e Westbrook, 2010). Por último, Kremer (2013) afirma que o tempo tem um papel fundamental nos requisitos para o projeto de um CPS. Além disso, problemas como incertezas na rede, atraso e perda de pacotes são abordados por Koutsoukos e Kottenstette (2008) com uma arquitetura de controle passivo.

2.1.3. Técnicas de controle

Sistemas de controle para CPSs ainda estão em fase inicial, e por se tratarem de sistemas dinâmicos complexos, é necessário que o controle tenha alto grau de confiança e segurança. Assim, Kottenstette e Karsai (2009) descrevem um modelo teórico baseado no conceito de passividade. Os autores afirmam que isso oferece uma direção promissora para os controladores robustos que CPSs necessitam.

2.1.4. Desafios

Além de compor definições e possibilidades, é importante que um estudo do CPS contenha os desafios que serão encontrados. Shi e Wan (2011), por exemplo, apontam seis desafios que a comunidade científica terá que discutir:

- Sistemas híbridos e seu controle: necessidade de uma nova teoria matemática que mescle sistemas baseados em eventos com sistemas baseados no tempo;
- Sensores e redes móveis: necessidade de aumentar a autonomia e a organização de um sistema para com a rede móvel, bem como definir informações prioritárias e essenciais;
- Robustez, confiabilidade e segurança: a incerteza do ambiente, ataques cibernéticos e erros em dispositivos físicos aparecem aqui como preocupações relevantes;
- Abstrações: novos métodos de computação e comunicação em tempo real são necessários;
- Desenvolvimento de um modelo base: um modelo base de CPS ainda não existe;
- Verificação, validação e certificação: uma interação entre métodos formais e testes precisa ser estabelecida.

2.1.5. Aplicações

Na literatura, o CPS aparece sumariamente com 3 aplicações: na área da indústria, hospitalar e dos transportes. No que tange à indústria, o CPS é aplicado em produções colaborativas, integração de projetos e estruturas com enfoque na otimização da produção e no monitoramento de equipamentos e máquinas (Leitão e Colombo, 2016). Nos hospitais, dispositivos médicos que se conectam com uma central podem ser desenvolvidos para fornecer informações a respeito da saúde dos pacientes. Nos transportes, o principal símbolo da aplicação do CPS são os veículos cooperativos (CVS) que usam redes sem fio para disseminar informações do mundo físico e as compartilhar com outros veículos (RAWUNG e PUTRADA, 2014).

Ao abordar um veículo como um CPS, sistemas que oferecem facilidades e comodidade para o motorista estão na vanguarda do assunto. Zhao e Li (2013), por exemplo, sugerem que a abordagem como CPS pode superar alguns desafios como a limitação da interatividade do usuário com o veículo, que se dá pelos painéis de controle. Além disso, o sistema pode passar a tomar decisões que vão de acordo com o perfil e com a vontade do usuário sem ter a necessidade de ele fazer alguma escolha, acabando assim com a descontinuidade que ocorre ao ter que recorrer ao usuário para a tomada de decisão. Ainda nesse artigo, os autores desenvolvem um controle de gestos para auxiliar nas ações do veículo.

Diversos nomes foram cunhados para se referir a um veículo que tem por trás um CPS, mas todos eles possuem como objetivos integrar pessoas, veículos e ciberespaços (Zhao e Li, 2013). Neste trabalho, optou-se por usar a expressão de Liu e Xie (2019), que se refere a um veículo ciber-físico como Sistema Ciber Físico Automotivo (ACPS). Ele conta com sistema embarcado, uma ou mais unidades de controle eletrônico (ECUs), sensores, atuadores, e estabelece comunicação por meio de barramentos.

2.2. Big Data e Real Time Analytics

No que tange à coleta de dados do ambiente proveniente de uma rede de sistemas inteligentes, é natural que se gere uma quantidade de dados que precisam ser tratados. Para Atat e Liu (2018) a formação de um ciclo de automação, combinando inteligência artificial, aprendizado de máquina, mineração de dados e computação em nuvem ajudam o armazenamento, processamento e análise de dados coletados. Isso permite que a capacidade de sensoriamento de um único nó vá além de suas capacidades e possa reduzir a latência e o consumo energético do sistema.

As tecnologias supracitadas abrem espaço para que a decisão desse tipo de sistema se aproxime cada vez mais do conceito de instantaneidade. É isso que o *real time analytics* (RTA) busca alcançar. Em seu trabalho, Trinks e Felden (2017) deixaram um Estado da Arte que resume os conceitos e parametriza algumas ideias que precisam ser consideradas ao falar de RTA. Inicialmente é importante seguir algumas definições trazidas para que não haja confusão de conceitos. Os autores diferenciam *Analysis* de *Analytics*. O primeiro “lida com uma investigação sistemática de um objeto observado”, o segundo “descreve a teoria de *analysis* e é usada como reserva de todos os métodos analíticos existentes”. Não obstante, ele ainda oferece a

Tabela 1 que traz uma classificação das expressões e termos corretos do seu objeto de estudo.

Tabela 1: Definições de RTA de Trinks e Felden (2017)

Termo	Descrição
Advanced Analytics	Combina métodos de aprendizado de máquina e estatística com o objetivo de derivar modelos de previsão.
Big Data Analytics	Identifica as técnicas avançadas de análise usadas na área de Big Data
Business Analytics	Descreve a investigação contínua de dados de negócios orientados para o passado com o objetivo de obter conhecimento sobre atividades de negócios passadas e futuras
Descriptive Analytics	Esta é uma análise descritiva que visa descrever e resumir um evento. Para esse fim, as ferramentas do relatório ou OLAP (*Online Analytical Processing*) podem ser usadas
Diagnostic Analytics	Uma análise causal, que tem como objetivo responder à pergunta "por que algo aconteceu". É sobre a coleta de conhecimento para tirar conclusões sobre a solução do problema
In-Memory Analytics	Este termo implica que os dados para processamento e análise sejam lidos diretamente da memória central. O acesso ao disco rígido não é necessário na área do In-Memory Analytics
Predictive Analytics	Caracteriza uma análise focada no futuro com o objetivo de revelar oportunidades
Prescriptive Analytics	Descreve uma análise prescritiva que gera uma recomendação para o alcance de determinadas metas de negócios. É a forma mais nobre de apoio à decisão
Real Time Analytics	Quando o tempo entre o evento acionador e a resposta necessária é muito baixo, é falado RTA. Se o tempo total de execução for zero, isso é chamado de aplicativo em tempo real

Fonte: Trinks and Felden (2017)

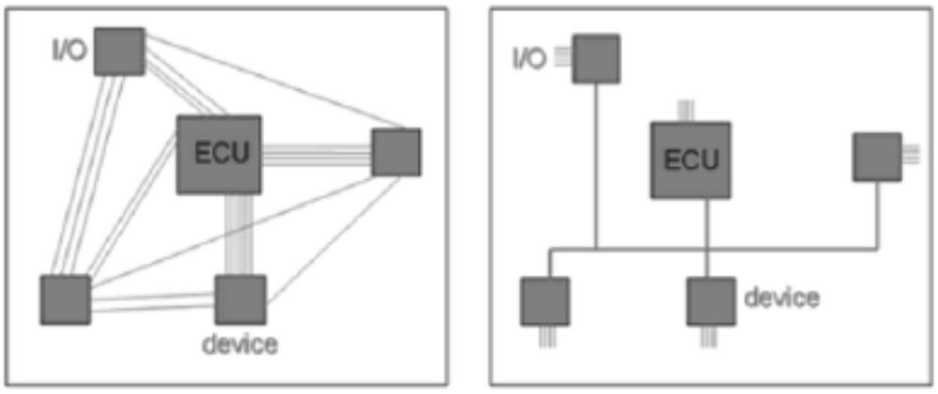
Sob seu ponto de vista, acrescentar a expressão *real time*, ou tempo real, ainda não é apropriado, pois o tempo entre o evento acionador e a resposta deve ser quase zero para valer a expressão completa, isso devido aos desafios da latência. Os autores descrevem quatro diferentes desafios de latência em um sistema dito de “tempo real”. O primeiro desafio é a latência da provisão dos dados, relacionado ao tempo que os sensores captam informações. O segundo é a latência relacionada à análise dos dados coletados, isto é, o tempo que demora para um sistema computacional processar os dados oriundos dos sensores. O terceiro é o tempo de tomada de decisão. Nesse momento, o sistema define aquilo que deverá ou não ser feito. E por último, há a latência de implementação, que se relaciona com a ação da decisão (TRINKS e FELDEN, 2017).

Para contornar os diversos percalços da latência, Trinks e Felden (2017) apresentam métodos de processar e armazenar os dados de interesse. O *Streming Anaytics*, por exemplo, é um método que não requer memória interna no dispositivo. Isso aumenta consideravelmente a velocidade no processamento, mas não permite que dados anteriores sejam reaproveitados em outro momento.

2.3. Rede CAN em veículos

A comunicação é feita por protocolo CAN (*Controller Area Network*). Esse protocolo surgiu em 1983 de uma iniciativa integrada da Mercedes-Benz com a Bosch. A tecnologia avançou e passou a ser aplicada em veículos de passeio em 1992, na época sua maior funcionalidade era a redução de fios no sistema do veículo, conforme a Figura 2 (ANDRADE, 2014).

Figura 2: Dispositivos I/O em veículo sem e com rede CAN

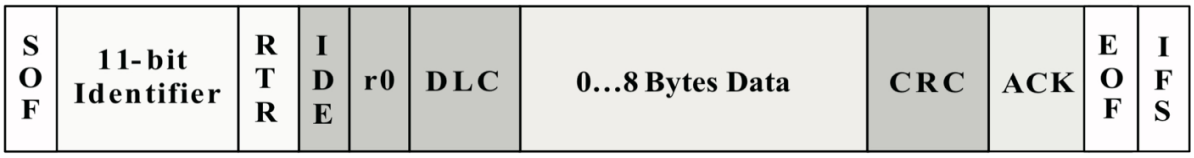


Fonte: Liu et al, 2019

O padrão de barramentos é serial e possui uma taxa máxima de sinalização é de 1 megabit por segundo (bps) com transmissão multi-mestre em que cada nó pode receber ou enviar informação, mas não simultaneamente (Naik e Kumbi, 2017). Ela permite monitorar variáveis como: velocidade, rotação do motor, consumo de combustível, temperatura do motor, emissões, e muitas outras. Essas informações são enviadas a toda rede, fornecendo consistência de dados em todos os nós do sistema.

O CAN utiliza uma comparação de ID's binários (cada ID é um módulo eletrônico) para definir a prioridade de envio da mensagem. Mensagens de alta prioridade possuem valores de ID menores, ou seja, o bit 0 é dominante e o bit 1 é recessivo (Andrade, 2014). A Figura 3 mostra o significado de uma mensagem CAN bit a bit, e a seguir é apresentado o significado de cada um de seus segmentos conforme Corrigan (2016).

Figura 3: Campos dos bits do protocolo CAN



Fonte: Corrigan, 2016

- SOF (*start of frame*): Apenas um bit usado para marcar o início da mensagem e sincronizar os nós no barramento;
- Identificador: Como já citado, estabelece a prioridade da mensagem;
- RTR (*remote transmission request*): Apenas um bit que será dominante quando são necessárias informações de outro nó. Apesar de o identificador determinar o nó de destino, todos recebem;
- IDE (*identifier extension*): Um único bit que o CAN comum sem extensões está sendo transmitido;
- r0: Bit reservado;
- DLC (*data length code*): 4 bits que contém o número de *bytes* do dado a ser transmitido;
- Dados: Até 64 bits da aplicação podem ser transmitidos;
- CRC (*cyclic redundancy check*): 15 bits que contém o número de bits transmitidos para que não haja erros;

- ACK (*acknowledges*): Cada nó reconhece a integridade de seus dados através do ACK de 2 bits, sendo um de reconhecimento, e outro um delimitador;
- EOF (*end of frame*): 7 bits que marcam o fim da mensagem CAN;
- IFS (*interframe space*): 7 bits que contém o tempo necessário para o controlador mover um *frame* recebido corretamente para sua posição correta em uma área de *buffer* de mensagem;

Do ponto de visa dos barramentos, o CAN se mostra eficiente pelo seu método construtivo. Ele consiste em dois fios trançados percorridos por correntes de mesmo módulo e sentidos opostos. Esse aspecto permite que os campos magnéticos gerados por essas correntes se cancelem. Somado a isso, o uso de receptores diferenciais atribui ao barramento uma alta imunidade a ruídos (CORRIGAN, 2016).

“As especificações do padrão ISO11898 de alta velocidade são fornecidas para uma taxa de sinalização máxima de 1 Mbps com um comprimento de barramento de 40m e um máximo de 30 nós. Também recomenda um comprimento máximo de ponta não terminada de 0,3 m. O cabo é especificado para ser um par trançado blindado ou não blindado com uma impedância característica de 120 Ω (Zo)” (CORRIGAN, 2016).

Ao aliar essa tecnologia robusta com a ascensão do CPS, um ACPS pode usar o protocolo CAN para coletar essas mensagens curtas e confiáveis para monitorar o desempenho do motor, das peças, proporcionar uma vigilância remota do veículo, elaborar estratégias para reduzir o consumo e até enviar essas informações para um servidor (ZHAO e LI, 2013), (LIU e XIE, 2019) e (HU, WANG e TANG, 2017).

2.4. Motor à combustão

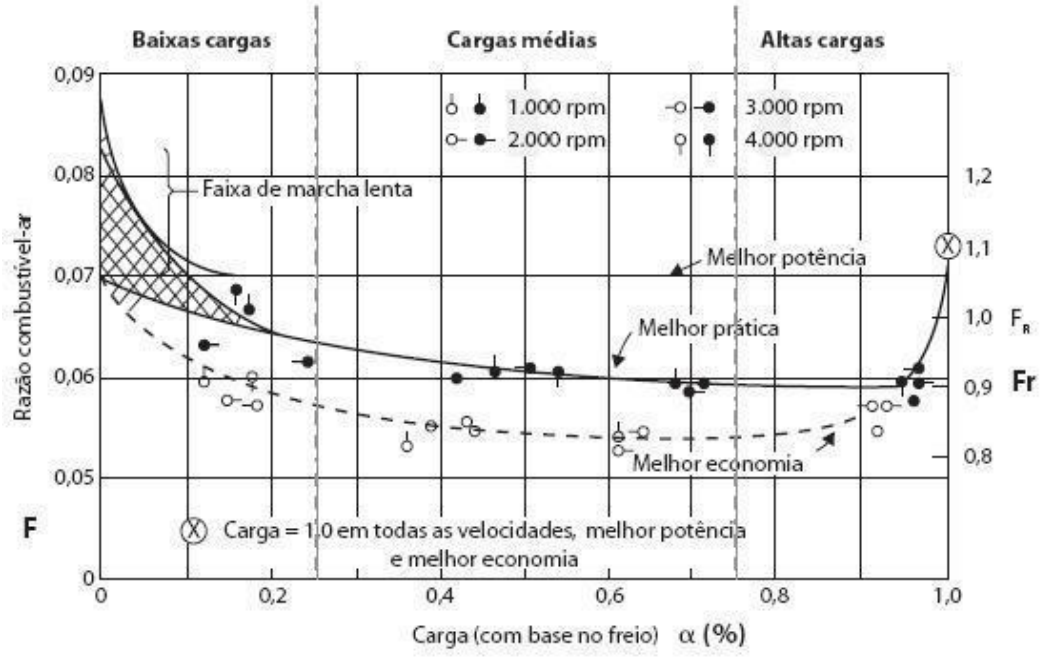
Para aliar o ferramental teórico de CPS descrito anteriormente ao sistema de um motor de ciclo Otto, é preciso conhecer os elementos que se relacionam com ele e que podem ser usados em prol de um melhor consumo energético.

Um primeiro conceito muito importante para ser desenvolvido é a relação Combustível-Ar. Brunetti (2003) apresenta como sendo uma razão entre a massa de combustível e a massa de ar que entram no cilindro. Vale destacar que uma mistura pobre (em que a razão estequiométrica da queima de combustível) torna a chama lenta e causa a instabilidade do motor. Já a rica vai causar vaporização e consumir energia térmica da chama, proporcionando, também, instabilidade. O cenário ideal é

a mistura econômica, levemente pobre, com excesso de ar que causa combustão completa e contribui para redução de emissão de monóxido de carbono (CO).

Em seguida, torna-se relevante considerar a importância do sistema de injeção. A Figura 4 mostra a curva de motor que é obtida fixando a posição da borboleta (potenciômetro que realiza o controle do fluxo de ar no cilindro) e a rotação. O sistema de injeção proporcionará a variação de massa de combustível admitida para obter a relação ar-combustível ideal (BRUNETTI, 2003).

Figura 4: Curva característica do motor em relação à mistura (os valores numéricos são apenas ordem de grandeza para efeitos didáticos)



Fonte: Brunetti (2003)

Observando a curva, Brunetti (2003) oferece quatro conclusões:

- A variação da qualidade da mistura só depende (aproximadamente) da carga, não da rotação;
- Em baixas cargas (borboleta pouco aberta), a mistura deve ser muito rica e ir empobrecendo à medida em que a carga se direciona a um nível médio;
- Em cargas médias, a mistura deve ser pobre e de qualidade constante;
- Altas cargas (borboleta muito aberta) com torque máximo requer mistura rica.

No cenário moderno, as injeções eletrônicas se destacam de outros sistemas desenvolvidos anteriormente. Ela conta com uma rede de sensores, bem como uma

unidade de comando (ECU) que dotada de um microprocessador com memórias ROM (*Ready-Only Memory*) e RAM (*Random-Acess Memory*) vai receber informações como: temperatura do ar, do combustível, volume de ar admitido, interruptores da borboleta, rotação do motor e até tensão da bateria a fim de fornecer um tempo de injeção da mistura coerente com o que foi previamente mapeado (BRUNETTI, 2003).

O funcionamento geral da ECU de um sistema de injeção eletrônica permite a abordagem do veículo como um CPS à medida que seus componentes eletrônicos se comunicam entre si e com componentes externos podendo agir em prol de algum fim específico.

2.5. Serviços de *Cloud*

Algumas empresas do mercado oferecem serviços de computação em nuvem para empresas ou desenvolvedores individuais. Com esse recurso é possível fazer o *upload*, *download*, armazenamento de dados e muito mais. Usar um ambiente pronto como esse proporciona aos projetistas a possibilidade de não se preocupar com a criação de um servidor, tão pouco com a segurança dos dados armazenados. Essa ferramenta terceiriza esses serviços por meio de uma empresa com alta confiabilidade (IBM CLOUD LEARN HUB, 2020).

Essas plataformas de nuvem abrangem nuvens públicas, dedicadas, locais e híbridas. Elas se diferenciam pela forma como o usuário e a empresa as opera. A maioria dos usuários escolhe o modelo híbrido, com algumas cargas de trabalho sendo atendidas por sistemas internos, alguns oriundos de provedores comerciais em nuvem e outros de provedores de serviço em nuvem pública (Longbottom, 2011). A nuvem proporciona aos usuários acesso instantâneo a mais de 150 serviços incluindo IoT. Seu uso se difunde cada vez mais nos setores de desenvolvimento de *softwares* e aplicativos. Para seu uso não há requisitos de *hardware*, porém há requisitos de *software*, e incluem a necessidade de um *browser* atualizado (IBM CLOUD LEARN HUB, 2020).

Os serviços de *cloud* podem ser divididos em quatro grandes blocos: *Platform-as-a-service* (PaaS), *Infrastructure-as-a-service* (IaaS), *Software-as-a-service* (SaaS), *Function-as-a-service* (FaaS).

- IaaS: Anteriormente conhecido como HaaS (*Hardware as a Service*), o IaaS é uma infraestrutura computacional provida e administrada via internet. Essa infraestrutura possui três grandes categorias: computação, armazenamento e

rede. O IaaS as oferece de forma virtualizada por um vendedor, e um usuário pode acessá-los como quiser, de forma compartilhada;

- SaaS: Um *software* que não é comercializado como um produto, mas como serviço. O sistema é abrigado em uma nuvem, o que reduz a infraestrutura por trás dele. Além disso, é possível ser acessado de qualquer lugar a qualquer momento e as melhorias são de responsabilidade da empresa fornecedora. Não precisa ser instalado na máquina e não precisa de *updates*, por exemplo, o YouTube;
- PaaS: Enquanto o IaaS é acessado por um administrador e o SaaS por um usuário comum, o PaaS é usado por um desenvolvedor que, para sua aplicação se atentará apenas para com os dados e a aplicação deles. A nuvem se responsabilizará por servidores, armazenamento, rede, sistema operacional e tempo de execução. Ele se caracteriza, portanto, como uma plataforma de desenvolvimento;
- FaaS: É um serviço de computação em nuvem derivado da filosofia *servless*, que busca abstrair a questão de servidor e máquinas onde rodam seus programas. “Permite criação, gerenciamento e entrega de solução na nuvem em um modelo baseado em eventos. Cria-se o serviço no formato de funções, que funcionam por um evento, com acesso a uma API ou uma alteração no banco de dados.” – IBM Cloud Learn Hub (2020)

Para esse projeto será usada a abordagem PaaS. Ela auxiliará no tráfego de informação atuando como elemento intermediário, recebendo informações do computador e enviando ao veículo.

Safonov (2016) compara a Oracle *Cloud* ao IBM *Cloud*, e já destaca a inferioridade do primeiro em PaaS. Para ele, a integração oferecida pela IBM é mais forte do que a oferecida pela Oracle Corporation, contudo não é deixada de lado a análise de outros pontos da ferramenta. Nela, o usuário encontrará IaaS, PaaS, SaaS, DaaS (*Data-as-a-Service*), que confere ao cliente a possibilidade de agregar e analisar seus dados, entre outros serviços. O modelo de implantação pode ser de nuvem pública, privada e híbrida, oferecida por meio de uma rede global de *data centers*.

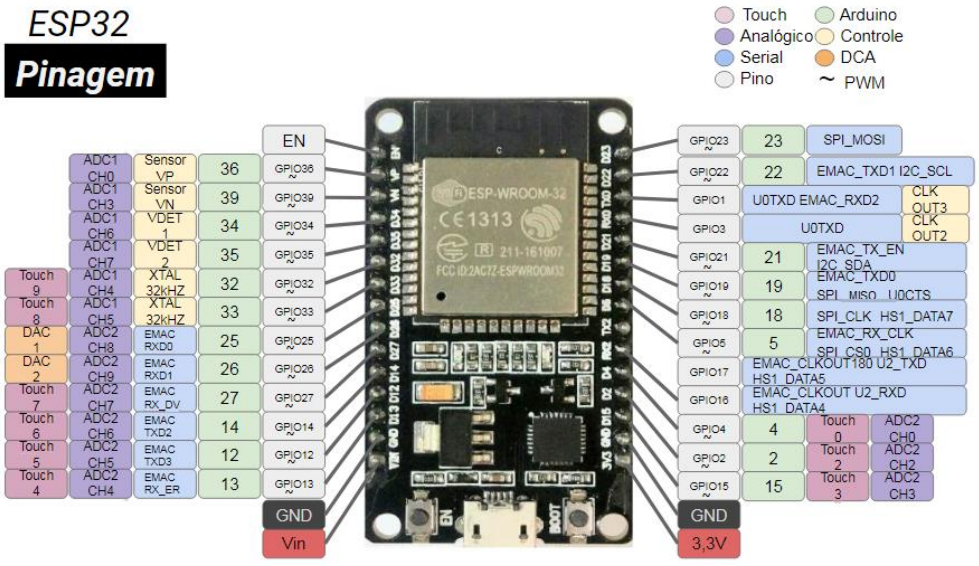
2.6. ESP 32

A família de microcontroladores ESP vem ganhando espaço no cenário de IoT. Sua peculiaridade é a possibilidade de processar protocolo WiFi. Muito usado em automação residencial, projetos que popularizam seu uso e exploram suas capacidades podem ser facilmente encontrados na internet (OLIVEIRA, 2017).

O módulo ESP-01 é o mais simples do mercado, além de contar com o microcontrolador ESP, possui memória externa de até 1MB, oito pinos (comunicação serial e alimentação) e é necessário uso de adaptadores USB para gravar. Seu consumo energético é muito baixo, atingindo no máximo 15mA em modo *sleep*, quando não há envio ou recepção de pacotes. Aplicá-los a um projeto mais robusto não é recomendável devido às suas limitações de projeto. Assim, são usados frequentemente como módulo de comunicação WiFi com um Arduino ou RaspberryPi (OLIVEIRA, 2017).

O ESP 32 é um módulo de alto desempenho e baixo consumo de energia. Conta com um microcontrolador de dois núcleos, bem como antena, interface USB-Serial, regulador 3.3V, 4MB de memória *flash*, blindagem férrica que evita interferências e 32 pinos programáveis. A Figura 5 mostra a pinagem deste microcontrolador (OLIVEIRA, 2017).

Figura 5: Pinagem do ESP 32



Fonte: Blog Curto Circuito, 2018

Sua programação aceita linguagem LUA ou C++, por meio da IDE do Arduino. Seu consumo energético se torna um pouco mais alto do que quando comparado aos

outros módulos da família, consome cerca de 80 mA em modo *sleep*, devido ao WiFi e Bluetooth integrado (OLIVEIRA, 2017).

2.7. *Message Queue Telemetry Transport* (MQTT)

O protocolo MQTT foi projetado pelos engenheiros Andy Stanford-Clark e Arlen Nipper em 1999. A ideia era conectar sistemas de telemetria de oleodutos por satélite. O protocolo, inicialmente era de uso particular, porém, em 2010, foi lançado em *royalties* e se tornou um padrão OASIS em 2014 (The HiveMQ Team, 2015).

Atualmente, cada vez mais dispositivos ganham a capacidade de se conectarem à internet, daí o protocolo MQTT oferece uma alternativa simples, leve e segura que possibilita a dinâmica de comunicação entre máquinas.

Sob o ponto de vista da arquitetura, um dispositivo pode ser *publish,* isto é, ele publica informações pré-estabelecidas, ou *subscribe*, o que recebe todas essas informações que estão sendo publicadas. Um outro elemento do MQTT é o *broker*, ele é um servidor que atua como intermediário entre *subscribers* e *publishers*, isto é, recebe todas as mensagens, filtra e decide quem está interessado e inscrito nela ou não. O *broker* pode ser tanto um servidor local quanto um serviço em nuvem (FERNANDES, 2021).

Do ponto de vista da aplicação desse projeto, com um protocolo MQTT, as informações de um veículo são coletadas e armazenadas no *broker* no tópico “velocidade” ou “temperatura”, por exemplo. A partir do momento que escolhem-se os *subscribers*, eles vão receber apenas as informações relativas a estes tópicos nos dispositivos cadastrados, facilitando o acesso aos dados de temperatura para as análises necessárias.

2.8. Consolidações sobre o capítulo

Sob o ponto de vista de uma elaboração de proposta, os conceitos de CPS foram associados ao estudo envolvendo veículos de motor a combustão e protocolo de comunicação CAN. Essa união tornou possível a elaboração de uma proposta que descreve veículos mais inteligentes e que são capazes de tornar decisões, utilizando de conceitos abordados em IoT e Big Data.

A fim de implementar a proposta que será descrita no capítulo seguinte, foram usados os conceitos de nuvem aliados ao que foi discutido a respeito do microcontrolador ESP32.

3. Proposta

Historicamente, o sistema de telemetria simbolizou um grande avanço nas competições automobilísticas. Do ponto de vista de Calderón (2013), um sistema de telemetria deve contar com 3 fatores: dispositivo eletrônico, comunicação sem fio e um *software*. O dispositivo eletrônico indicado pelo autor necessita atender requisitos de dimensões, facilidade de controle, consumo de energia e, principalmente, precisa encapsular e enviar os dados para o sistema de comunicação sob a forma de um protocolo robusto e funcional. O sistema de transmissão desses pacotes necessita de um canal sem fio apropriado à médias distâncias, precisa ser robusto, seguro e apresentar poucos ruídos. Para criar um sistema de comunicação para esse fim é necessário eleger o tipo de tecnologia a utilizar, o tipo de processador e a arquitetura mais indicada para o projeto. O *software*, por sua vez, precisa ser capaz de se conectar ao veículo em busca de dados técnicos, dados do piloto e apresentar todos esses dados graficamente.

Com a chegada da internet, fica evidente a possibilidade de conectá-la com os conceitos de Calderón e Ruiz (2013). Essa união permite estender o alcance da comunicação e diminuir o tempo de latência. Exemplos como o de Husni e Hertantyo (2016) e o de Wang e Lei (2012) permitem verificar que existem dois tipos de abordagens para construir uma lógica de telemetria baseada em internet.

Husni e Hertantyo (2016) fazem uso de um ambiente pronto da IBM, o BlueMix, para fazer o *upload* de informações de natureza técnica à nuvem do servidor. Nessa nuvem um sistema de análise de Big Data informa se o veículo precisa de algo e, se precisar, o sistema envia mensagens no celular do condutor.

Outra lógica para a construção da relação de um veículo que se comunique com um servidor pode ser feita sem ambientes prontos. Wang e Lei (2012) desenvolveram um sistema baseado em 3 pontos: servidor, cliente e uma unidade de microcontrolador. O servidor controla os modos do veículo e envia informações do estado do mesmo para o cliente via protocolo *Transmission Control Protocol/ Internet Protocol* (TCP/IP). O cliente envia informações do veículo para o servidor e informações de controle para o microcontrolador, ele atua como uma central que recebe informações dos sensores do veículo e as envia ao cliente.

Os dois exemplos permitem refletir que a literatura a respeito do assunto se divide em usar um ambiente já consolidado pelo mercado ou utilizar um ambiente

próprio. É claro que em um ambiente próprio, muitas vezes os recursos serão limitados, porém focados ao que se precisa. Por isso, vai do engenheiro ou do pesquisador determinar qual método é mais apropriado e mais aplicável ao seu projeto, neste caso foi decidido utilizar uma solução comercial.

A proposta desse trabalho é tornar veículos mais inteligentes abordando-os como um CPS e os conectando com o intuito de obter uma rede de veículos. Essa abordagem permitirá que se reúna um grande volume de dados para que cada unidade de veículo possa acessá-los e melhorar seu desempenho, adaptar-se a mudanças do ambiente e prever falhas.

3.1. Descrição dos elementos

A proposta deste trabalho considera a composição da interface de monitoração e controle remoto de um veículo automotor pelos seguintes elementos:

- Unidade de coleta de dados do mundo físico: este primeiro elemento é o responsável por realizar a leitura das informações oriundas do mundo físico, essas informações podem ser das mais diversas naturezas dependendo de cada elemento que é inserido nessa unidade. Por exemplo, sensor de rotação, sensor de velocidade, sensor de temperatura, etc.;
- Microcontrolador: este é responsável por agregar as informações do mundo físico e se comunicar com o próximo elemento do ciclo;
- Componente provedor de rede: sua função é permitir a emissão e o recebimento dos dados a grandes distâncias com integridade;
- Gerenciador de banco de dados e transações: trata-se de um banco de dados responsável pelo armazenamento das informações históricas coletadas pelo primeiro elemento descrito. Esse banco localiza-se no exterior do veículo e provém dados que são interpretados e analisados com o intuito de promover maior inteligência na utilização do veículo;
- Dispositivo de controle remoto: atua como interface e interpretador das informações oriundas do veículo; pode contar com um operador ou realizar decisões de forma automática;
- Unidade de atuação no mundo físico: atua no mundo físico com base nas decisões tomadas pelo elemento de controle remoto; esse elemento

consiste sumariamente em atuadores dispostos no veículo preparados para realizar ações que tenham sido necessárias.

Alguns dos elementos citados serão dispostos no veículo como é o caso da “unidade de coleta de dados do mundo físico”, “microcontrolador”, “componente provedor de rede” e “unidade de atuação no mundo físico”, juntos eles compõem o arcabouço físico da interface de monitoração e controle remoto de um veículo automotor. Por outro lado, os elementos de “gerenciador de banco de dados e transações” e “dispositivo de controle remoto” se apresentam como unidades do mundo *cyber*. O fluxo de informações será formado ao interagirem entre si.

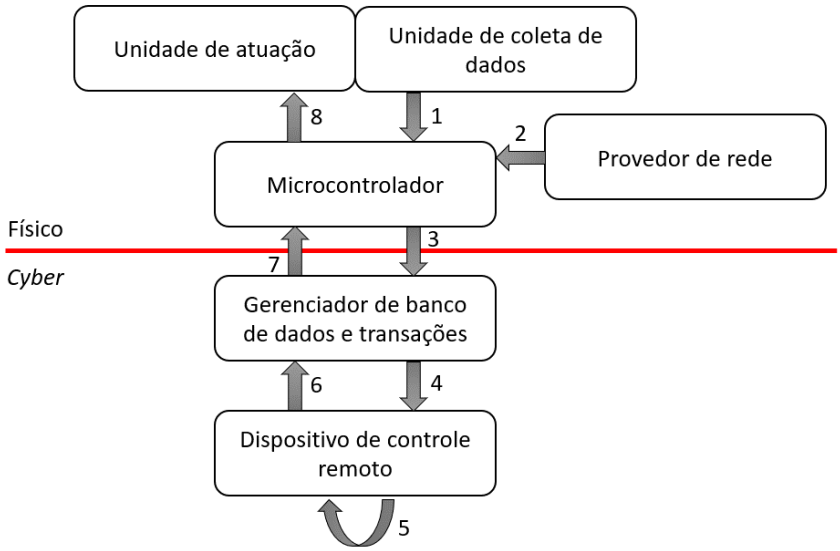
3.2. Interações

As interações dos elementos citados podem ser vistas na Figura 6 e são divididas em:

1. Envio de informações dos sensores, via rede CAN, para o microcontrolador;
2. Fornecimento de sinal para viabilizar a comunicação do microcontrolador;
3. Envio das informações integradas no microcontrolador para o gerenciador de banco de dados e transações;
4. Consulta do banco de dados realizada pelo dispositivo de controle remoto;
5. Processamento de informações realizada pelo dispositivo de controle remoto;
6. Atualização de valores do banco de dados em forma de comando;
7. Envio das informações atualizadas do servidor para o microcontrolador;
8. Transmissão dos comandos para os atuadores via rede CAN.

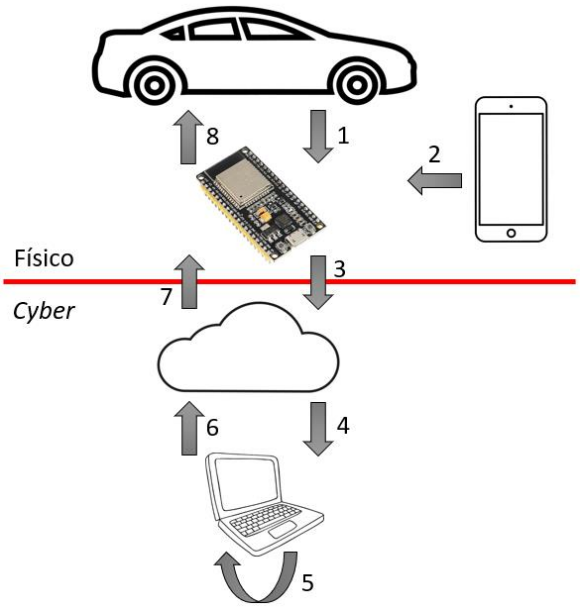
No que tange à abordagem do veículo como CPS, a proposta é que se use de diversas tecnologias à disposição para compor um fluxo de informação. O conceito de internet das coisas permite integrar veículos ao redor do globo de modo a permitir o compartilhamento das informações lidas pelos sensores de cada uma das unidades equipadas com essa tecnologia.

Figura 6: Ilustração dos elementos e interações



As informações dos sensores do veículo e o controle dos atuadores serão transmitidos através de protocolo CAN para um *hardware* que será instalado no veículo. Esse *hardware* deverá ser capaz de se comunicar com a unidade de controle eletrônico (ECU), recebendo os valores de diversos sensores instalados no veículo, que servem para o seu funcionamento, como sensor de rotação do motor, sensor lambda, sensor de temperatura, sensor de posição da borboleta (TPS), entre outros, e também deverá enviar comandos para modificar os parâmetros dos atuadores como o bico injetor e a ignição. Um esquema de funcionamento pode ser visto na Figura 7.

Figura 7: Esquema de funcionamento do sistema proposto



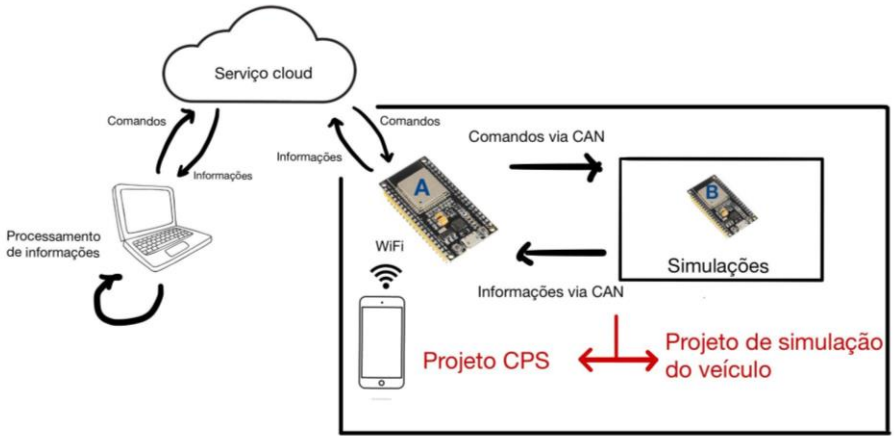
4. Implementação de um sistema de monitoramento / telemetria veicular

Nessa seção é realizada a discussão sobre o sistema proposto, explicando as partes envolventes.

4.1. Visão geral

A proposta desse trabalho busca a realização de sensoriamento e controle de forma remota de parâmetros de um veículo. A Figura 8 mostra um esquema de funcionamento do sistema proposto.

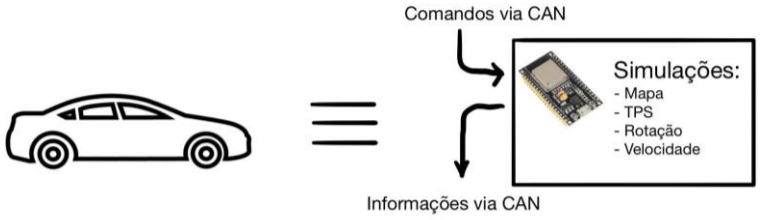
Figura 8: Esquema de funcionamento da adaptação do sistema proposto



4.2. Representação do veículo

Tomando o veículo com uma caixa preta, é importante destacar que ela precisa receber e enviar dados via rede CAN e exibir parâmetros monitorados. Dessa forma, torna-se irrelevante se os testes serão impostos em um veículo real ou em um modelo virtual, desde que esse modelo realize esse processo via rede CAN. A Figura 9 mostra o esquema proposto para a representação do automóvel.

Figura 9: Esquema do modelo virtual do veículo



O modelo será composto por uma segunda placa ESP32, que será chamada de Placa B. Ela simulará os parâmetros de um veículo. Assim, enquanto a placa A faz a conexão WiFi com a nuvem, a B integra a modelagem do veículo virtual.

Um primeiro passo para atingir um modelo de veículo compatível com a ideia original do projeto é modelar a injeção eletrônica do veículo. Neste projeto foi inicialmente utilizado o Excel, que foi utilizado para testes iniciais e cujos resultados estão documentados na seção “Testes Iniciais”. O arquivo em Excel contém uma tabela que representa o mapa do motor. Além disso também há algumas variáveis aleatórias que são usadas para indicar qual a magnitude da alteração que deve ser feita no mapa a fim de se obter um melhor consumo de combustível do veículo.

Através das variáveis aleatórias de velocidade, rotação e TPS, o programa desenvolvido em *Visual Basic for Applications* (VBA) irá identificar se a mistura de combustível é pobre ou rica e fará o ajuste no mapa de acordo com a leitura do sensor lambda.

4.3. Roteamento de sinal

Com a movimentação constante do *hardware*, que permanecerá no veículo, há preocupação com o alcance do sinal de internet utilizado. Isso porque conforme o veículo se desloca, a distância do ponto inicial aumenta e sua intensidade diminui. Assim, seria necessária uma rede Wi-Fi com uma cobertura muito grande, o que não é possível. Pensando nessa limitação, a solução é utilizar um celular com acesso à internet que compartilha seu sinal de rede através do roteamento.

A placa de rede do dispositivo móvel é capaz de realizar o processo de conversão da rede móvel para um *hotspot* Wi-Fi. Assim, é possível usá-lo como roteador para algum outro dispositivo. Nesse caso, com o celular dentro do veículo, a intensidade do sinal se manterá constante, fornecendo uma conexão suficientemente robusta para o envio e recebimento de informações. Contudo, no futuro os veículos devem ser nativos conectados.

4.4. Comunicação ESP32/Nuvem

As informações coletadas pelo ESP32 precisam ser enviadas para um computador remoto. A maneira encontrada foi a utilização de um serviço de nuvem que consegue captar esses dados e enviar para um servidor com um atraso muito pequeno que, para a percepção humana, pode ser considerado tempo real.

A utilização de um serviço de nuvem que é capaz de captar dados e enviar para um servidor com um atraso estimado em 300ms é ideal para o projeto. Como o funcionamento do motor não varia nessa velocidade, esse atraso é tolerável e não influencia na obtenção de informações. Da mesma forma, é capaz de enviar comandos para o ESP32, de forma que os parâmetros dos atuadores possam ser alterados.

O protocolo dessa comunicação é o MQTT. Por ser leve, aberto, simples e projetado para ser fácil de implementar torna-o ideal para uso em muitas situações, incluindo ambientes restritos, como para comunicação em contextos *Machine to Machine* (M2M) e *Internet of Things* (IoT), onde um código enxuto é necessário e/ou a largura de banda da rede é preciosa. O protocolo é executado em TCP / IP ou em outros protocolos de rede que fornecem conexões bidirecionais ordenadas, sem perdas (BANKS, 2019).

4.5. Comunicação Nuvem/Computador

Os dados que foram armazenados no servidor em nuvem podem ser acessados em tempo real de qualquer computador com acesso à internet em qualquer lugar do mundo. O usuário ainda tem a possibilidade de alterar algum valor dos atuadores por meio da própria interface em nuvem. Cabe o adendo de que o banco de dados em nuvem pode ser usado como treinamento de um classificador presente no computador, que determinaria quais são os melhores parâmetros para enviar aos atuadores, buscando do veículo um trabalho mais eficiente.

O aplicativo da web Arduino IoT Cloud no desktop ou móvel permite que o usuário conecte, gerencie e monitore seus dispositivos de qualquer lugar do mundo. O Arduino IoT Cloud também cria automaticamente o código para programar o dispositivo (nesse caso o ESP32). O Arduino IoT Cloud realiza a comunicação do dispositivo IoT com a nuvem através do protocolo SSL padrão do setor para criptografia. As famílias de placas Arduino MKR e Arduino Portenta têm chips de autenticação de criptografia *on-board* e são protegidas usando autenticação baseada em certificado X.509 (Arduino Cloud IoT, 2022).

Essa ferramenta possibilitou que os dados como velocidade, rotação, TPS e lambda, sejam não só armazenados e consultados, mas também torna o controle do veículo viável dado que ela permite que um operador altere parâmetros e os envie de volta para o veículo.

5. Testes e validação do fluxo de informação

Com o objetivo de dar vida ao fluxo de informação que monitora e controla um veículo, utilizou-se de testes isolados para no final unir todos os componentes em uma única corrente de informação.

Para melhor familiarização com os serviços de nuvem, foram realizados testes no IBM Cloud e no Arduino IoT Cloud. Além disso, um primeiro simulador de um veículo foi realizado no Excel para facilitar a visualização dos parâmetros, e posteriormente foi utilizado um segundo ESP32 para essa finalidade. Do ponto de vista de conexão à internet, os testes procuraram enviar a servidores locais e globais informações de sensores e LEDs. Para a programação dos microcontroladores, o *software* utilizado foi a IDE do Arduíno.

5.1. Testes iniciais

Uma primeira bateria de testes foi realizada a fim de validar o microcontrolador escolhido. Ao longo das seções, serão discutidos os resultados individuais de cada teste de conexão do ESP32 à internet.

5.1.1. ***Servidor local***

Os testes no servidor local foram os primeiros a serem desenvolvidos já que era necessária uma familiarização com as ferramentas e a linguagem de programação do ESP32.

Foi feita uma montagem simples de um servidor web que acende ou apaga dois LEDs para testar o funcionamento do WiFi. Para isso, foram utilizados os seguintes componentes, ligados conforme o esquemático mostrado na Figura 10:

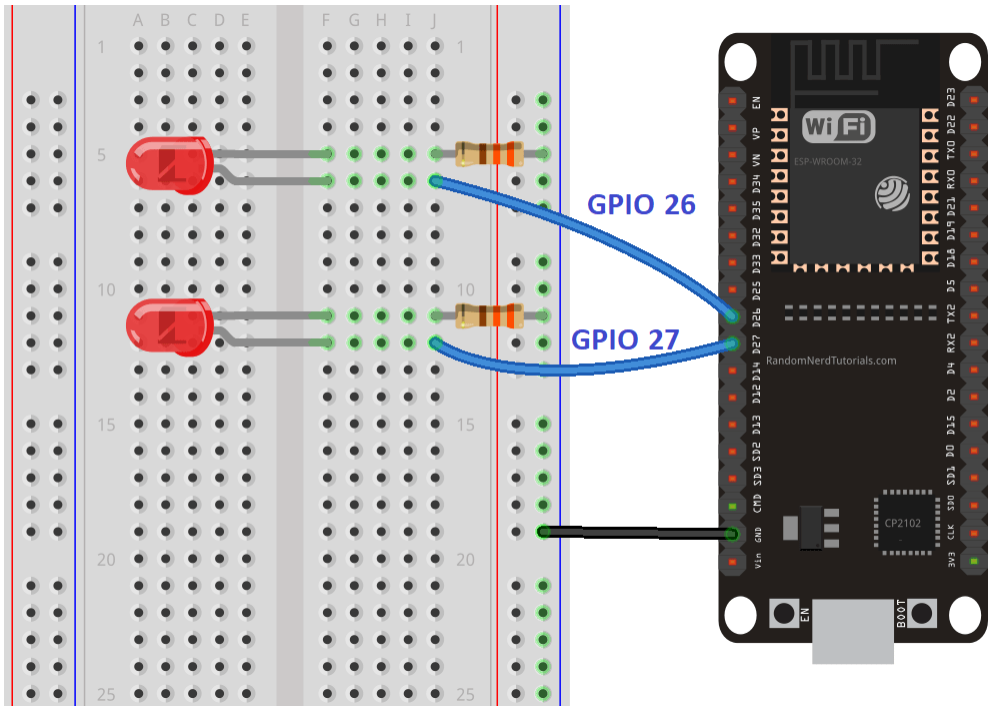
- ESP32
- 2x LEDs
- 2x resistores de 330Ω
- *Protoboard*
- *Jumpers*

Após realizar as ligações e acrescentar os parâmetros da rede WiFi, o código de Santos (2018), no anexo A, foi enviado para o ESP32 por meio da IDE do Arduino.

Com o código no módulo, o endereço IP do ESP32, por onde será feito o comando de liga/desliga dos LEDs, foi obtido por meio do Monitor Serial da IDE do

Arduino. Apertando o botão de *reset* do módulo, o endereço foi mostrado em um dos campos de informações.

Figura 10: Esquemático das ligações do Web Server



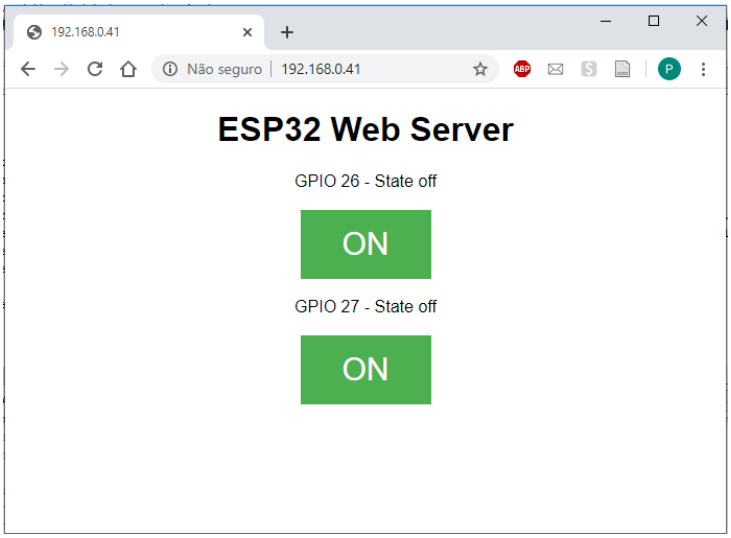
Fonte: Santos (2018)

Conectado a uma rede WiFi e tendo o endereço IP do ESP32, digita-se o IP em um navegador web para que uma página igual à da Figura 11 seja exibida, com os botões liga/desliga dos LEDs. O endereço funciona para qualquer dispositivo conectado à mesma rede, inclusive o celular.

Ao apertar algum dos botões, o respectivo LED trocará seu *status* (aceso ou apagado), como demonstrado na Figura 12.

Essa montagem serviu de teste para o funcionamento do módulo WiFi, mas se percebeu que ela provavelmente não serviria para este projeto, pois requer que o ESP32 e o computador estejam na mesma rede. Como o protótipo estaria dentro de um veículo, e esse veículo estaria em movimento, faria pouco sentido assumir que haveria uma rede WiFi que alcance os dois. Por isso, testou-se outra estratégia que visasse uma comunicação sem que os componentes estivessem na mesma rede.

Figura 11: Servidor Web do ESP32



5.1.2. *Servidor global*

Em um segundo teste, enviar os dados do ESP32 para um servidor global (uma nuvem) foi o desafio. O teste com servidor local havia demonstrado que o emissor e o coletor da informação da rede teriam que estar conectados em pontos de acessos distintos. A partir dessa reflexão, o teste consistiu em enviar informações do ESP32 para a IBM Cloud via ponto de acesso A e ler as informações do IBM Cloud via ponto de acesso B.

Foi feita outra montagem simples, porém visando testar a comunicação usando uma rede global, em que os dados pudessem ser acessados de qualquer lugar. Para isso, utilizou-se os seguintes componentes, ligados conforme o desenho esquemático mostrado na Figura 13:

- ESP32
- Sensor de temperatura LM35
- *Protoboard*
- *Jumpers*

Como resultado, obteve-se a montagem apresentada na Figura 14.

Para fazer o teste, utilizou-se a IBM *Cloud*. Apenas criando um ID para um dispositivo e colocando no campo designado no site da IBM, como mostra a Figura 15, as informações que o sensor fornece já começam a ser exibidas.

Figura 12: Demonstração de funcionamento do servidor web

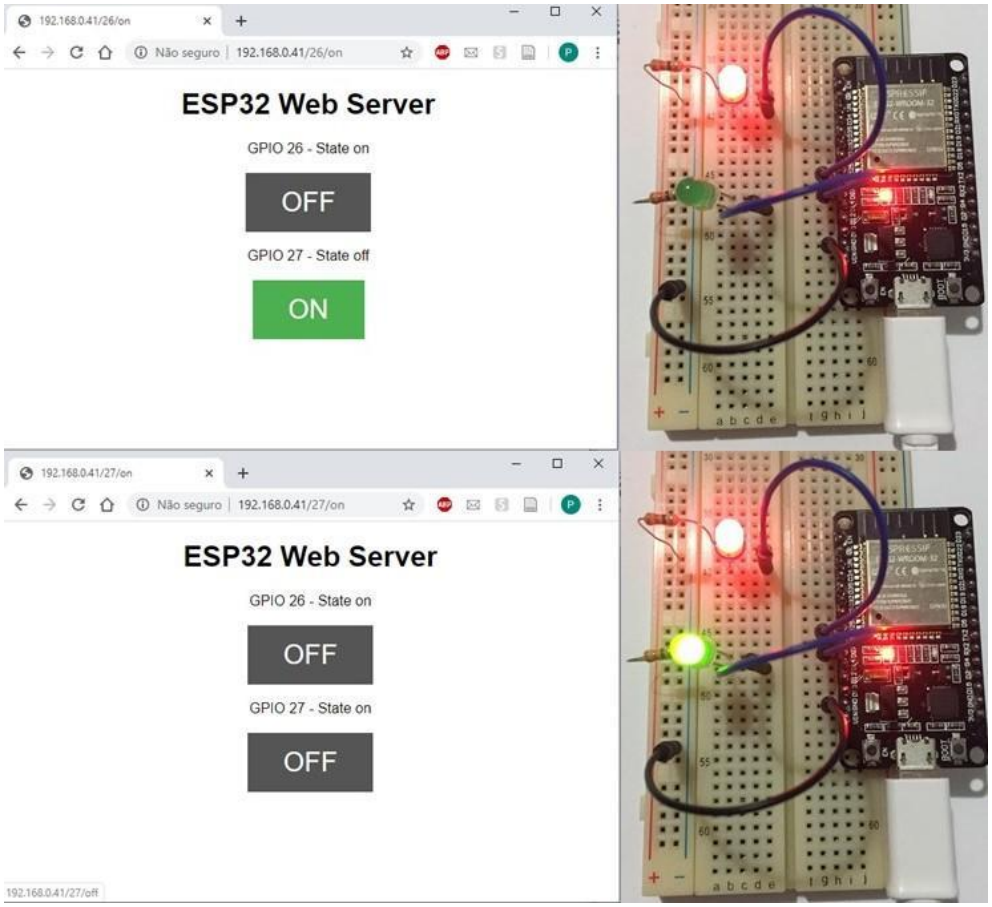


Figura 13: Desenho esquemático das ligações feitas para a testagem da IBM Cloud

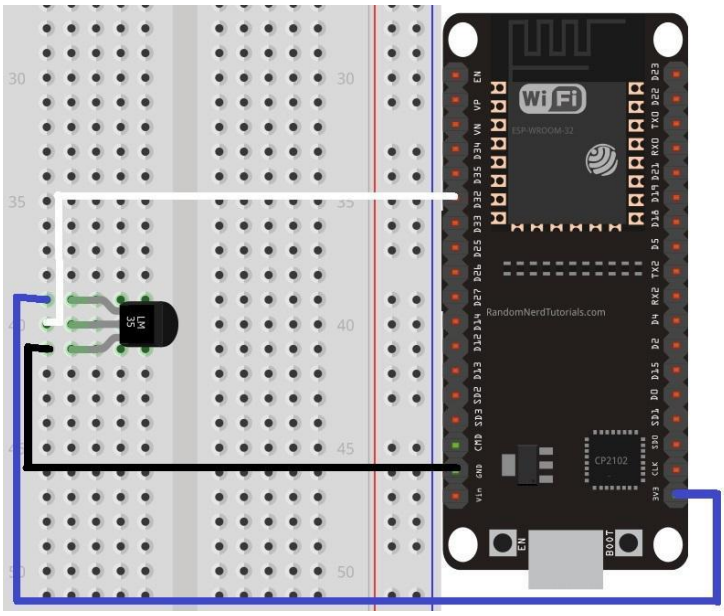


Figura 14: Implementação do circuito projetado na Figura 13

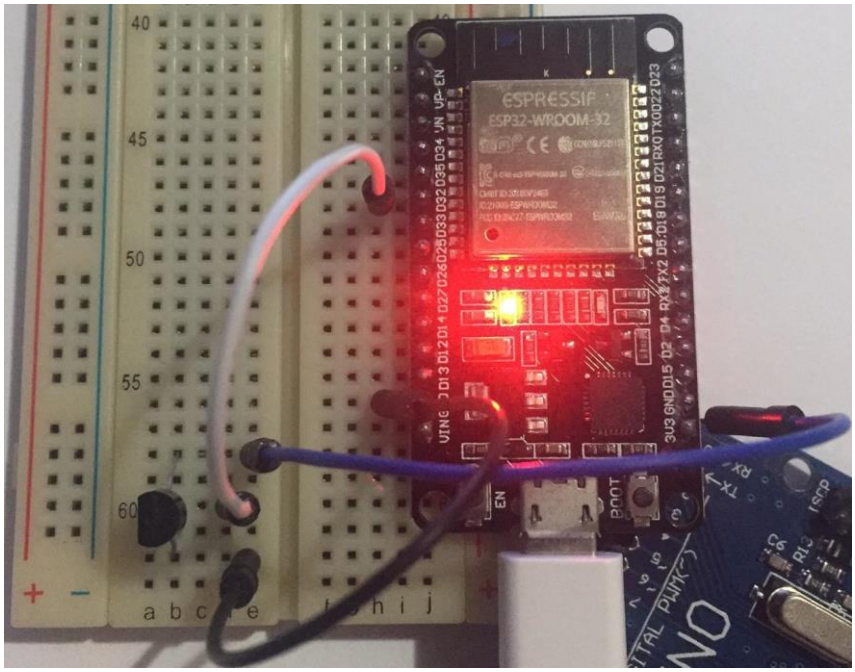
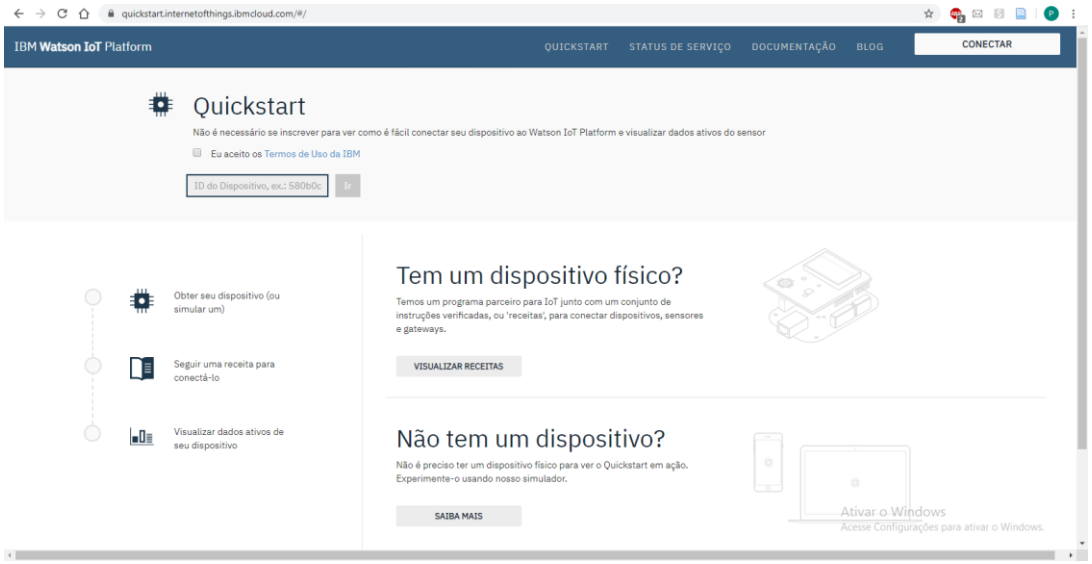


Figura 15: Site de um serviço de Cloud



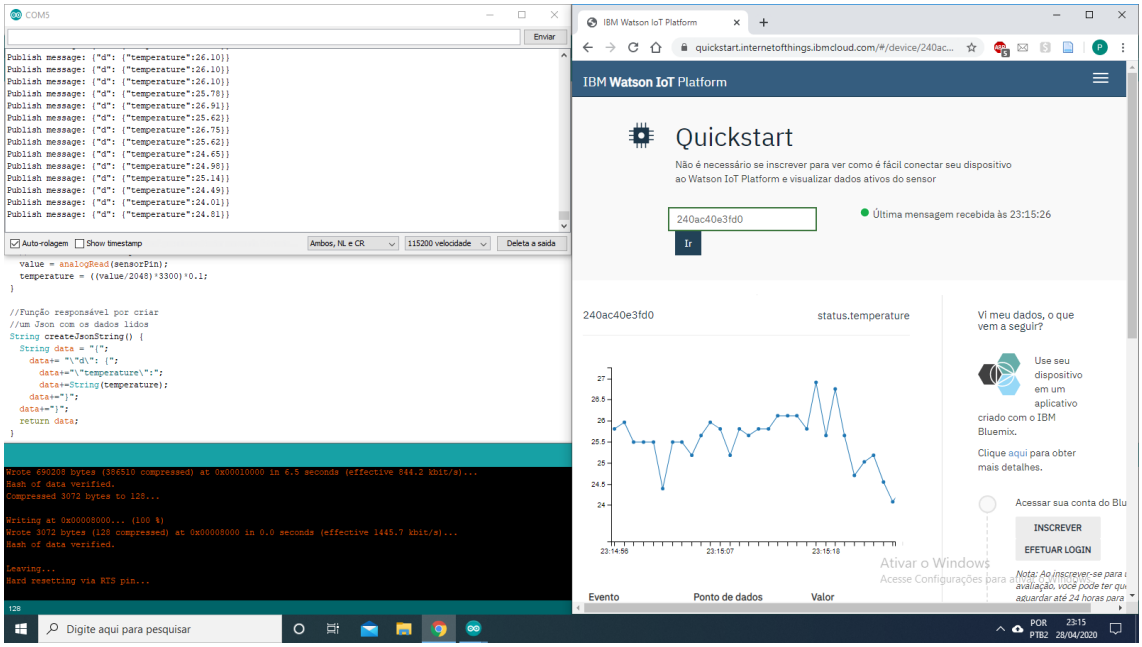
O código final, mais adequado para os testes realizados, foi obtido após modificações no código de Koyanagi (2018) e está documentado no Anexo B.

Com o código no ESP32, as ligações feitas e o ID preenchido no *site*, as informações puderam ser visualizadas na plataforma da IBM, sincronizadas com as vistas no monitor serial, como pode ser observado na Figura 16.

Tanto o *software* quanto o *hardware* se comportaram como esperado no projeto. Os dados puderam ser lidos e gravados na nuvem e visualizados via um

dashboard da plataforma. Contudo, apesar do resultado desse teste ter oferecido o direcionamento de que um serviço de nuvem comercial seria viável, o IBM Cloud foi descartado por não tornar possível em uma versão gratuita, a emissão de dados da nuvem para o ESP32, impossibilitando o controle via nuvem.

Figura 16: Informações sendo vistas em tempo real por uma solução de Cloud



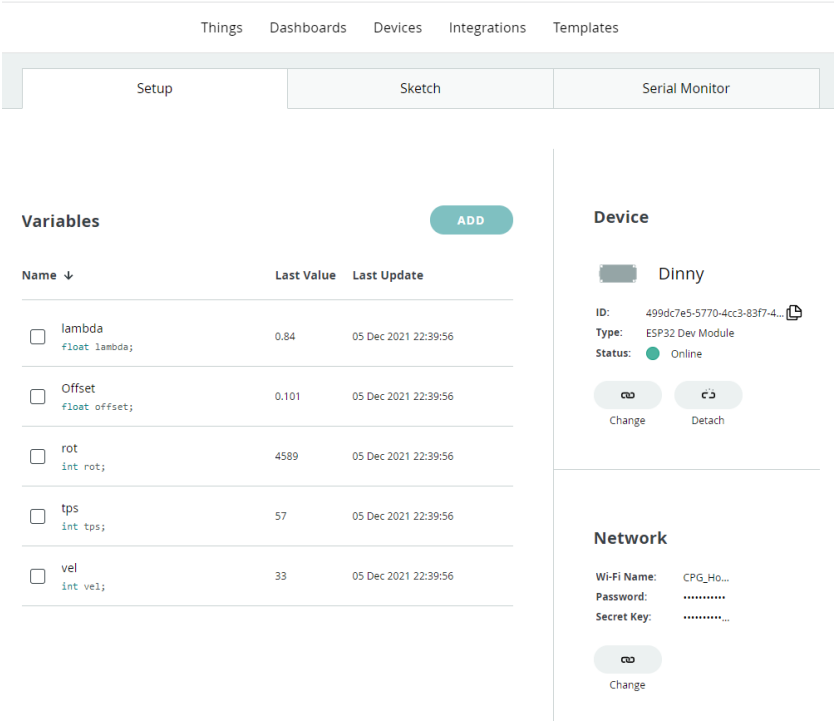
5.2. Cloud computing

Outra solução comercial robusta e capaz de receber uma grande quantidade de dados oriundos de diversas fontes, que seja fácil de usar e de conectar a outros serviços é o IoT Cloud Arduino. A título de teste, foi decidido o uso de 5 variáveis para testar a viabilidade da plataforma e ela se comportou bem recebendo os dados simulados. Ao confirmar a viabilidade com 5 variáveis, um número maior de parâmetros pode ser explorado futuramente, se a implementação for elevada a um nível comercial. O que diferencia o IoT Cloud Arduino de outras soluções comerciais como o IBM Cloud é a facilidade de gerenciar os dados e os escrever utilizando ferramentas populares como o próprio Arduino. Além disso, a plataforma oferece diversos recursos adicionais que prometem aumentar a gama de possibilidades de implementações e funções da ferramenta (Arduino Cloud IoT, 2022).

Foram declaradas variáveis que seriam utilizadas para monitorar e controlar o veículo. Na aba *device*, foi necessário configurar o dispositivo ESP32 e na aba

Network inserir as informações da rede WiFi utilizada para conectar o ESP32. A Figura 17 ilustra o *setup* completo do projeto.

Figura 17: *Setup* do IoT Cloud Arduino



Essa plataforma possui uma aba *Sketch* para que o código seja desenvolvido e carregado na placa sem a necessidade de uma IDE, como ilustrado na Figura 18. O código utilizado se encontra no Anexo E.

Também está disponível no IoT Cloud Arduino a ferramenta para a construção de um *dashboard*, com ilustrado na Figura 19. Nele é possível criar uma central de controle e monitoramento das variáveis utilizadas. Neste projeto foram consideradas as variáveis de monitoramento de velocidade, rotação, TPS e sensor lambda. Por meio delas o usuário poderá monitorar o desempenho do veículo enquanto ele se desloca.

Por outro lado, a variável *offset* permite que o usuário controle o passo de alteração do valor do mapa do motor, isto é, dependendo da leitura feita do sensor lambda. Ele poderá remapear o motor para um tempo de injeção de combustível maior ou menor.

Figura 18: *Sketch* para desenvolvimento do código

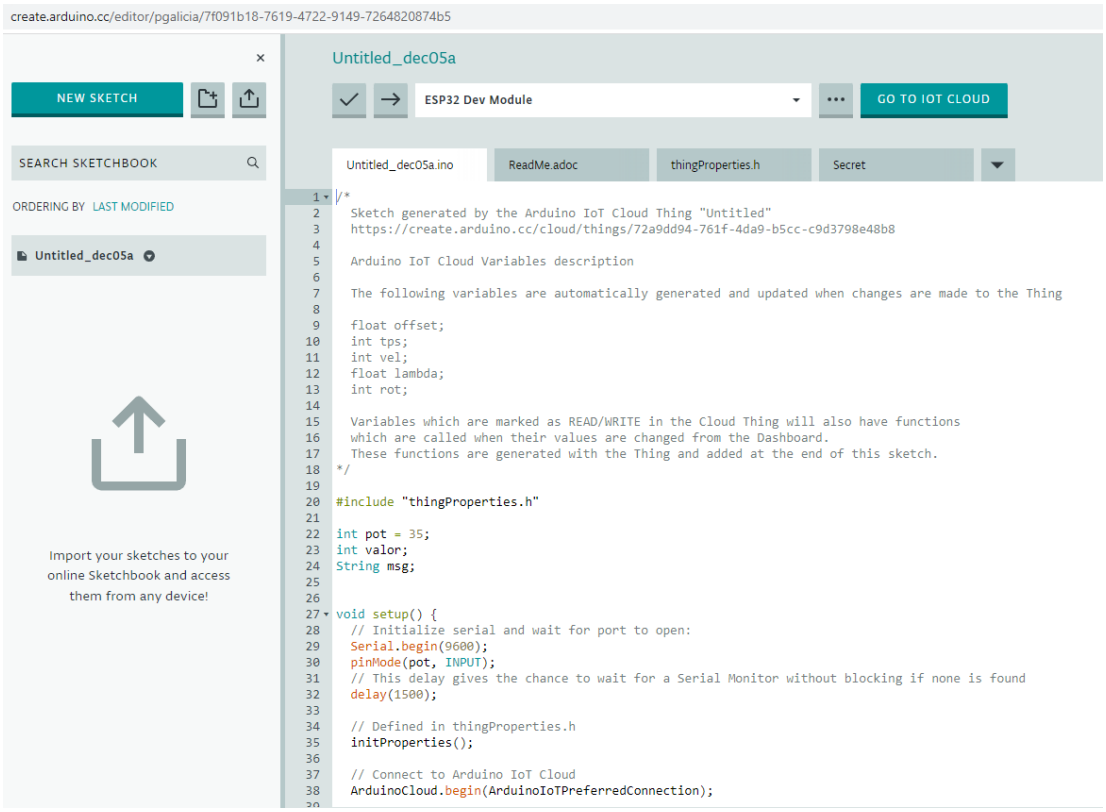
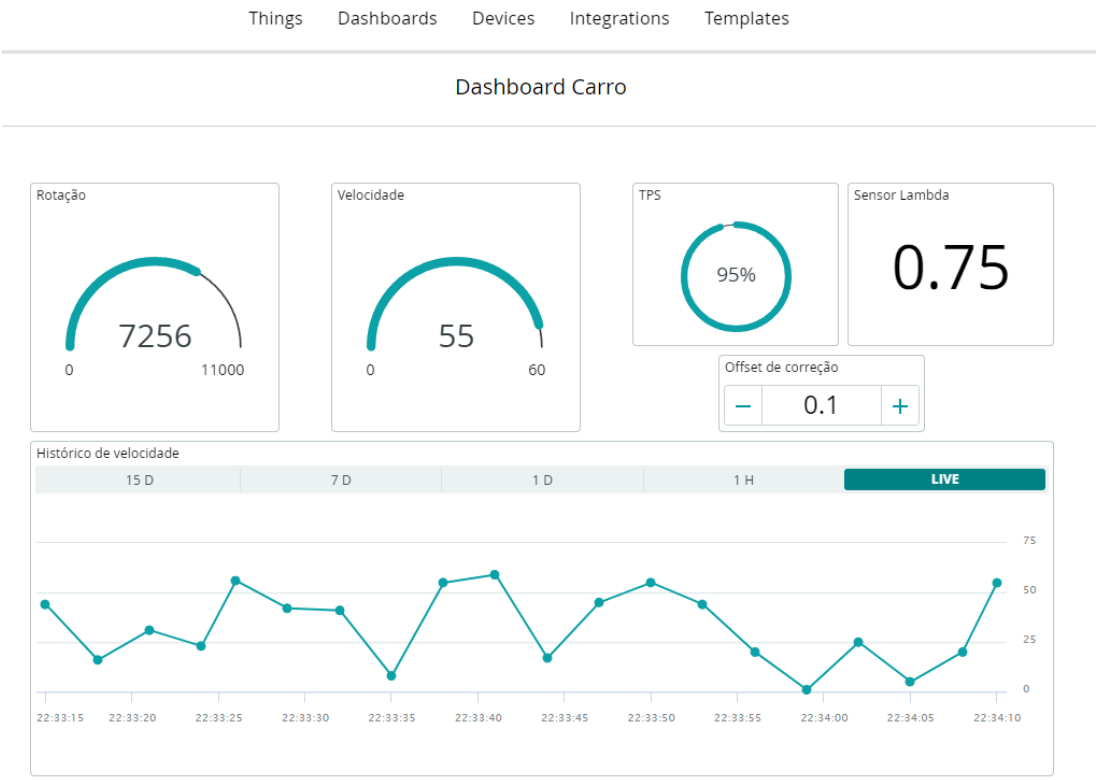


Figura 19: *Dashboard* para monitoramento e controle



5.3. Simulador do veículo

A planilha desenvolvida tem como intuito simular uma injeção eletrônica. A ideia inicial foi utilizar o *Microsoft Excel* devido a facilidade de uso e a possibilidade mais clara de visualizar os dados. Caso ele atendesse as necessidades do projeto, a lógica implementada nele poderia ser replicada de uma outra forma mais robusta e eficiente e só assim poderia enviar informações à nuvem.

Com alguns parâmetros iniciais de entrada (rotação do motor e porcentagem de acionamento da borboleta do acelerador), o simulador encontra o ponto de interesse no mapa do motor e ajusta o valor de acordo com o *offset* estabelecido previamente. Caso o valor medido no sensor lambda seja menor que 1, significa que a mistura está rica, então o valor cruzado na tabela precisa diminuir (valor – *offset*). Caso contrário (mistura pobre) o valor da tabela aumentará (valor + *offset*).

O simulador funciona gerando valores aleatórios para Rotação, Aceleração e Lambda, como pode ser visto na Figura 20, e realizando o procedimento descrito, alterando os valores do mapa do motor, como pode ser visto na Figura 21. O código escrito para realizar o procedimento no *Microsoft Excel* encontra-se no Anexo C.

Figura 20: Interface de geração de parâmetros

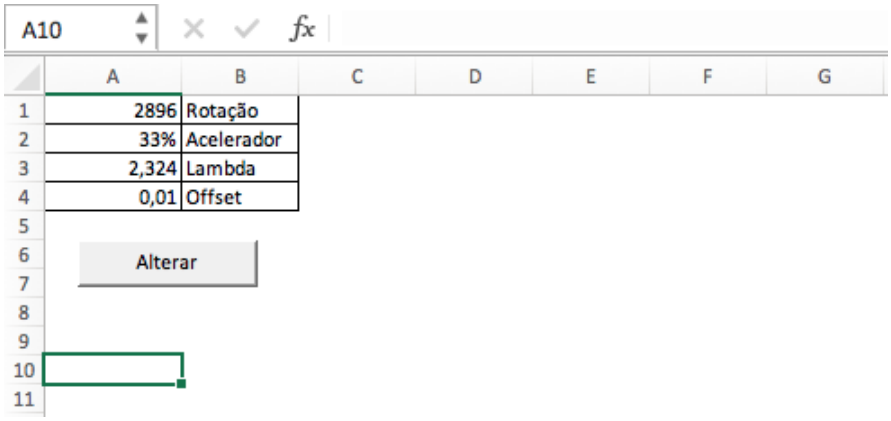


Figura 21: Mapa do motor

	A	B	C	D	E	F	G	H	I	J	K	L
1		0	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
2	0%	0,1	0,1	0,11	0,11	0,1	0,1	0,14	0,14	0,1	0,24	0,24
3	10%	0,1	0,11	0,12	0,12	0,1	0,12	0,14	0,15	0,11	0,17	0,22
4	20%	0,1	0,11	0,15	0,12	0,1	0,13	0,14	0,12	0,12	0,13	0,13
5	30%	0,1	0,11	0,14	0,14	0,12	0,11	0,11	0,15	0,12	0,13	0,14
6	40%	0,15	0,16	0,16	0,15	0,13	0,12	0,1	0,12	0,13	0,26	0,26
7	50%	0,15	0,16	0,11	0,13	0,1	0,15	0,15	0,12	0,12	0,26	0,27
8	60%	0,1	0,1	0,1	0,1	0,1	0,14	0,16	0,12	0,13	0,1	0,1
9	70%	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
10	80%	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,11	0,11	0,16	0,16
11	90%	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,11	0,12	0,17	0,18
12	100%	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,11	0,19

A planilha exibirá informações relevantes do veículo e enviará essas informações para o ESP32. Vale destacar que a planilha da Figura 21 não tem fins de controle, ela apenas exibe os dados que no veículo real estariam sendo monitorados por sensores.

Após ser validada, a planilha usada para simular a injeção eletrônica foi transformada em um código em C++. Inicialmente, aquilo que havia sido desenvolvido na planilha foi carregado num microcontrolador responsável pela simulação do veículo, porém ao plotar os valores num gráfico, percebeu-se uma forte discrepância entre os valores simulados e o comportamento de um veículo real.

Foi necessário criar uma simulação tal que os valores não deem saltos que fogem da realidade. Por exemplo, a velocidade de um veículo nunca vai de 1 a 40 em apenas um intervalo de medição muito curto. Assim, foi construída uma simulação que leva em conta o valor anterior e aleatoriamente incrementará ou diminuirá uma determinada variável. Observando o gráfico da Figura 22, conclui-se que o seu perfil se assemelha com o desenvolvimento real da velocidade de um veículo (o mesmo vale para rotação e *Throttle Position Sensor*-TPS).

Figura 22: Progressão das variáveis com ajuste

Como já dito ao longo do trabalho, a simulação em si não é o foco, e assim a título de teste, o comportamento das variáveis foi validado a fim de que esses dados sejam enviados íntegros à nuvem.

O código pode ser encontrado no Anexo D e estará carregado no ESP B, que simula o veículo. Nesse código também é possível notar a presença de variáveis aleatórias. Ao invés de medir a velocidade, a rotação do motor, o sensor lambda e outros parâmetros do veículo, eles são simulados de forma aleatória no código.

Um mapa em um veículo real é escrito com seus parâmetros de fábrica, assim na simulação também há um mapa inicial que cumpre esse papel. Dependendo da leitura da sonda lambda e dos parâmetros de rotação e TPS, um observador ao longe poderia alterar o valor do *offset* a fim de diminuir no mapa esse valor caso a mistura seja rica. Na Figura 23, observa-se esse procedimento funcionando, a leitura do sensor lambda para uma determinada rotação vs TPS indicou mistura rica, assim essa região do mapa foi alterada com o intuito de quando esses valores forem lidos novamente, os novos valores estarão otimizados.

Em um segundo momento, o sensor lambda indicou uma mistura pobre e o *offset* foi calibrado de forma que a alteração fosse de maior amplitude do que a anterior. Para isso, o operador via interface em nuvem observando que a mistura foi pobre aumentou manualmente o valor *offet* e garantiu que o mapa fosse rescrito de forma a corrigir essa imperfeição, aumentando o tempo de injeção, como pode ser visto na Figura 24.

Figura 23: Ajuste do mapa para mistura rica

```
0.00|0.12|0.13|0.27|0.24|0.35|0.68|1.41|1.91|1.25|2.00|3.30|-
0.00|0.15|0.15|0.25|0.25|0.35|0.22|0.61|0.61|1.00|1.20|3.50|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.16|0.26|0.23|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.41|0.83|1.22|0.20|0.30|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.15|0.15|0.25|0.25|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.88|1.00|1.78|2.12|3.25|4.12|-
0.00|0.18|0.16|0.26|0.23|0.35|0.97|1.11|2.12|2.66|3.82|4.30|-
Lambda: 0.80
Offset: 0.10
0.00|0.02|0.03|0.27|0.24|0.35|0.68|1.41|1.91|1.25|2.00|3.30|-
0.00|0.05|0.05|0.25|0.25|0.35|0.22|0.61|0.61|1.00|1.20|3.50|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.16|0.26|0.23|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.41|0.83|1.22|0.20|0.30|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.15|0.15|0.25|0.25|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.88|1.00|1.78|2.12|3.25|4.12|-
0.00|0.18|0.16|0.26|0.23|0.35|0.97|1.11|2.12|2.66|3.82|4.30|-
```

Figura 24: Ajuste do mapa para mistura pobre

```
0.00|0.02|0.03|0.27|0.24|0.35|0.68|1.41|1.91|1.25|2.00|3.30|-
0.00|0.05|0.05|0.25|0.25|0.35|0.22|0.61|0.61|1.00|1.20|3.50|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.16|0.26|0.23|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.41|0.83|1.22|0.20|0.30|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.15|0.15|0.25|0.25|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.88|1.00|1.78|2.12|3.25|4.12|-
0.00|0.18|0.16|0.26|0.23|0.35|0.97|1.11|2.12|2.66|3.82|4.30|-
Lambda: 1.40
Offset: 0.20
0.00|0.02|0.03|0.27|0.24|0.35|0.68|1.41|1.91|1.25|2.00|3.30|-
0.00|0.05|0.05|0.45|0.45|0.35|0.22|0.61|0.61|1.00|1.20|3.50|-
0.00|0.14|0.41|0.82|0.44|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.16|0.26|0.23|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.41|0.83|1.22|0.20|0.30|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.15|0.15|0.25|0.25|0.35|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.14|0.41|0.62|0.24|0.38|0.22|0.10|0.10|0.20|0.20|0.30|-
0.00|0.18|0.11|0.24|0.28|0.36|0.88|1.00|1.78|2.12|3.25|4.12|-
0.00|0.18|0.16|0.26|0.23|0.35|0.97|1.11|2.12|2.66|3.82|4.30|-
```

Vale destacar que este tipo de correção poderia ter sido instalado no veículo sem a necessidade do *cloud computing* que o sistema exige, porém a simulação realizada demonstra que é possível o tráfego de informação descrito no trabalho e uma vez que esse fluxo for escalado a uma frota de veículos, dados de comportamento do mapa e algoritmos de otimização do valor *offset* podem tornar o veículo muito mais eficiente.

O teste de eficiência levando em conta a injeção eletrônica se apresenta como apenas um elemento que pode ser monitorado e controlado, porém não é a única forma de validação da abordagem. Para tanto, uma vez demonstrado que o fluxo de informações é aplicável e que há viabilidade de escrita e leitura de parâmetros durante o funcionamento do veículo, outros parâmetros de comportamento poderiam ser acessados e controlados da mesma forma. Como exemplo, pode-se citar: otimização da pressão dos pneus, pressão atmosférica, temperatura do motor, temperatura ambiente, etc.

6. Conclusão

Abordar um veículo como um sistema ciber-fisico é o novo desafio das grandes montadoras. Um veículo que reaja às mudanças do ambiente poderá não somente otimizar o consumo, mas também prever acidentes, falhas e diminuir o trânsito nas grandes cidades.

O estudo realizado evidenciou que as novas tecnologias como Big Data, Internet das coisas e computação em nuvem estão em ascensão no campo da engenharia e que diversas áreas do conhecimento tendem a aproveitar dos avanços trazidos.

O projeto desenvolvido, apesar de suas limitações, demonstrou que é possível monitorar e controlar parâmetros de um veículo. Mesmo não tendo sido possível a implementação e testes em um veículo real, a proposta deste trabalho tornou mais próxima a implementação de veículos inteligentes. Utilizando ferramentas disponíveis na internet, foi possível simular veículos conectados a servidores remotos. Esses servidores são usados de forma a permitir um fluxo que possibilita que o veículo reaja a comandos via web.

Em trabalhos futuros, seria possível utilizar um serviço de nuvem pago, a fim de conectar APIs que ajudariam no tratamento dos dados e na implementação de uma inteligência por trás da alteração de parâmetros. Além disso, o monitoramento e controle não precisariam se restringir à melhoria do consumo, mas também no conforto do piloto, preservação das peças e prevenção de falhas.

Com a nova tendência de diminuição de veículos próprios e aumento de veículos compartilhados, a indústria de aluguel de automóveis pode aproveitar essa abordagem. Isso porque essas empresas trabalham com veículos de passeios populares e geralmente do mesmo modelo. Monitorando esses veículos e coletando esses dados através de uma nuvem, a empresa pode conhecer não só o produto e como ele reage em diversos tipos de ambiente, mas também conhecer seus clientes: os motoristas. Esse tipo de informação pode ajudar a empresa a cuidar melhor dos seus veículos, desenhá-los sob medida aos seus consumidores e até evitar que alguns tipos de motoristas coloquem em risco o seu produto.

7. Referências

Andrade R. (2014) “Sistemas de comunicação CAN FD: modelamento por software e análise temporal". Disponível em https://teses.usp.br/teses/disponiveis/3/3140/tde-06082015-111553/publico/Dissertacao_Ricardo_rev2_17.pdf

Arduino Cloud IoT (2022) – Disponível em: <https://docs.arduino.cc/cloud/iot-cloud> Acesso em 02/04/2022

Atat R., Liu L., Wu J., Li G., Ye C. and Yang Y. (2018) "Big Data Meet Cyber-Physical Systems: A Panoramic Survey," in IEEE Access, vol. 6, pp. 73603-73636.

Automotive Engineering, (1989). “Automotive electronics: the future?” August 1989 Volume 97, Number 8 – Society of Automotive Engineers, Inc. pp. 26

Banks A., Briggs E., Borgendale K. and Gupta R. “MQTT Version 5.0” March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

Blog Curto Circuito. Disponível em: <https://www.curtocircuito.com.br/blog/conhecendo-esp32> Acesso em 04/05/2020.

Brunetti F. (2003) "Motores de combustão interna - Volume 1".

Calderón A. G., Galbeño Ruiz G. G. G. and Bohórquez A. C. G. (2013), "GPRS telemetry system for high-efficiency electric competition vehicles," 2013 World Electric Vehicle Symposium and Exhibition (EVS27), Barcelona, pp. 1-7.

Colombo A. W., Bangemann T., Karnouskos S., Delsing J., Stluka P., Harrison R., Jammes F., Lastra J.L. (2014) "Industrial cloud-based cyber-physical systems" The IMC-AESOP Approach.

Cook H., Adrian M., Feinberg D. (2019). Magic Quadrant for Operational Database Management Systems, 25 November 2019. Disponivel em: <http://www.oracle.com/br/database/autonomous-database.html> Acesso em 04/05/2020

Corrigan S. (2016) “Introduction to the Controller Area Network (CAN)” 2016 - Application Report, disponível em: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf

Crenshaw T. L., Gunter E., Robinson C. L. et al (2008) "The simplex reference model: Limiting fault-propagation due to unreliable components in Cyber-Physical System architectures " in Proc. of IEEE International Real-Time Systems Symposium 2008.

Fernandes, Nathalia (2021) – “O que é o protocolo MQTT?” – Disponível em: <https://www.hitecnologia.com.br/blog/o-que-e-protocolo-mqtt/> Acesso em 02/04/2022

Grimheden M. E., Törngren M. (2015) "Towards Curricula for Cyber-Physical Systems" Proceedings of the WESE’ 14: Workshop on Embedded and Cyber-Physical Systems Education.

Hu X., Wang H. and Tang X. (2017), "Cyber-Physical Control for Energy-Saving Vehicle Following With Connectivity," in IEEE Transactions on Industrial Electronics, vol. 64, no. 11, pp. 8578-8587.

Husni E., Hertantyo G. B., Wicaksono D. W., Hasibuan F. C., Rahayu A. U. and Triawan M. A. (2016), "Applied Internet of Things (IoT): Car monitoring system using IBM BlueMix," 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA), Lombok, pp. 417-422.

IBM Cloud Learn Hub. Disponível em: <https://www.ibm.com/cloud/learn/paas#toc-paas-iaas--xsjkA5Od> Acesso em 04/05/2020

Kang W. (2009) "Adaptive real-time data management for Cyber-Physical Systems " PhD Thesis University of Virginia 2009.

Kong L. H., Jiang D. W., and Wu M. Y. (2010) “Optimizing the spatio-temporal distribution of Cyber-Physical Systems for environment abstraction,” in Proc. of International Conference on Distributed Computing Systems.

Kottenstette N., Karsai G., and Sztipanovits J. (2009) “A passivity-based framework for resilient Cyber Physical Systems,” in Proc. of 2nd International Symposium on Resilient Control Systems.

Koutsoukos X., Kottenstette N., Hall J. et al (2008) "Passivity-based control design for Cyber-Physical Systems". Available at: http://citeseerx.ist.psu.edu/.

Koyanagi, F. (2018). IBM Watson com ESP32 como Endpoint. Disponível em: <https://www.fernandok.com/2018/0/ibm-watson-com-esp32-como-endpoint.html> Acesso em 04/05/2020.

Kremer U. (2013) “Cyber-Physical Systems: A case for soft real-time,” Available at: http://www.research.rutgers.edu/.

Lakshmanan K., Niz D., Rajkumar R., Moreno G. (2010) “Resource allocation in distributed mixed-criticality Cyber-Physical Systems,” in Proc. of International Conference on Distributed Computing Systems.

Leitão, P.; Colombo, A. W.; Karnouskos, S. (2016), Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges, Computers in Industry, Vol. 81, pp. 11-25.

Li K. W., Liu Q. W., Wang F. R., Xie X. (2009) “Joint optimal congestion control and channel assignment for multi-radio multi-channel wireless networks in Cyber-Physical Systems,” in Proc. of Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing.

Liberatore V. (2010), “Bandwidth allocation in sense-and-respond systems,” Report, Available at: http://home.case.edu/~vxl11/ NetBots/.

Lindberg M., and Årzén K. E. (2010) “Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties,” in Proc. of the 31st IEEE Real-Time Systems Symposium.

Liu Y., Xie G., Jin L. and Li R. (2019), "Energy Optimization on Dynamic Multiple Functions Automotive Cyber-Physical Systems," 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, pp. 857-864.

Longbottom, Clive (2011). "IBM gets smart with cloud". CloudPro. Retrieved 2 September 2011.

Mäkiö-Marusik E., Ahmad B., Harrison R., Mäkiö J. and Colombo A. W. (2018) "Competences of cyber physical systems engineers — Survey results," 2018 IEEE Industrial Cyber-Physical Systems (ICPS), St. Petersburg, pp. 491-496.

Naik P., Kumbi A., Telkar N., Kotin K. and Katti K. C. (2017), "An automotive diagnostics, fuel efficiency and emission monitoring system using CAN," 2017 International Conference on Big Data, IoT and Data Science (BID), Pune, pp. 14-17.

Oliveira, S. de (2017). “Internet das coisas com ESP8266, Arduino e Raspberry Pi”, Novatec Editora Ltda, pp. 49-53.

Parolini L., Toliaz N., Sinopoli B., and Krogh B. H. (2010) "A Cyber-Physical Systems approach to energy management in data centers " in Proc. of First International Conference on Cyber-Physical Systems. Stockholm Sweden.

Rawung R. H. and Putrada A. G. (2014), "Cyber physical system: Paper survey," 2014 International Conference on ICT For Smart Society (ICISS), Bandung, pp. 273-278.

Safonov, V. O. (2016). Trustworthy Cloud Computing. John Wiley & Sons. ISBN 9781119113515.

Samanta B. (2012) "Multidisciplinary education and research using computational intelligence" 2012 Proceedings of IEEE Southeastcon pp. 1-6.

Sami M., Malek M., Bondi U., Regazzoni F. (2017) "Embedded Systems Education: Job Market Expectations" SIGBED Rev. vol. 14 pp. 22-28.

Santos, R. (2018). ESP32 Web Server – Arduino IDE. Disponível em: <https://randomnerdtutorials.com/esp32-web-server-arduino-ide/> Acesso em 04/05/2020.

Shi J., Wan J., Yan H. and Suo H. (2011) "A survey of Cyber-Physical Systems," 2011 International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, pp. 1-6.

Tan Y., Vuran M. C., and Goddard S. (2010) "A concept lattice-based event model for Cyber-Physical Systems " in Proc. of CCPS Apr 2010 Stockholm Sweden.

Tang Q. H., Gupta S. K. S., and Varsamopoulos G. (2008) “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach,” IEEE Transactions on Parallel and Distributed Systems, Vol. 19, pp. 1458-1472.

Thacker R. A., Jones K. R., Myers C. J., Zheng H. (2010) "Automatic abstraction for verification of Cyber-Physical Systems " in Proc. of CCPS, Stockholm Sweden.

The HiveMQ Team (2015) – “Introducing the MQTT Protocol - MQTT Essentials: Part 1” – Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/> Acesso em 02/04/2022

Tidwell T., Gao X. Y., Huang H. M., Lu C., Dyke S., Gill C. (2009) "Towards configurable realtime hybrid structural testing: A Cyber-Physical Systems approach " in Proc. of IEEE International Symposium on Object/ Component/Service-Oriented Real-Time Distributed Computing.

Törngren M., Grimheden M.E., Gustafsson J., Birk W. (2017) "Strategies and Considerations in Shaping Cyber-physical Systems Education" SIGBED Rev. vol. 14 pp. 53-60.

Törngren M., Herzog E. (2016) "Towards Integration of CPS and Systems Engineering in Education" Proceedings of the 2016 Workshop on Embedded and Cyber-Physical Systems Education.

Trinks S. and Felden C. (2017) "Real time analytics — State of the art: Potentials and limitations in the smart factory," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, pp. 4843-4845.

Wade J., Cohen R., Blackburn M., Hole E. and Bowen N. (2015) "Systems Engineering of Cyber-Physical Systems Education Program" Proceedings of the WESE' 15: Workshop on Embedded and Cyber-Physical Systems Education.

Wang Q., Lei Z. and Qu S. (2012), "Design of car remote monitoring system based on Internet," 2012 Proceedings of International Conference on Modelling, Identification and Control, Wuhan, Hubei, China, pp. 631-635.

Wang, S.; Zhang, Y.; Yang, Z. (2016), A Graphical Hierarchical CPS Architecture, In: International Symposium on System and Software Reliability (ISSSR), pp. 97-105.

Zhang F. M., Szwaykowska K., Wolf W. and Mooney V. (2005) "Task scheduling for control oriented requirements for Cyber-Physical Systems" in Proc. of 2008 Real-Time Systems Symposium 2005 pp. 47-56.

Zhao S., Li S., Chen L., Ding Y., Zheng Z. and Pan G. (2013), "iCPS-Car: An Intelligent Cyber-physical System for Smart Automobiles," 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, pp. 818-825.

Zhu Y., Westbrook E., Inoue J., Chapoutot A., Salama C., Peralta M., Martin T., Taha W., O'Malley M., Cartwright R., Ames A., Bhattacharya R. (2010) "Mathematical equations as executable models of mechanical systems," in Proc. of CCPS, Stockholm, Sweden.

Anexo A

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid     = "";
const char* password = "";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output26, OUTPUT);
  pinMode(output27, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output26, LOW);
  digitalWrite(output27, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
43.   }
44.   // Print local IP address and start web server
45.   Serial.println("");
46.   Serial.println("WiFi connected.");
47.   Serial.println("IP address: ");
48.   Serial.println(WiFi.localIP());
49.   server.begin();
50. }
51.
52. void loop(){
53.   WiFiClient client = server.available();   // Listen for incoming clients
54.
55.   if (client) {                             // If a new client connects,
56.     Serial.println("New Client.");          // print a message out in the serial por
    t
57.     String currentLine = "";                // make a String to hold incoming data f
    rom the client
58.     while (client.connected()) {            // loop while the client's connected
59.       if (client.available()) {             // if there's bytes to read from the cli
    ent,
60.         char c = client.read();             // read a byte, then
61.         Serial.write(c);                    // print it out the serial monitor
62.         header += c;
63.         if (c == '\n') {                    // if the byte is a newline character
64.           // if the current line is blank, you got two newline characters in a row.

65.           // that's the end of the client HTTP request, so send a response:
66.           if (currentLine.length() == 0) {
67.             // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)

68.             // and a content-
    type so the client knows what's coming, then a blank line:
69.             client.println("HTTP/1.1 200 OK");
70.             client.println("Content-type:text/html");
71.             client.println("Connection: close");
72.             client.println();
73.
74.             // turns the GPIOs on and off
75.             if (header.indexOf("GET /26/on") >= 0) {
76.               Serial.println("GPIO 26 on");
77.               output26State = "on";
78.               digitalWrite(output26, HIGH);
79.             } else if (header.indexOf("GET /26/off") >= 0) {
80.               Serial.println("GPIO 26 off");
```

```
81.               output26State = "off";
82.               digitalWrite(output26, LOW);
83.             } else if (header.indexOf("GET /27/on") >= 0) {
84.               Serial.println("GPIO 27 on");
85.               output27State = "on";
86.               digitalWrite(output27, HIGH);
87.             } else if (header.indexOf("GET /27/off") >= 0) {
88.               Serial.println("GPIO 27 off");
89.               output27State = "off";
90.               digitalWrite(output27, LOW);
91.             }
92.
93.             // Display the HTML web page
94.             client.println("<!DOCTYPE html><html>");
95.             client.println("<head><meta name=\"viewport\" content=\"width=device-
    width, initial-scale=1\">");
96.             client.println("<link rel=\"icon\" href=\"data:,\">");
97.             // CSS to style the on/off buttons
98.             // Feel free to change the background-color and font-
    size attributes to fit your preferences
99.             client.println("<style>html { font-family: Helvetica; display: inline-
    block; margin: 0px auto; text-align: center;}");
100.                    client.println(".button { background-
    color: #4CAF50; border: none; color: white; padding: 16px 40px;");
101.                    client.println("text-decoration: none; font-
    size: 30px; margin: 2px; cursor: pointer;}");
102.                    client.println(".button2 {background-
    color: #555555;}</style></head>");
103.
104.                    // Web Page Heading
105.                    client.println("<body><h1>ESP32 Web Server</h1>");
106.
107.                    // Display current state, and ON/OFF buttons for GPIO 26
108.                    client.println("<p>GPIO 26 - State " + output26State + "</p>");
109.                    // If the output26State is off, it displays the ON button

110.                    if (output26State=="off") {
111.                      client.println("<p><a href=\"/26/on\"><button class=\"button\">
    ON</button></a></p>");
112.                    } else {
113.                      client.println("<p><a href=\"/26/off\"><button class=\"button b
    utton2\">OFF</button></a></p>");
114.                    }
115.
```

```
116.                   // Display current state, and ON/OFF buttons for GPIO 27
117.                   client.println("<p>GPIO 27 - State " + output27State + "</p>");
118.                   // If the output27State is off, it displays the ON button
119.                   if (output27State=="off") {
120.                     client.println("<p><a href=\"/27/on\"><button class=\"button\">
    ON</button></a></p>");
121.                   } else {
122.                     client.println("<p><a href=\"/27/off\"><button class=\"button b
    utton2\">OFF</button></a></p>");
123.                   }
124.                   client.println("</body></html>");
125.
126.                   // The HTTP response ends with another blank line
127.                   client.println();
128.                   // Break out of the while loop
129.                   break;
130.                 } else { // if you got a newline, then clear currentLine
131.                   currentLine = "";
132.                 }
133.               } else if (c != '\r') {  // if you got anything else but a carriage r
    eturn character,
134.                 currentLine += c;      // add it to the end of the currentLine
135.               }
136.             }
137.           }
138.           // Clear the header variable
139.           header = "";
140.           // Close the connection
141.           client.stop();
142.           Serial.println("Client disconnected.");
143.           Serial.println("");
144.         }
145.       }
```

Anexo B

```
1.  //Verifica qual ESP está sendo utilizado
2.  //e importa a lib e wifi correspondente
3.  #if defined(ESP8266)
4.  #include <ESP8266WiFi.h>
5.  #else
6.  #include <WiFi.h>
7.  #endif
8.
9.  //Lib de MQTT
10. #include <PubSubClient.h>
11.
12. //Intervalo entre os envios
13. #define INTERVAL 1000
14.
15. //Substitua pelo SSID da sua rede
16. #define SSID "Pedro"
17.
18. //Substitua pela senha da sua rede
19. #define PASSWORD "12345678"
20.
21. //Server MQTT que iremos utlizar
22. #define MQTT_SERVER "quickstart.messaging.internetofthings.ibmcloud.com"
23.
24. //Nome do tópico que devemos enviar os dados
25. //para que eles apareçam nos gráficos
26. #define TOPIC_NAME "iot-2/evt/status/fmt/json"
27.
28. //ID que usaremos para conectar
29. //QUICK_START deve permanecer como está
30. const String QUICK_START = "d:quickstart:arduino:";
31.
32. //No DEVICE_ID você deve mudar para um id único
33. //Aqui nesse exemplo utilizamos o MAC Address
34. //do dispositivo que estamos utilizando
35. //Servirá como identificação no site
36. //https://quickstart.internetofthings.ibmcloud.com
37. const String DEVICE_ID = "240ac40e3fd0";
38.
39. //Concatemos o id do quickstart com o id do nosso
40. //dispositivo
41. const String CLIENT_ID =  QUICK_START + DEVICE_ID;
42.
```

```
43. //Cliente WiFi que o MQTT irá utilizar para se conectar
44. WiFiClient wifiClient;
45. 
46. //Cliente MQTT, passamos a url do server, a porta
47. //e o cliente WiFi
48. PubSubClient client(MQTT_SERVER, 1883, wifiClient);
49. 
50. //Tempo em que o último envio foi feito
51. long lastPublishTime = 0;
52. 
53. int sensorPin = 32;
54. 
55. //Variável para guardarmos o valor da temperatura
56. float temperature = 0;
57. 
58. void setup() {
59.   Serial.begin(115200);
60.   pinMode(sensorPin,INPUT);
61.   //Conectamos à rede WiFi
62.   setupWiFi();
63.   //Conectamos ao server MQTT
64.   connectMQTTServer();
65. }
66. 
67. void loop() {
68.   //Tempos agora em milisegundos
69.   long now = millis();
70. 
71.   //Se o tempo desde o último envio for maior que o intervalo de envio
72.   if (now - lastPublishTime > INTERVAL) {
73.     //Atualizamos o tempo em que ocorreu o último envio
74.     lastPublishTime = now;
75.     //Fazemos a leitura da temperatura e umidade
76.     readSensor();
77.     Serial.print("Publish message: ");
78.     //Criamos o json que enviaremos para o server mqtt
79.     String msg = createJsonString();
80.     Serial.println(msg);
81.     //Publicamos no tópico onde o servidor espera para receber
82.     //e gerar o gráfico
83.     client.publish(TOPIC_NAME, msg.c_str());
84.   }
85. }
86. 
```

```
87. //Função responsável por conectar à rede WiFi
88. void setupWiFi() {
89.   Serial.println();
90.   Serial.print("Connecting to ");
91.   Serial.print(SSID);
92.
93.   //Manda o esp se conectar à rede através
94.   //do ssid e senha
95.   WiFi.begin(SSID, PASSWORD);
96.
97.   //Espera até que a conexão com a rede seja estabelecida
98.   while (WiFi.status() != WL_CONNECTED) {
99.     delay(500);
100.          Serial.print(".");
101.        }
102.
103.        //Se chegou aqui é porque conectou
104.        Serial.println("");
105.        Serial.println("WiFi connected");
106.      }
107.
108.      //Função responsável por conectar ao server MQTT
109.      void connectMQTTServer() {
110.        Serial.println("Connecting to MQTT Server...");
111.        //Se conecta ao id que definimos
112.        if (client.connect(CLIENT_ID.c_str())) {
113.          //Se a conexão foi bem sucedida
114.          Serial.println("connected");
115.        } else {
116.          //Se ocorreu algum erro
117.          Serial.print("error = ");
118.          Serial.println(client.state());
119.        }
120.      }
121.
122.      //Função responsável por realizar a leitura
123.      //da temperatura e umidade
124.      void readSensor(){
125.        float value;
126.        //Faz a leitura da temperatura
127.        value = analogRead(sensorPin);
128.        temperature = ((value/1024)*3300)*0.1;
129.      }
130.
```

```
131.      //Função responsável por criar
132.      //um Json com os dados lidos
133.      String createJsonString() {
134.        String data = "{";
135.          data+= "\"d\": {";
136.            data+="\"temperature\":";
137.            data+=String(temperature);
138.          data+="}";
139.        data+="}";
140.        return data;
141.      }
```

Anexo C

```
1.  Sub Button2_Click()
2.  Chama_Aleatorios
3.  Pinta_Mapa
4.  Altera_Mapa
5.  End Sub
6.
7.  Function Gera_Aleatorio(s As Double, i As Double)
8.  Gera_Aleatorio = ((s - i + 1) * Rnd + i)
9.  End Function
10.
11. Sub Chama_Aleatorios()
12.     Worksheets("Entrada").Range("A1").Select
13.     ActiveCell.FormulaR1C1 = "=Gera_Aleatorio(10000,0)"
14.     Worksheets("Entrada").Range("A2").Select
15.     ActiveCell.FormulaR1C1 = "=Gera_Aleatorio(0.10000,0)"
16.     Worksheets("Entrada").Range("A3").Select
17.     ActiveCell.FormulaR1C1 = "=Gera_Aleatorio(2,0)"
18. End Sub
19.
20. Sub Altera_Mapa()
21. Dim Rotacao As Integer, Aceleracao As Double, Lambda As Double, Beep As Double
22. Rotacao = Worksheets("Entrada").Range("A1")
23. Aceleracao = Worksheets("Entrada").Range("A2")
24. Lambda = Worksheets("Entrada").Range("A3")
25. Beep = Worksheets("Entrada").Range("A4")
26. result_coluna = Verifica_Coluna(Rotacao)
27. coluna = result_coluna(0)
28. coluna_plus = result_coluna(1)
29. result_linha = Verifica_Linha(Aceleracao)
30. linha = result_linha(0)
31. linha_plus = result_linha(1)
32. Worksheets("Mapa").Range(Cells(linha + 2, coluna + 2), Cells(linha + 2 + linha_plus,
     2 + coluna + coluna_plus)).Select
33. If coluna_plus = 0 And linha_plus = 0 Then
34.     Worksheets("Mapa").Cells(linha + 2, coluna + 2).Value = Cells(linha + 2, coluna
    + 2).Value + Beep
35. ElseIf coluna_plus = 0 And linha_plus <> 0 Then
36.     Worksheets("Mapa").Cells(linha + 2, coluna + 2).Value = Cells(linha + 2, coluna
    + 2).Value + Beep
37.     Worksheets("Mapa").Cells(linha + 2 + linha_plus, coluna + 2).Value = (Cells(linh
    a + 2 + linha_plus, coluna + 2).Valu + Beep)
38. ElseIf coluna_plus <> 0 And linha_plus = 0 Then
```

```
39.     Worksheets("Mapa").Cells(linha + 2, coluna + 2).Value = (Cells(linha + 2, coluna
     + 2).Value + Beep)
40.     Worksheets("Mapa").Cells(linha + 2, coluna_plus + coluna + 2).Value = (Cells(lin
    ha + 2, coluna_plus + coluna + 2).Value + Beep)
41. ElseIf coluna_plus <> 0 And linha_plus <> 0 Then
42.     Worksheets("Mapa").Cells(linha + 2, coluna + 2).Value = Cells(linha + 2, coluna
    + 2).Value + Beep
43.     Worksheets("Mapa").Cells(linha + 2 + linha_plus, coluna + 2).Value = (Cells(linh
    a + 2 + linha_plus, coluna + 2).Value + Beep)
44.     Worksheets("Mapa").Cells(linha + 2, coluna_plus + coluna + 2).Value = (Cells(lin
    ha + 2, coluna_plus + coluna + 2).Value + Beep)
45.     Worksheets("Mapa").Cells(linha + 2 + linha_plus, coluna_plus + coluna + 2).Value
     = (Cells(linha + 2, coluna_plus + coluna + 2).Value + Beep)
46. End If
47. End Sub
48.
49. Sub Pinta_Mapa()
50. Dim Rotacao As Integer, Aceleracao As Double
51. Rotacao = Worksheets("Entrada").Range("A1")
52. Aceleracao = Worksheets("Entrada").Range("A2")
53. Sheets("Mapa").Select
54.     Cells.Select
55.     With Selection.Interior
56.         .Pattern = xlNone
57.         .TintAndShade = 0
58.         .PatternTintAndShade = 0
59.     End With
60.     Range("A1").Select
61. result_coluna = Verifica_Coluna(Rotacao)
62. coluna = result_coluna(0)
63. coluna_plus = result_coluna(1)
64. result_linha = Verifica_Linha(Aceleracao)
65. linha = result_linha(0)
66. linha_plus = result_linha(1)
67. Worksheets("Mapa").Range(Cells(linha + 2, coluna + 2), Cells(linha + 2 + linha_plus,
     2 + coluna + coluna_plus)).Select
68. With Selection.Interior
69.         .Pattern = xlSolid
70.         .PatternColorIndex = xlAutomatic
71.         .Color = 65535
72.         .TintAndShade = 0
73.         .PatternTintAndShade = 0
74.     End With
75. End Sub
```

```
76.
77. Function Verifica_Coluna(Rotacao As Integer) As Variant()
78. Dim a As Integer, b As Integer
79. If Rotacao = 0 Then
80. a = 0
81. b = 0
82. ElseIf Rotacao > 0 And Rotacao < 1000 Then
83. a = 0
84. b = 1
85. ElseIf Rotacao = 1000 Then
86. a = 1
87. b = 0
88. ElseIf Rotacao > 1000 And Rotacao < 2000 Then
89. a = 1
90. b = 1
91. ElseIf Rotacao = 2000 Then
92. a = 2
93. b = 0
94. ElseIf Rotacao > 2000 And Rotacao < 3000 Then
95. a = 2
96. b = 1
97. ElseIf Rotacao = 3000 Then
98. a = 3
99. b = 0
100.        ElseIf Rotacao > 3000 And Rotacao < 4000 Then
101.        a = 3
102.        b = 1
103.        ElseIf Rotacao = 4000 Then
104.        a = 4
105.        b = 0
106.        ElseIf Rotacao > 4000 And Rotacao < 5000 Then
107.        a = 4
108.        b = 1
109.        ElseIf Rotacao = 5000 Then
110.        a = 5
111.        b = 0
112.        ElseIf Rotacao > 5000 And Rotacao < 6000 Then
113.        a = 5
114.        b = 1
115.        ElseIf Rotacao = 6000 Then
116.        a = 6
117.        b = 0
118.        ElseIf Rotacao > 6000 And Rotacao < 7000 Then
119.        a = 6
```

```
120.        b = 1
121.        ElseIf Rotacao = 7000 Then
122.        a = 7
123.        b = 0
124.        ElseIf Rotacao > 7000 And Rotacao < 8000 Then
125.        a = 7
126.        b = 1
127.        ElseIf Rotacao = 8000 Then
128.        a = 8
129.        b = 0
130.        ElseIf Rotacao > 8000 And Rotacao < 9000 Then
131.        a = 9
132.        b = 1
133.        ElseIf Rotacao = 9000 Then
134.        a = 9
135.        b = 0
136.        ElseIf Rotacao > 9000 And Rotacao < 10000 Then
137.        a = 9
138.        b = 1
139.        ElseIf Rotacao = 10000 Then
140.        a = 10
141.        b = 0
142.        End If
143.        Verifica_Coluna = Array(a, b)
144.        End Function
145.
146.        Function Verifica_Linha(Aceleracao As Double) As Variant()
147.        Dim a As Integer, b As Integer
148.        If Aceleracao = 0 Then
149.        a = 0
150.        b = 0
151.        ElseIf Aceleracao > 0 And Aceleracao < 0.1 Then
152.        a = 0
153.        b = 1
154.        ElseIf Aceleracao = 0.1 Then
155.        a = 1
156.        b = 0
157.        ElseIf Aceleracao > 0.1 And Aceleracao < 0.2 Then
158.        a = 1
159.        b = 1
160.        ElseIf Aceleracao = 0.2 Then
161.        a = 2
162.        b = 0
163.        ElseIf Aceleracao > 0.2 And Aceleracao < 0.3 Then
```

```
164.        a = 2
165.        b = 1
166.        ElseIf Aceleracao = 0.3 Then
167.        a = 3
168.        b = 0
169.        ElseIf Aceleracao > 0.3 And Aceleracao < 0.4 Then
170.        a = 3
171.        b = 1
172.        ElseIf Aceleracao = 0.4 Then
173.        a = 4
174.        b = 0
175.        ElseIf Aceleracao > 0.4 And Aceleracao < 0.5 Then
176.        a = 4
177.        b = 1
178.        ElseIf Aceleracao = 0.5 Then
179.        a = 5
180.        b = 0
181.        ElseIf Aceleracao > 0.5 And Aceleracao < 0.6 Then
182.        a = 5
183.        b = 1
184.        ElseIf Aceleracao = 0.6 Then
185.        a = 6
186.        b = 0
187.        ElseIf Aceleracao > 0.6 And Aceleracao < 0.7 Then
188.        a = 6
189.        b = 1
190.        ElseIf Aceleracao = 0.7 Then
191.        a = 7
192.        b = 0
193.        ElseIf Aceleracao > 0.7 And Aceleracao < 0.8 Then
194.        a = 7
195.        b = 1
196.        ElseIf Aceleracao = 0.8 Then
197.        a = 8
198.        b = 0
199.        ElseIf Aceleracao > 0.8 And Aceleracao < 0.9 Then
200.        a = 8
201.        b = 1
202.        ElseIf Aceleracao = 0.9 Then
203.        a = 9
204.        b = 0
205.        ElseIf Aceleracao > 0.9 And Aceleracao < 1# Then
206.        a = 9
207.        b = 1
```

```
208.        ElseIf Aceleracao = 1# Then
209.        a = 10
210.        b = 0
211.        End If
212.        Verifica_Linha = Array(a, b)
213.        End Function
214.
```

Anexo D – ESP B

```
1.  // DEFININDO MAPA INICIAL
2.  float mapa[11][12]={
3.                      {0.00, 0.12, 0.13, 0.27, 0.24, 0.35, 0.68, 1.41, 1.91, 1.25,
    2.00, 3.30},
4.                      {0.00, 0.15, 0.15, 0.25, 0.25, 0.35, 0.22, 0.61, 0.61, 1.0, 1.2,
    3.50},
5.                      {0.00, 0.14, 0.41, 0.62, 0.24, 0.38, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.30},
6.                      {0.00, 0.18, 0.11, 0.24, 0.28, 0.36, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.30},
7.                      {0.00, 0.18, 0.11, 0.24, 0.28, 0.36, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.30},
8.                      {0.00, 0.18, 0.16, 0.26, 0.23, 0.35, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.3},
9.                      {0.00, 0.41, 0.83, 1.22, 0.2, 0.3, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.3},
10.                     {0.00, 0.15, 0.15, 0.25, 0.25, 0.35, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.3},
11.                     {0.00, 0.14, 0.41, 0.62, 0.24, 0.38, 0.22, 0.1, 0.1, 0.2, 0.2,
    0.3},
12.                     {0.00, 0.18, 0.11, 0.24, 0.28, 0.36, 0.88, 1.00, 1.78, 2.12,
    3.25, 4.12},
13.                     {0.00, 0.18, 0.16, 0.26, 0.23, 0.35, 0.97, 1.11, 2.12, 2.66,
    3.82, 4.30}
14.                   };
15. float lambdaSensor;
16. float off = 0.1;
17. float rotation;
18. float velocidade;
19. float tps_acel;
20. float linha;
21. float coluna;
22. int r_linha;
23. int r_coluna;
24.
25. void setup() {
26.   // put your setup code here, to run once:
27.   Serial.begin(9600);
28. }
29.
30. void loop() {
31.   // put your main code here, to run repeatedly:
```

```
32.   lambdaSensor = random(0, 200);
33.   lambdaSensor = lambdaSensor/100;
34.   rotation = random(0, 11001);
35.   tps_acel = random(0, 101);
36.   velocidade=random(0,61);
37.
38.   linha = rotation/1000;
39.   coluna = tps_acel/10;
40.
41.   if (lambdaSensor<1)
42.   {
43.     r_linha = round(linha * 10) / 10.0;
44.     r_coluna = round(coluna * 10) / 10.0;
45.     mapa[r_linha][r_coluna] = mapa[r_linha][r_coluna] - off;
46.     mapa[r_linha+1][r_coluna+1] = mapa[r_linha+1][r_coluna+1] - off;
47.     mapa[r_linha][r_coluna+1] = mapa[r_linha][r_coluna+1] - off;
48.     mapa[r_linha+1][r_coluna] = mapa[r_linha+1][r_coluna] - off;
49.
50.   }
51.   else if (lambdaSensor>1)
52.   {
53.     r_linha = round(linha * 10) / 10.0;
54.     r_coluna = round(coluna * 10) / 10.0;
55.     mapa[r_linha][r_coluna] = mapa[r_linha][r_coluna] + off;
56.     mapa[r_linha+1][r_coluna+1] = mapa[r_linha+1][r_coluna+1] + off;
57.     mapa[r_linha+1][r_coluna] = mapa[r_linha+1][r_coluna] + off;
58.     mapa[r_linha][r_coluna+1] = mapa[r_linha][r_coluna+1] + off;
59.   }
60.
61.   String rot = String(rotation,0);
62.   while (6-rot.length()!= 0){
63.     rot = "0"+rot;
64.   }
65.
66.   String vel = String(velocidade,0);
67.   while (2-vel.length()!= 0){
68.     vel = "0"+vel;
69.   }
70.   String tps = String(tps_acel,0);
71.   while (2-tps.length()!= 0){
72.     vel = "0"+tps;
73.   }
74.
75.   String lambda = String(lambdaSensor,2);
```

```
76.
77.   String offset = String(off,1);
78.
79.   String junto=rot+vel+tps+lambda+offset;
80.   Serial.print(junto);
81.   delay(1500);
82.
83. }
```

Anexo E – ESP A

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Untitled"
  https://create.arduino.cc/cloud/things/72a9dd94-761f-4da9-b5cc-c9d3798e48b8

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to the Thing

  float offset;
  int tps;
  int vel;
  float lambda;
  int rot;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

#include "thingProperties.h"

int pot = 35;
int valor;
String msg;


void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  pinMode(pot, INPUT);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

```

```
  /*
     The following function allows you to obtain more information
     related to the state of network and IoT Cloud connection and errors
     the higher number the more granular information you'll get.
     The default is 0 (only errors).
     Maximum is 4
 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();
  // Your code here
  //vel=analogRead(pot);

  if(Serial.available()>0){
    msg = (Serial.readString());
  }

  String rot_aux = msg.substring(0,6);
  String vel_aux = msg.substring(6,8);
  String tps_aux = msg.substring(8,10);
  String lambda_aux = msg.substring(10,14);
  String offset_aux = msg.substring(14);

  rot = rot_aux.toInt();
  vel = vel_aux.toInt();
  tps = tps_aux.toInt();
  lambda = lambda_aux.toFloat();
  offset = offset_aux.toFloat();
}


/*
  Since Mapa is READ_WRITE variable, onMapaChange() is
  executed every time a new value is received from IoT Cloud.
*/
void onMapaChange()  {
  // Add your code here to act upon Mapa change
}


/*
```

```
84.   Since Offset is READ_WRITE variable, onOffsetChange() is
85.   executed every time a new value is received from IoT Cloud.
86. */
87. void onOffsetChange()  {
88.   // Add your code here to act upon Offset change
89.   Serial.println(offset);
90. }
91.
```