Relatório de Trabalho

Project Factory

João Coelho, nº 20220753
Ricardo Dias, nº 20220494

Introdução

Este documento serve como relatório para o projeto da cadeira *Project Factory* e do 6º Semestre da licenciatura de Engenharia Informática. O mesmo irá sofrer alterações conforme o desenvolver do projeto, tendo as mesmas a intuição de acompanhar o progresso e requisitos das *miletstones* presentes no *briefing* de projeto.

Como breve introdução, o projeto envolve a construção de um veículo capas de transportar carga ao longo de diversos obstáculos. Este veiculo também será capas de comunicar com uma API externa. Os equipamentos necessários para a construção dos mesmos estão ao nosso critério, tal como a estrutura e construção do veiculo.

Pesquisa

De forma a organizar a nossa pesquisa, decidimos focarmos-nos em projetos parecidos, já feitos antes, e, principalmente, projetos que já demonstrem conseguir passar aos obstáculos definidos no *briefing*. Os obstáculos definidos são: “Rampa, desenhada para provocar um salto”, “Superfície irregular, desenhada para provocar vibração no veículo” e “Paredes de bloqueio, desenhadas para obrigar a contornar um obstáculo”. Ou seja, para além do *know how* base de criar um veículo com um microcontrolador, também procuramos soluções para o veiculo conseguir: subir uma rampa e saltar dela, suportar chão irregular e altas vibrações e contornar obstáculos.

Ao pesquisar sobre outros projetos de carros telecomandados e segundo algumas indicações de professores, descobrimos que:

- Será melhor fazer a comunicação do carro com a API externa com protocolos menos “pesados” que http (mqtt ou tcp socktes)
- Existem frameworks de desenvolvimento de API’s de forma rápida, que garantem a integração de um cliente mqtt e dashboards (Node Red).
- Será necessario um mqtt broker para realizar comunicação mqtt (ex: Mosquito)
- Em termos de hardware, foi recomendado à turma usar um ESP32
- As especificações “normais" de um motor seriam: de corrente 150ma, com 12V e 1800rpm de máximo e 500rpm mínimo.
- Será necessario um driver para controlar os motores.
- É indicado usarmos sensores ultra-sónicos de forma ao carro ter uma noção do seu redor.

Ao pesquisar sobre formas de subir uma rampa e saltar dela com o veiculo, descobrimos que:

- Necessitamos de torque suficiente nos motores para subir uma inclinação.
- Necessitamos de velocidade suficiente ao subir, para o veiculo não ficar preso no pico da rampa.
- Precisamos de conseguir recentrar o veiculo após o salto.

Ao pesquisar sobre formas de suportar chão irregular e altas vibrações, descobrimos que:

- Podemos inspirar-nos em veículos todo terreno, que têm o chassi acima dos veículos normais.
- Precisamos de o veículo seja construído com materiais resistentes a vibrações, tal como plástico ou madeira, e usar parafusos ou encaixes de forma á estrutura não se desfazer no percurso.

Ao pesquisar sobre formas de contornar obstáculos, descobrimos que:

- Existem rodas omni-direcionais, mais especificamente as *Mecanum*, que não requerem virar o chassi do carro para o mesmo andar para o lado. Por outro lado estas rodas requerem maior torque que as normais.

Requisitos

Após analisar os requisitos do projeto, decidimos definir os requisitos da nossa solução. Como tal, segue uma lista das funcionalidades, relativas a cada "agente" da nossa estrutura, que vão preencher os requisitos do *briefing*:

Cliente do Veículo:

- Enviar a percentagem de bateria, via mqtt.
- Enviar a velocidade instantánea, via mqtt.
- Enviar a Inclinação, via mqtt.
- Detetar se está dentro do percurso.
- Enviar o momento em que começou e acabou um percurso, via mqtt.
- Receber e executar ações de movimento, via mqtt.
- Executar movimentos predefinidos (ex: fazer 360º, donuts, etc...)
- Condução autónoma:
  - Iniciar e terminar uma sessão de condução autónoma, via pedido mqtt e botão integrado.
  - Enviar a informação lida pelos sensores (sensores ultrasónicos, sensor de troca de cor, etc) necessário para o planeamento do caminho a fazer.
  - Receber e executar ações de movimento, via mqtt.

API:

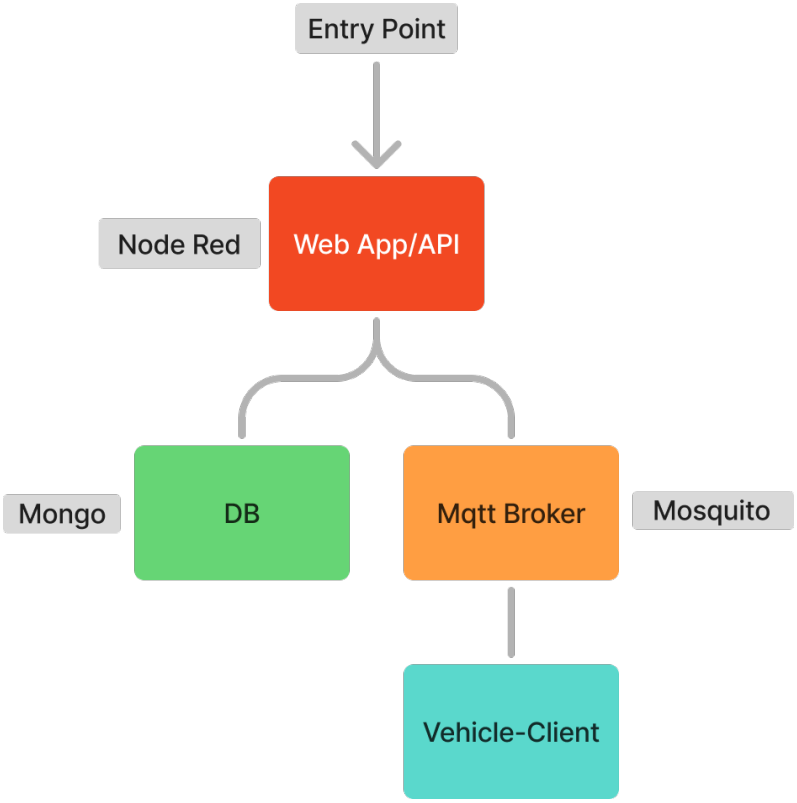
- Guardar e enviar histórico de velocidades instantâneas.
- Guardar e enviar historio de viagens realizadas.
- Enviar Instruções de movimento ao veiculo.
- Enviar instruções para executar movimentos predefinidos.
- Condução autónoma:
  - Começar uma sessão de condução autónoma.
  - Processar informação e calcular movimentos a realizar.
  - Enviar movimentos a realizar.

WebApp:

- Dashboard:
  - Mostrar a percentagem atual de bateria.
  - Mostrar o histórico de velocidades instantâneas.
  - Mostrar a inclinação atual.
  - Mostrar o histórico de viagens.
  - Controlo Remoto (8 direções) e movimento executar movimentos predefinidos.
  - Iniciar uma sessão de condução autónoma.

Infraestrutura

De forma a definir a nossa infraestrutura computacional, decidimos desenhar o diagrama representado abaixo:



Tal como podemos observar existem quatro agentes: a Web App/API, a DB (Base de Dados), o mqtt Broker e o Vehicle-Client.

A Web App/Api será uma plataforma realizada em Node Red, que receberá e enviará informação do veiculo, podendo guardar a mesma na base de dados. Decidimos juntar a Web App com a API de forma a simplificar a infraestrutura e aproveitar os módulos de *dashboards* do Node Red.

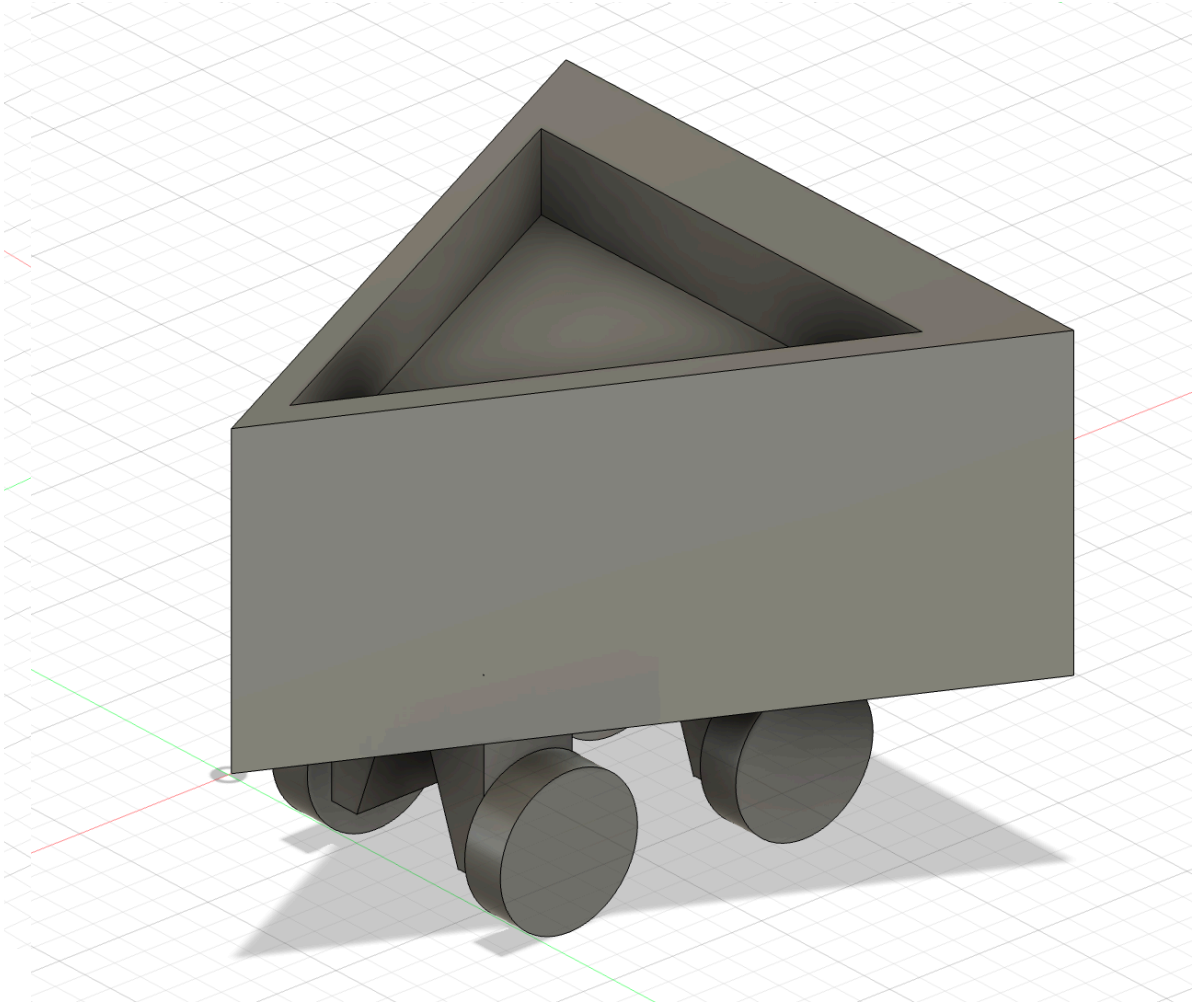
Na base de dados será guardada informação como as viagens realizadas e velocidades atingidas, de forma a podermos gerir o histórico do veiculo. A mesma será feita em Mongo Db, de forma a simplificar o formato da informação e o processo de guardar e receber a mesma.

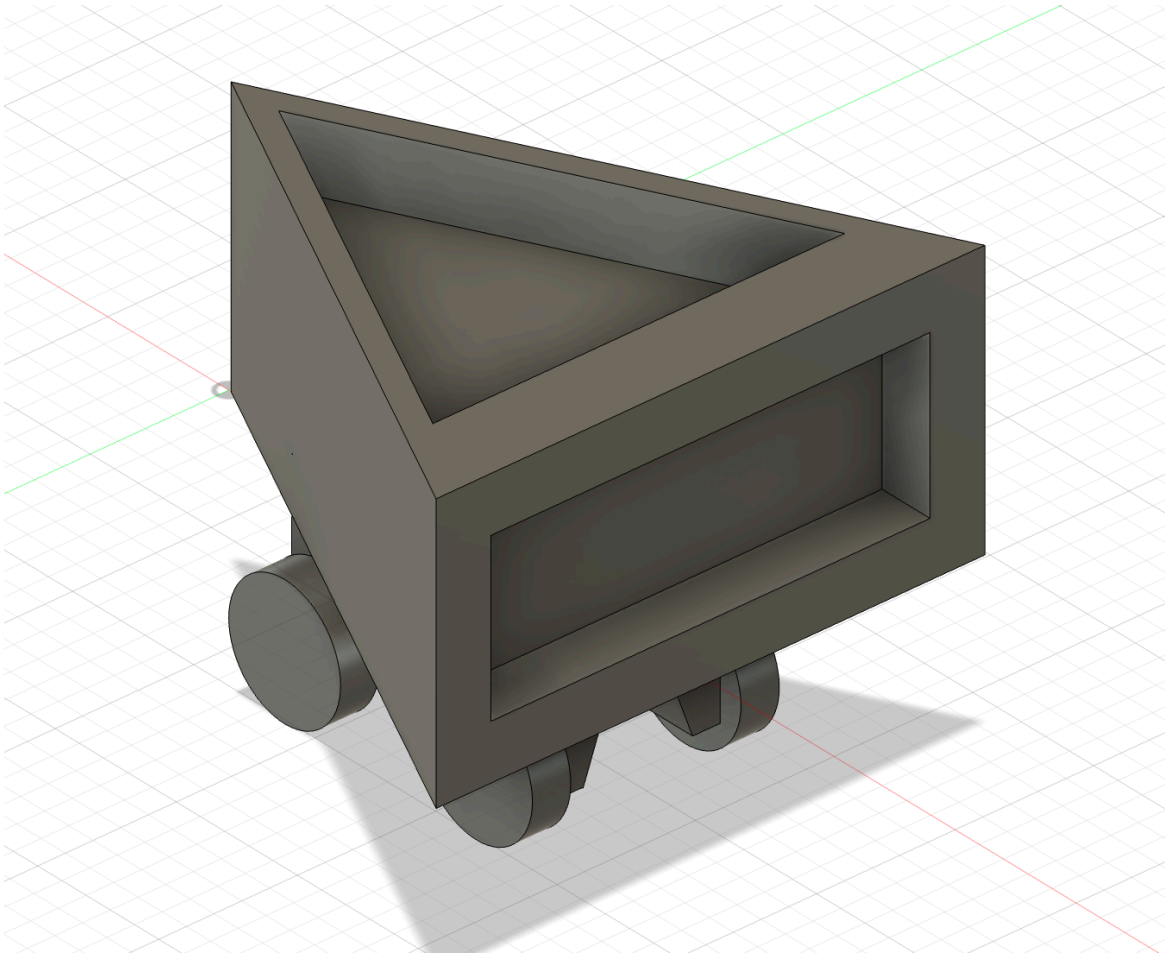
O mqtt Broker é uma parte essencial de conexões mqtt, o mesmo recebe a informação enviada pelos clientes e distribui-a nos canais apropriados. Escolhe-mos usar o Mosquito por ser simples na sua configuração.

O Vehicle-Client, será o programa a correr no veiculo. Este programa irá receber a informação enviada pelas outras estruturas, via mqtt.

Modelo 3D

De forma á melhor visionar o chassi do veiculo e preparar a sua futura impressão 3D, decidimos criar um modelo do mesmo. Seguem de seguida imagens do modelo:



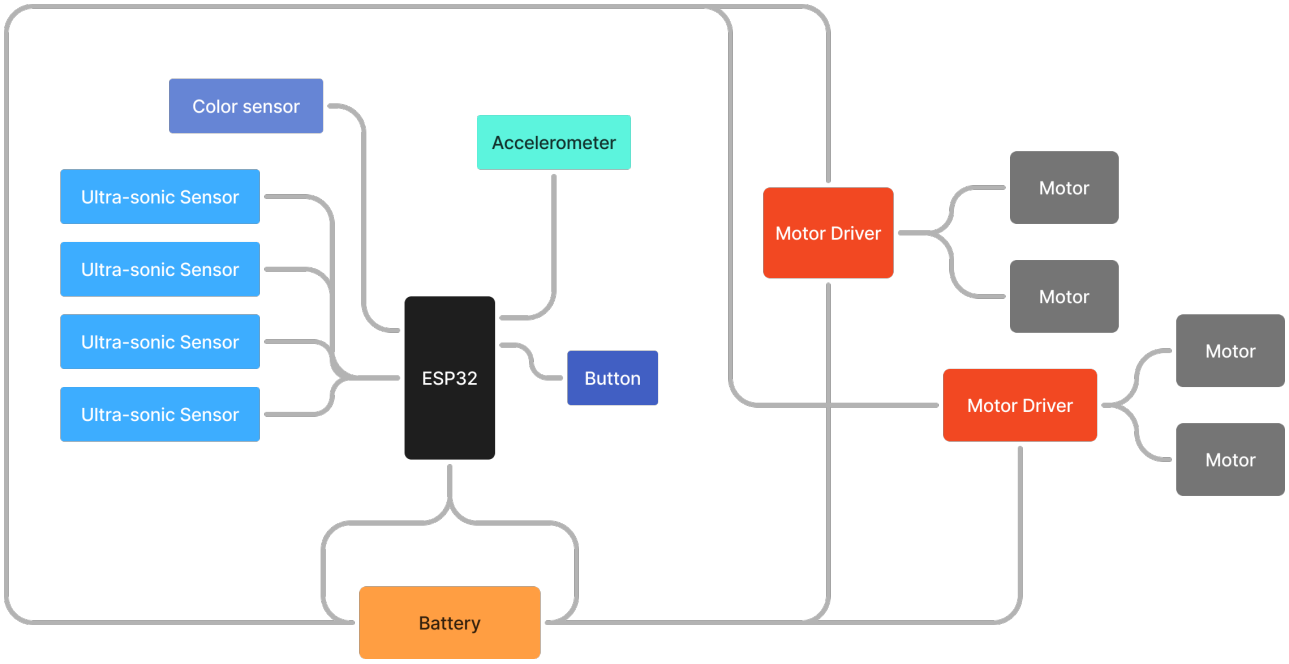


Como podemos observar, o chassi é formado por uma figura triangular, com um encaixe em cima para a carga a ser transportada, um encaixe na parte de trás para componentes como o botão e o ESP32, e um suporte embaixo onde residem as rodas. As rodas encontram-se em suportes, e não no chassi, de forma ao veiculo poder passar sobre terreno irregular mais facilmente.

Circuitos

O circuito do veiculo é simples na sua execução. O mesmo foca-se em conectar os sensores ao ESP32 e a bateria ao mesmo e aos *drivers* dos motores, de forma a garantir que, caso seja necessário um motor com maior voltagem, o mesmo continua a ser alimentado.

Segue o diagrama do circuito:



Material

De forma a ser possível construir o veiculo do nosso projeto, prevemos a que seja necessário o seguinte material:

- 1x Micro-controlador ESP32
- 4x Sensores ultra-sónicos
- 1x Sendo de Cor
- 1x Botão
- 1x Acelerómetro
- 1x Batteria
- 4x Motores (1800 RPM)
- 2x *Drivers* para Motores (cada *driver* deve suportar dois motores)
- 4x Rodas Mecanum (2 direitas, 2 esquerdas)
- Bobina de Filamento PLA, para o chassi

O material foi escolhido de forma a manter a solução o mais simples possível, como tal decidimos usar *drivers* de moteres duplos e só usar um sensor de cor na parte de baixo do veículo, já que o veiculo só é considerado fora do trajeto caso passa todo para fora da área. Decidimos usar PLA para a constituição do chassi já que é uma solução *cost effective* e comum em impressão 3D.

Trabalho Realizado

Devido aos prazos apertados neste projeto, decidimos que era importante começar já o desenvolvimento de algumas estruturas do projeto, para alem do pedido para a *milestone* 1. Segue a lista do trabalho já realizado:

Ricardo:

- DashBoard (versão protótipo) (Red-Node)
- Implementação de um mqqt-broker (Mosquitto)
- Criação de diferentes prototype-clients para interagir com o broker (Publisher && Subscriber)

João:

- Modelos de Dados
- Diagramas
- Relatório
- Slides
- Planeamento geral de interfaces do projeto

Partilhados:

- Pesquisa

Cronograma

Para o desenvolvimento deste projeto criamos um planeamento capaz de entregar todas as funcionalidades previstas. Incluindo também tempos de correção de erros e melhorias de funcionalidades. Sendo assim o planeamento esperado é o seguinte:

- Criação da base de dados
- Integração da base de dados com a API
- Desenvolvimento do client do veículo
- Integração do client do veículo com a API
- Desenvolvimento da Condução Autónoma
- Desenvolvimento do protótipo físico
- Construção do modelo físico

Links Importantes

Github Mestre - Repositório mestre do projeto (incluindo o repositório da API e do cliente do veiculo)

Clickup de Projeto - Repositório de tarefas;