UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR)
CURSO DE ENGENHARIA DE COMPUTAÇÃO

PEDRO FRANCESCON CITTOLIN
RICARDO LUIS SOUZA ARDISSONE
VIVIANE L B MORONI DE SOUZA

EASY MARKET: CARRINHO DE COMPRAS AUTOMATIZADO

OFICINA DE INTEGRAÇÃO 2 – RELATÓRIO FINAL

CURITIBA

2021

PEDRO FRANCESCON CITTOLIN
RICARDO LUIS SOUZA ARDISSONE
VIVIANE L B MORONI DE SOUZA

EASY MARKET: CARRINHO DE COMPRAS AUTOMATIZADO

Relatório Final da disciplina Oficina de Integração 2, do curso de Engenharia de Computação, apresentado aos professores que ministram a mesma na Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção da aprovação na disciplina.

Orientador: Prof. Dr. César Manuel Vargas Benítez
Prof. Dr. Heitor S. Lopes

CURITIBA

2021

Este trabalho é dedicado à memória das mais de 430 mil vítimas, até o momento, que morreram em circunstância da contaminação por Covid-19 em nosso país.

AGRADECIMENTOS

Agradecemos aos professores César Benítez e Heitor Lopez que nos apoiaram durante o desenvolvimento deste projeto. Também aos nossos familiares que nos apoiaram neste momento onde as atividades acadêmicas se entrelaçaram à rotina familiar.

Agradecemos a nosso familiares pelo auxilio em algumas áreas de conhecimentos das quais temos menos domínio, testes, filmagem, montagem e outros auxílios necessários devido as dificuldades impostas pela pandemia de reunião dos membros da equipe.

RESUMO

. EASY MARKET: CARRINHO DE COMPRAS AUTOMATIZADO. 50 f. Oficina de Integração 2 – Relatório Final – Curso de Engenharia de Computação, UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR). Curitiba, 2021.

Este relatório descreve o planejamento e desenvolvimento do projeto Easy Market, um carrinho de mercado automatizado. O carrinho Easy Market detecta automaticamente os produtos nele colocados ou retirados. Além disso, segue o comprador no mercado durante a compra. O projeto foi desenvolvido a partir de uma Raspberry Pi 3B e um kit 2WD para estrutura mecânica e acoplagem dos dispositivos eletrônicos. Foram também necessários diversos sensores e atuadores para implementação das funcionalidades citadas, como o RFID, uma célula de medição de peso, sensor ultrassônico, câmera para reconhecimento de imagens, motores, entre outros.

Palavras-chave: Carrrinho, Mercado, Eletrônico

ABSTRACT

. EASY MARKET: AUTOMATIC MARKET CART. 50 f. Oficina de Integração 2 – Relatório Final – Curso de Engenharia de Computação, UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR). Curitiba, 2021.

This report describes the planning and development of the Easy Market project, which is an automatic market cart. The Easy Market cart detects if a product was added or removed from the cart. It also follows the client during shopping. The project was developed using a Raspberry Pi 3B and a 2WD kit for mechanical structure and attachment of electronic devices. Several sensors and actuators were required for the implementation of the mentioned features, such as a RFID reader, load cell, ultrassonic sensor, camera for image processing, motors and others.

Keywords: Cart, Grocery, Electronic

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

RFID	Radio Frequency Identification
CI	Circuito Integrado
ADC	Analog-Digital Converter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
USB	Universal Serial Bus
TTL	Transistor–Transistor Logic
BGR	Blue, Green, Red
RGB	Red, Green, Blue
HSV	Hue, Saturation, Value
PID	Proporcional, Integrativo, Derivativo
HW	HardWare
RPi	Raspberry Pi
HF	High-Frequency
API	Application Programming Interface
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
CRUD	Create, Read, Update, Delete
BD	Banco de Dados
RDBMS	Relational DataBase Management System
SW	SoftWare
FPS	Frames Per Second
PWM	Pulse-Width Modulation
UML	Unified Modeling Language
PoC	Proof of Concept

LISTA DE SÍMBOLOS

V	Volts
cm	Centímetro
g	Grama

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Diante do atual cenário de pandemia, o simples e essencial ato de fazer compras no mercado se tornou um risco a saúde pública. Mesmo com restrições do número de pessoas que podem entrar nos mercados e regras mais rígidas como uso de máscara e controle de temperatura ainda existem diversos riscos que um comprador está sujeito ao ir ao mercado. Os carrinhos são sempre compartilhados entre diferentes compradores e os produtos não apenas passam na mão de diferentes clientes como também do caixa, que atende dezenas de compras por dia e se expõe constantemente.

A partir disso vimos a possibilidade de aplicar tecnologia e resolver algo comum aos mercados, o carrinho de compras, de forma que tornasse o processo de fazer compras muito mais prático e seguro.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O primeiro principal objetivo do Easy Market é acabar com a necessidade de caixas de mercado detectando automaticamente os produtos que são adicionados ou retirados do carrinho. Dessa forma ao fim da compra o pagamento pode ser realizado sem nenhuma outra interação com caixas de mercado por um aplicativo que já contém a lista de todos os produtos adquiridos pelo comprador.

O segundo objetivo principal do Easy Market é o de reduzir o contato do comprador com o carrinho - que é um dos objetos potencialmente mais contaminados em um mercado. Por meio de reconhecimento de imagens o carrinho pode então detectar a localização do comprador e se mover automaticamente para segui-lo durante a compra, de forma que o comprador não precise de contato físico com o carrinho.

1.2.2 OBJETIVOS ESPECÍFICOS

- Montar carrinho com motores
- Acoplar baterias, regulador de tensão e ponte H
- Acoplar cesta ao chassi
- Acoplar à cesta a balança e o leitor RFID
- Acoplar sensor ultrassônico
- Acoplar uma câmera
- Controlar a movimentação do carrinho
- Implementar funcionamento do leitor de RFID
- Implementar o funcionamento da balança
- Identificar distância aos objetos pelo sensor ultrassônico
- Obter imagens do comprador pela câmera acoplada
- Projetar o Hardware
- Projetar o Software
- Desenvolver software mobile
- Desenvolver software para placa
- Desenvolver software para o servidor
- Conectar todos os elementos do projeto
- Realizar testes de integração

2 FUNDAMENTAÇÃO TEÓRICA

2.1 HARDWARE

Existem células de carga, sensores utilizados para medir peso para vários níveis de peso. A saída dessa célula é analógica, sendo necessário um outro CI para converter para um sinal digital. Esse CI, no nosso caso o HX711, tem que ser um ADC especializado, devido a baixa tensão e pequenas variações de tensão da célula de carga.(AVIA SEMICONDUCTOR,) (BUILD. . . ,)

Detecção de RFID opera com um sensor, esse sempre alimentado, detectando *tags*, que podem ser ou ativas (ou seja, alimentadas com uma fonte normal), ou passivas (ou seja, a única alimentação é a própria radioemissão do detector RFID). Além disso, RFIDs podem serem distinguidos pela frequência ema que operam, com as *tags* e o detector tendo que ter a mesma frequência de operação. A principal diferença entre os casos é o preço e a distancia máxima de detecção. *Tags* ativas tem alcance significativamente maior que as passivais, mas tem um custo muito alto para serem utilizadas de forma descartável, como é o caso nesse projeto. Quanto a frequência, elas variam em disponibilidade e preço e escolhemos alta-frequência devido a esses fatores. (JECHLITSCHEK, 2006)

Um motor pode requer tensão diferente da da logica do circuito, e também para evitar que esse motor cause oscilações na alimentação e interferência nos circuitos digitais é recomendável utilizar alimentações separadas para os motores, com uma tensão mais elevada. A força efetiva do motor, porem, pode ser ajustada utilizando a técnica de PWM. Nesse caso, um sinal digital varia em *duty-cicle* de 0% a 100%, ou seja, ele varia quantos por cento do tempo o sinal está alto. Com uma frequência de PWM suficiente, o efeito disso é que a potência recebida pelo motor, e portanto a força deste, vai variar de 0 a 100%.

A comunicação e conexão entre os componentes de HW requer uso de vários métodos de comunicação, como SPI, I2C, USB e protocolos digitais específicos a um componente. O único dado analógico, saindo da célula de carga, é primeiro processado pelo HX711 então não foi necessário tratar nenhum sinal realmente analógico.

Diferentes dispositivos digitais podem serem compatíveis com diferentes níveis lógicos de tensão, como TTL em 5 V e outros dispositivos com logica baseada em 3.3V. É necessário então checar os valores máximos e mínimos aceitos ou emitidos como saída e entradas nos níveis altos e baixos informados nos *datasheets* em cada par de ligação digital para garantir a compatibilidade. Caso não sejam, é necessário incluir circuitos tradutores, sejam redutores ou aumentadores de tensão.

2.2 SOFTWARE

Utilizamos diagramas de blocos, e diagramas de sequencia UML para projetar o SW.

A comunicação entre os componentes de SW principais usa uma arquitetura cliente-servidor utilizando API REST. Essa API então é sem estado, ou seja, cada transação é independente, é acessível via HTTP, utilizando os verbos padrões HTTP para informar qual transação devera ser executada. Isso permite grande flexibilidade e compatibilidade com uma variedade de dispositivos, e facilitando a implementação da comunicação em ambos os SWs clientes.

O banco de dados do servidor é um BD relacional, apenas necessitando de uma tabela extra para realizar uma relação muitos-para-muitos com dados extras.

2.3 PROCESSAMENTO DE IMAGENS

O processamento da imagem tem dois objetivos: Encontrar a posição X do alvo, e a distancia. Para isso antes precisamos encontrar o alvo. Para simplificar, decidimos que poderíamos definir um alvo arbitrariamente, ao invés de termos que nos adaptar a por exemplo seguir uma pessoa especifica sem nenhum indicio adicional de como distinguir uma pessoa de outra. Após alguns testes, decidimos concentramos em um simples detector de cor.

O primeiro passo do processamento é realizar a conversão do sistema de cores da imagem de BGR ou RGB para o sistema HSV. Isso porque com o sistema HSV, que se aproxima mais de como a visão humana opera, podemos especificar uma cor especifica a ser buscada ajustando o valor de H, minimizando o efeito das variações de luminosidade e da câmera na imagem. Após isso, é aplicada uma mascara com limites baixos e altos a imagem. Disso resulta uma nova imagem binaria, apenas com a cor buscada. Em seguida, executamos transformações morfológicas para ajustar a imagem e reduzir ruídos. No nosso caso, utilizamos dilatação. (GOODDAY451999, 2020)

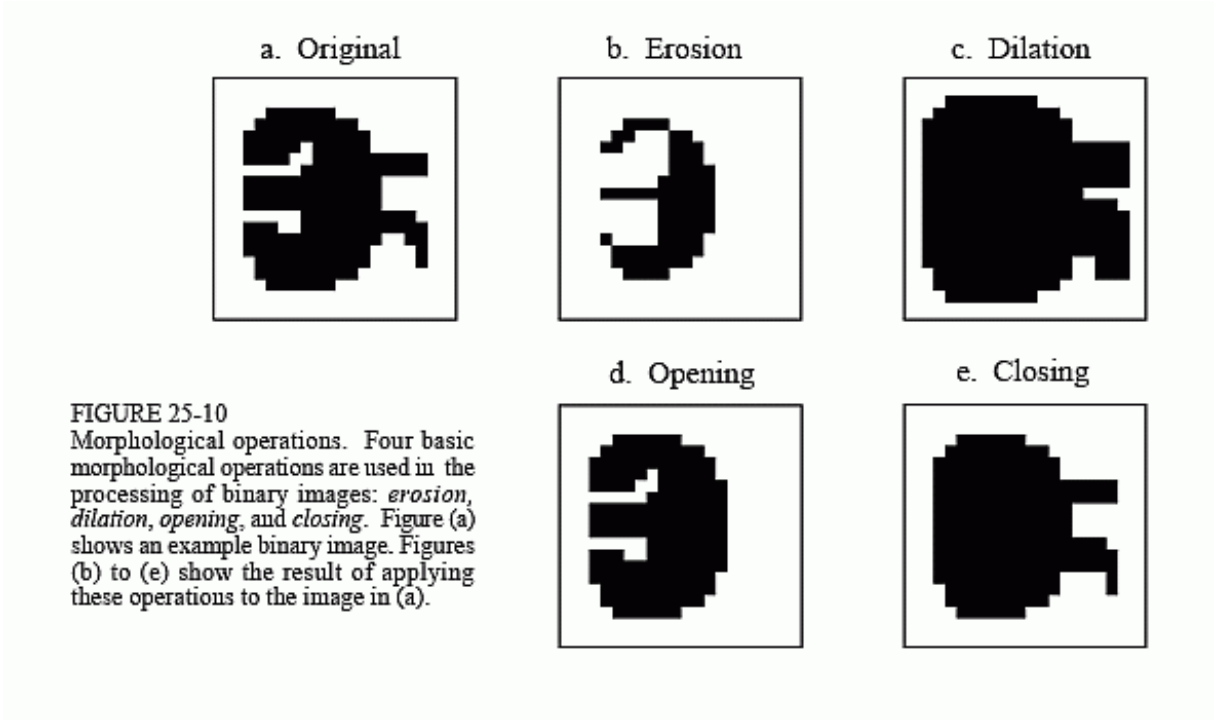

Figura 1: Demonstração das transformações morfológicas de imagens. (SMITH, 1997)

Uma operação de AND é então aplicada para obtermos apenas as partes desejadas da imagem original. Nesse imagem então executamos a operação de busca de contornos. Esses contornos então podem serem analisados para verificar se atendem a requisitos como convexidade, densidade e terem forma quadrada. Com dois contornos encontrados, podemos prosseguir.

2.3.1 OBTENDO DISTANCIA

Existem vários métodos para obter uma distancia câmera-objeto em uma imagens. Com duas câmeras, pode ser feita triangulação. Se o objeto tem um tamanho conhecido, podemos comparar o tamanho do objeto na imagem com o tamanho real.(ROSEBROCK, 2015a) Alguns desses métodos requerem calibração da câmera antes do uso. Como podíamos decidir o alvo, decidimos utilizar um simples método de medir a tamanho em pixels do alvo e ajustar a uma reta com a distancia objeto-alvo.

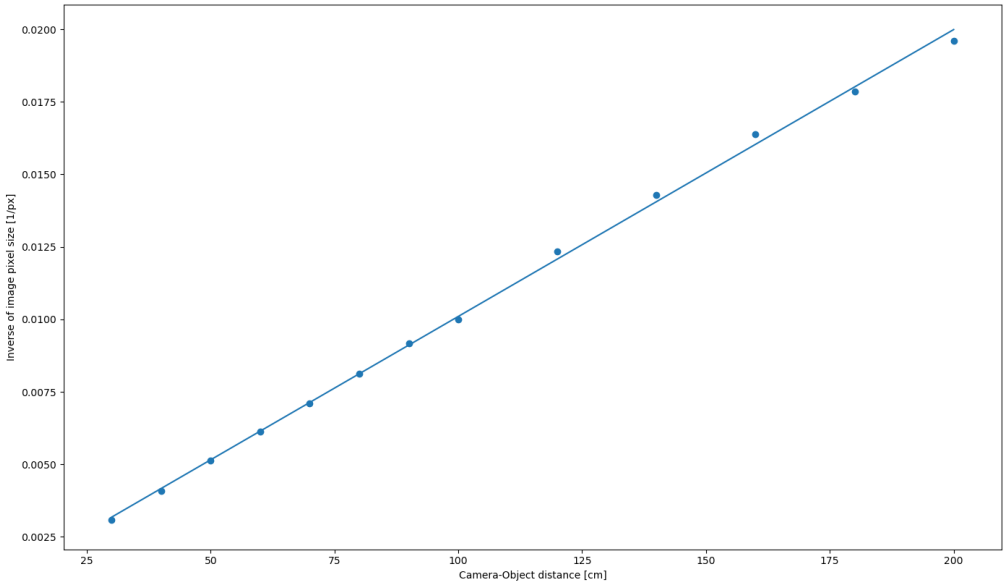

Figura 2: Gráfico da distância câmera-objeto vs tamanho em pixels do objeto

Medir o tamanho de um quadrado pode falhar devido a falhas parciais momentâneas de detecção, então utilizamos o método mais confiável de detectar a distancia em pixels entre dois objetos.

2.3.2 FILTRAGEM

Mesmo depois de todo esse processamento de imagens, ainda ocorrem falsos-positivos, ou seja, detectar um alvo que não é desejado, ou falsos-negativos, ou seja, ignorar um alvo que deveria ser detectado. Normalmente esses erros são momentâneos. Para evitar que eles tenham grande efeito, utilizamos a técnica da media móvel ponderada, um tipo de filtro passa baixa.

$$MMP = \frac{n * p_M + (n-1) * p_{M-1} + ... + p_{(M-n)+1}}{n + (n-1) + ... + 1}$$

2.4 CONTROLE

Esses dados da câmera são utilizados como entrada para o controle do motor. Inicialmente utilizamos um sistema de controle on-off simples, com o carrinho podendo estar em apenas alguns poucos estados discretos dependendo da posição do alvo.

Posteriormente, após experimentos implementamos parcialmente um controle PID mais completo. Para esse sistema, é necessário uma saída ajustável, no nosso caso, o PWM do motor, uma entrada, no nosso caso a posição do alvo na câmera, uma posição desejada da da entrada, no nosso caso, o alvo centralizado a frente do carrinho e da imagem, e uma retroalimentação, no nosso caso provida pela câmera. O erro então é a distancia do alvo no eixo X ate o centro da imagem.

O controle PID então ajusta o PWM com base nos 3 fatores(O'HANLON,) (BAYER; ARAúJO, 2011):

- Proporcional, ou seja, proporcional ao erro momentâneo, que causa maiores valores de PWM quando o alvo esta mais longe do centro, requerendo um ajuste maior de posição do carrinho
- Integrativo, ou seja, integral do erro acumulado, que causa maiores valores de PWM caso o erro se mantenha alto, devido por exemplo ao carrinho ter travado e estar faltando força para se mexer.
- Derivativo, ou seja, derivada do erro momentâneo, que ajusta o PWM com base na variação do erro para melhorar a estabilidade do sistema e evitar *overshoot*.

Esse sistema requer uma velocidade de resposta razoável, considerando a velocidade de rotação do carrinho. Também é necessário ajustar os parâmetros do PID para a aplicação.

3 METODOLOGIA

3.1 VISÃO GERAL

Para construir o sistema apresentado, precisamos desenvolver três partes principais e integra-las: estrutura mecânica, esquema eletrônico e software para coordenação das funções.

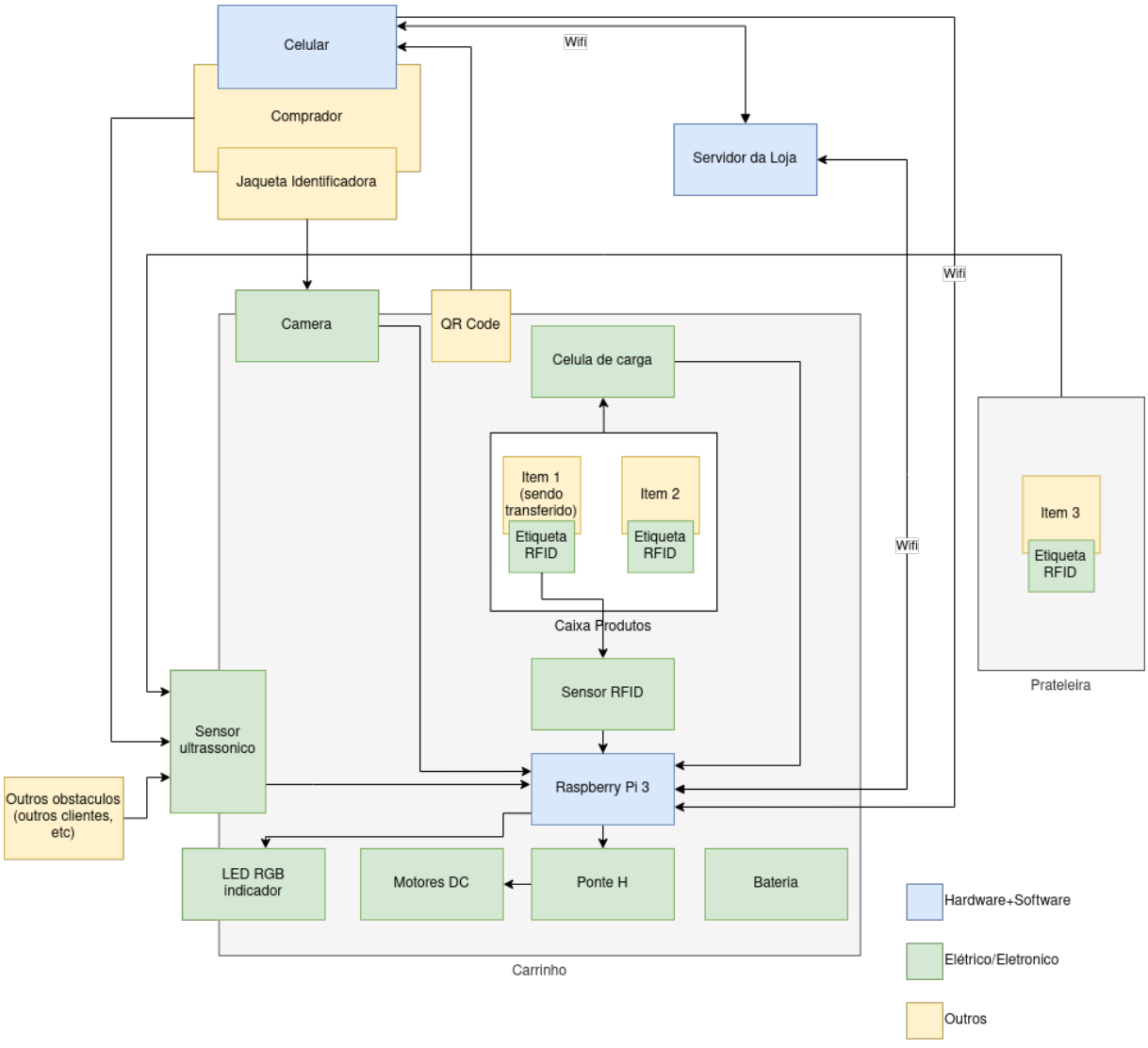

Figura 3: Diagrama de Blocos - Visão Geral do Sistema

3.2 PROJETO MECÂNICO

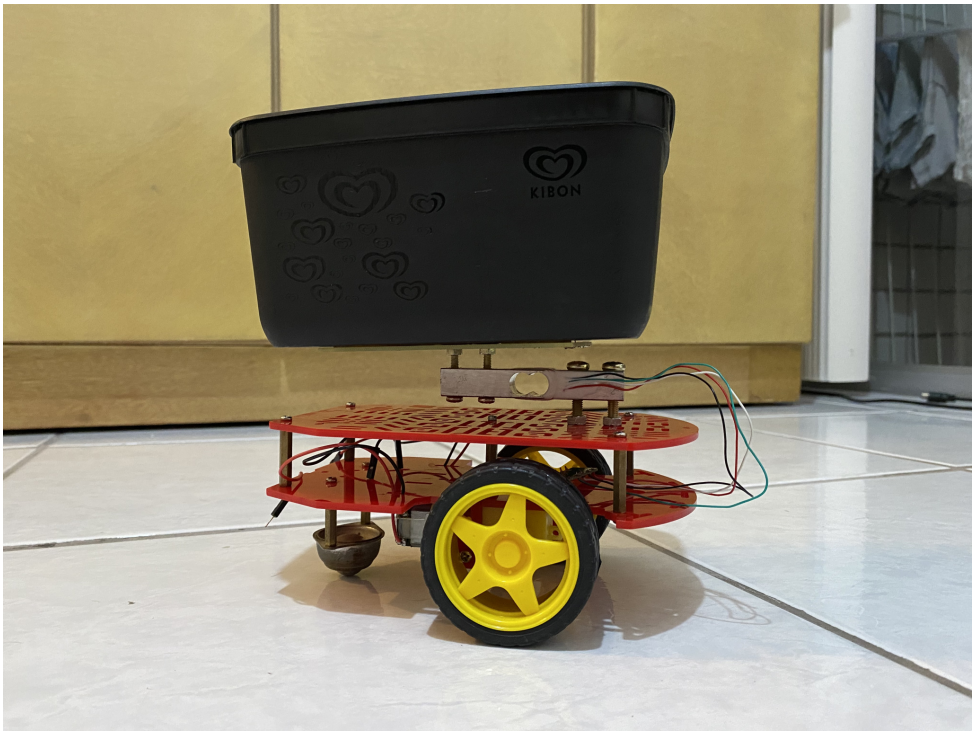

Figura 4: Estrutura Mecânica

Para a estrutura mecânica do projeto utilizamos um Chassi obtido por empréstimo junto aos orientadores, já acoplado com os motores e rodas. Também foi utilizado a própria célula de carga, componente do projeto eletrônico, para acoplamento da cesta ao carrinho. Para acoplamento dos demais componentes eletrônicos ã cesta utilizamos cola quente, além de fita adesiva dupla face.

O alvo escolhido depois dos experimentos iniciais foram os 2 quadrados coloridos, de tamanho fixo e distância entre ambos também fixa, valores conhecidos pelo software. Possibilitando, Dessa forma, o cálculo da distância do cliente ao carrinho.

3.3 PROJETO DE *HARDWARE*

O HW do carrinho esta centralizado em um *Raspberry Pi*. Necessitamos do RPi para conseguirmos fazer o processamento de imagens em tempo razoável.

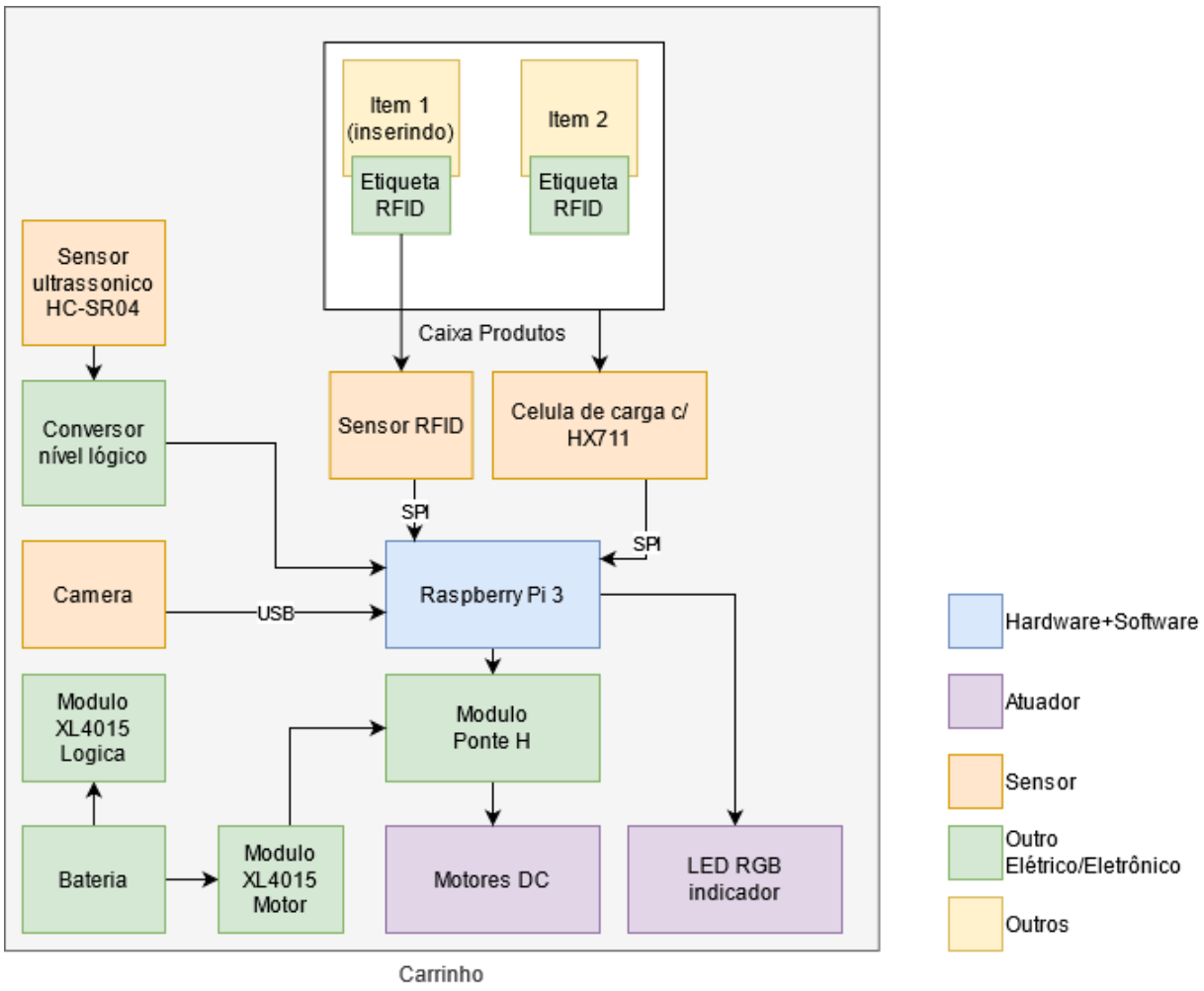

Figura 5: Diagrama de blocos do carrinho, incluindo os elementos de HW. Criado usando draw.io (DIAGRAMS.NET,)

Componentes e sensores utilizados:

- Detector RFID HF RC522. (NXP B.V., 2007) (GUS, 2017)
- 2x Módulo Regulador de Tensão Step Down XL4015 (HAREENDRAN, 2020)
- Módulo Ponte-H Motor (HANDSON TECHNOLOGY,) (STMICROELECTRONICS, 2000)
- 2x Motor DC (MOTOR. . . ,)
- Sensor ultrassônico SR04 (ELEC FREAKS,)
- Sensor de peso HX711 (AVIA SEMICONDUCTOR,).

Devido a incompatibilidade de níveis lógicos entre a saída do HC-SR04(ELEC FREAKS,) e a entrada do Raspberry Pi(RASPBERRY PI FOUNDATION,), é necessário

adicionar um simples circuito tradutor de níveis utilizando um divisor de tensão. Apos analise dos níveis de entrada e saída, chegamos a conclusão que o mesmo não era necessário para a conexão com o modulo L298(STMICROELECTRONICS, 2000), apesar de nominalmente o RPi e o L298 operarem em diferentes níveis lógicos.

O HW para executar o aplicativo, que pode ser qualquer dispositivo iOS, e do servidor, que pode ser instalado em qualquer PC compatível com o Python e o servidor Flask, estão fora do escopo do projeto.

3.4 PROJETO DE *SOFTWARE*

O software pode ser basicamente dividido em 3 componentes, cada um rodando em uma plataforma: Aplicativo Mobile, Servidor e Carrinho. Esse último pode ser dividido em partes executando no Node-RED e Python, com as partes em Python podendo ser separadas entre a parte de sensor (câmera e ultrassônico), e de controle do motor.

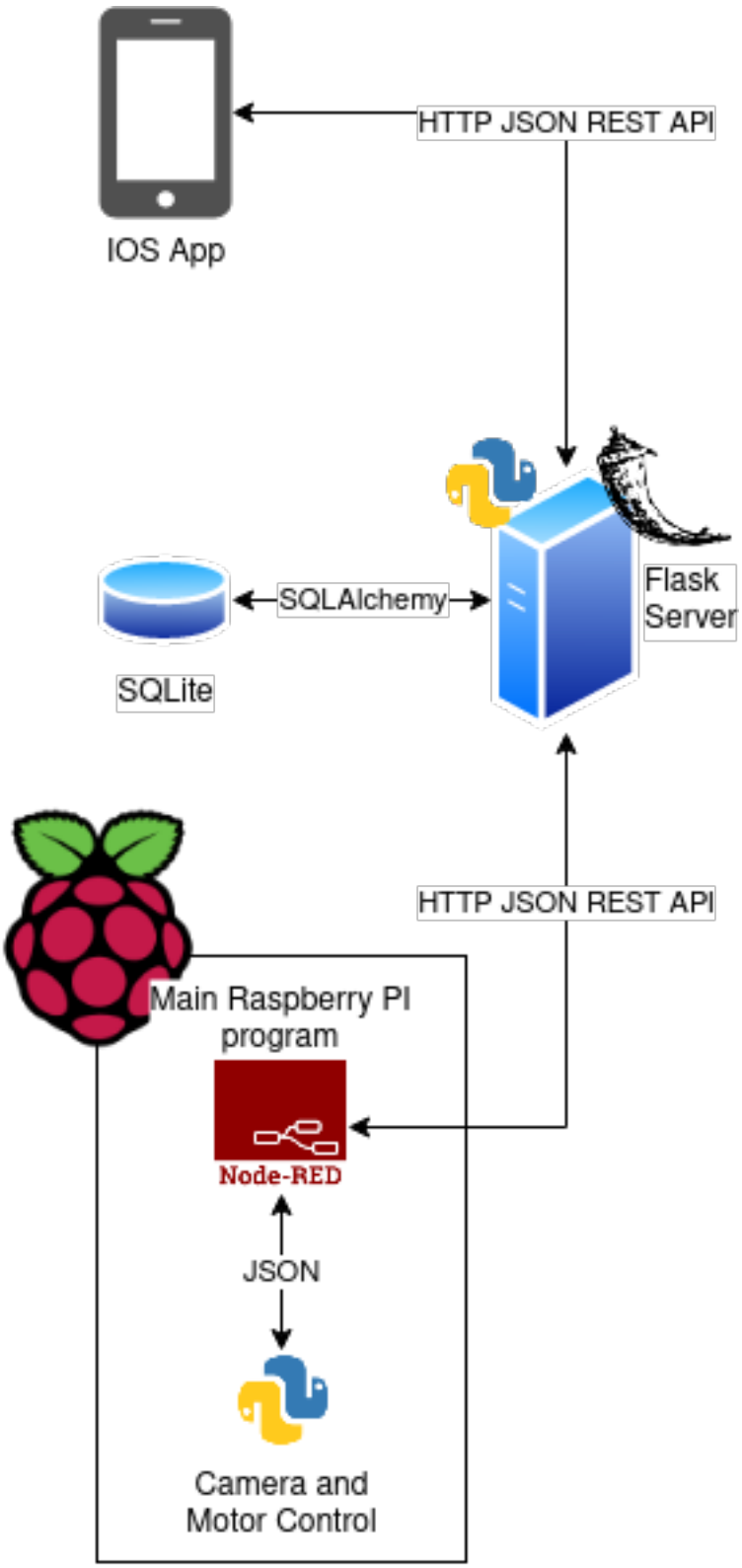

Figura 6: Diagrama dos blocos de software. Criado usando draw.io (DIAGRAMS.NET,)

3.4.1 INTEGRAÇÃO

A integração e comunicação entre os 3 elementos principais é centralizada no servidor e foi projetada utilizando dois diagramas de sequencia. Um para a criação de conta de um novo comprador, e um diagrama para a a compra em si.

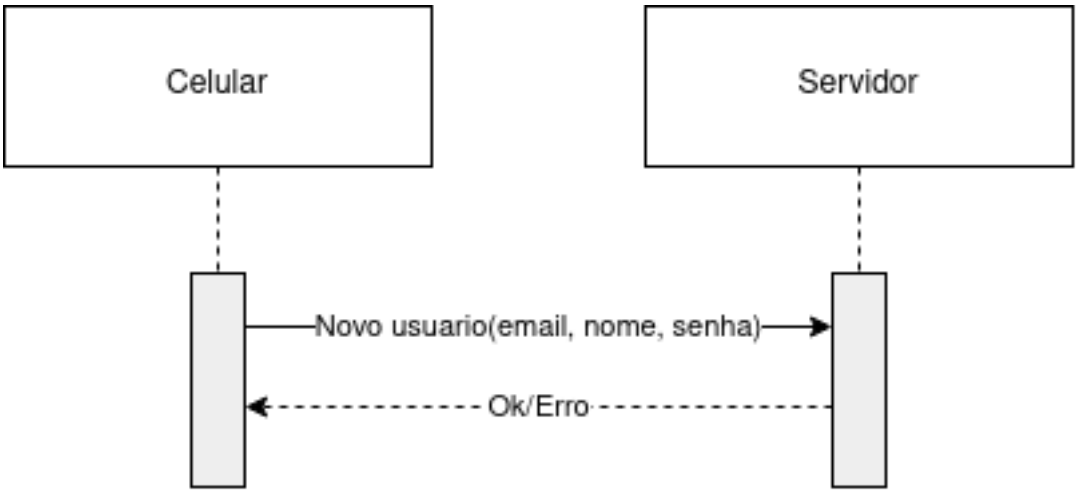

Figura 7: Diagrama de sequencia da comunicação Aplicativo-Servidor para criação de uma conta. Criado usando draw.io (DIAGRAMS.NET,)

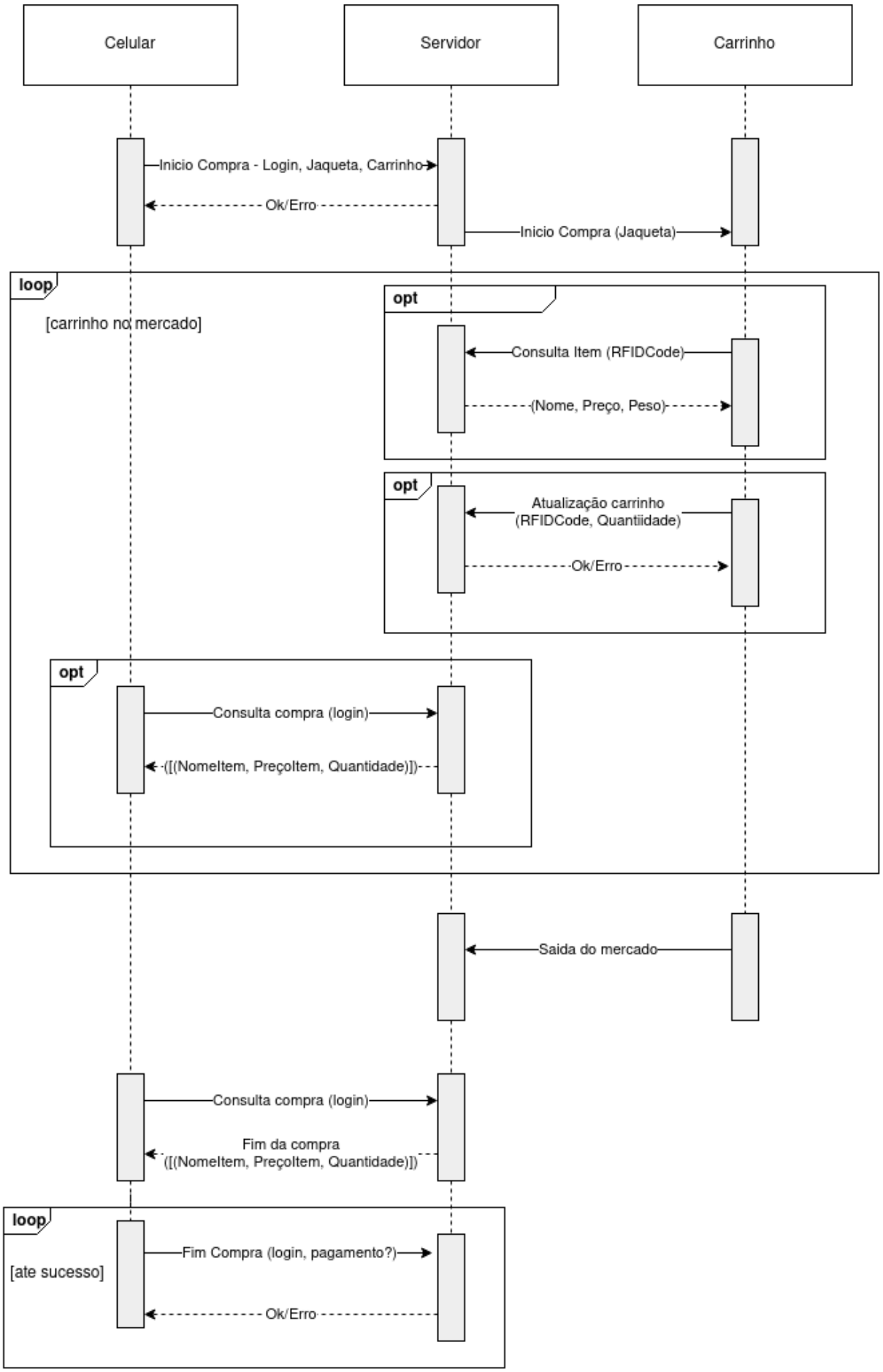

Figura 8: Diagrama de sequência da comunicação entre os 3 elementos principais de software durante a compra.

Para realizar essa comunicação utiliza-se uma API REST HTTP, transferindo dados em JSON conforme necessário. Essa API foi projetada utilizando a especificação OpenAPI(OPENAPI INITIATIVE,).

3.4.2 APLICAÇÃO *MOBILE*

Utilizamos a plataforma Sketch(SKETCH B.V.,) com a biblioteca Bloo Free Wireframe UI Kit (VIJAY,) para desenhar as telas que compõem o aplicativo, bem como indicar a navegação entre as mesmas. O desenvolvimento, para dispositivos iOS, foi programado utilizando a IDE XCODE (APPLE INC.,). O aplicativo foi planejado com uma tela inicial de login ou cadastro. Seguida de uma tela de cadastro do meio de pagamento, a princípio apenas cartão de crédito, caso o usuário ainda não tenha cadastrado. Depois de ter uma forma de pagamento cadastrada ele é levado a uma tela onde pode iniciar uma compra escaneando o código QR de um carrinho. Depois de escanear o código uma compra é iniciada e o aplicativo passa a mostrar os itens atualmente inseridos no carrinho, bem como seus respectivos preços. Quando a saída do cliente do mercado é identificada, a compra é finalizada e ocorre a tentativa de pagamento. Se for bem sucedida uma tela confirmação do pagamento e com os dados da compra são mostradas. Se ocorre algum erro, o cliente é redirecionado a tela de cadastro do meio de pagamento e uma nova tentativa é realizada ao fim do recadastramento. Além disso, uma funcionalidade extra planejada foi a possibilidade de controlar o carrinho através do aplicativo, numa tela acessível a partir da tela da compra em andamento. Como pode ser visto na figura 9:

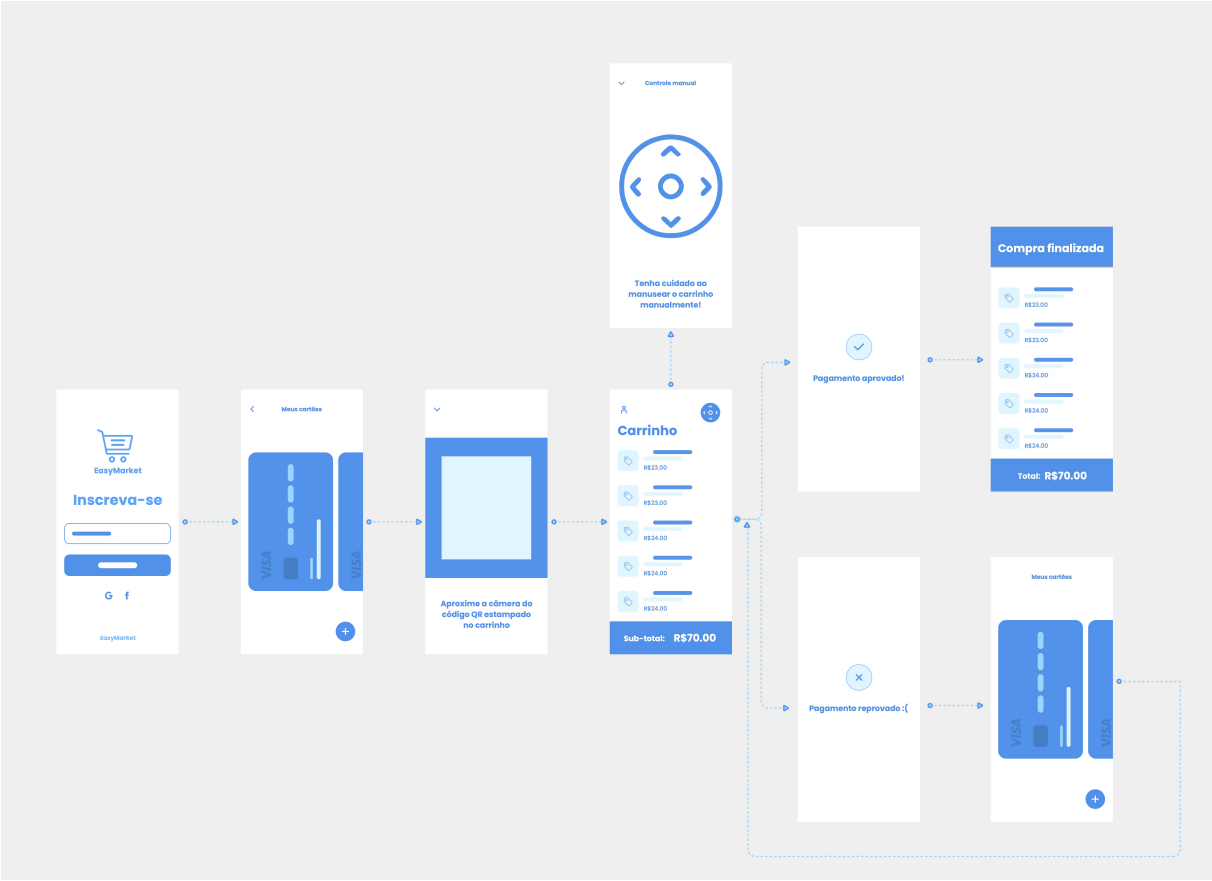

Figura 9: *Wireframes*: Navegação entre as Telas do Aplicativo

Planejou-se gastar um número considerável de horas, no escopo do projeto, para esta atividade. Visto que esse tipo de desenvolvimento, mesmo de aplicativos simples, costumam envolver uma equipe especializada, além de um prazo de pelo menos alguns meses. Portanto era conhecida a limitação do que poderia de fato ser entregue considerando as condições deste projeto.

3.4.3 SERVIDOR

O servidor foi gerado a partir da especificação da comunicação entre componentes, utilizando o OpenAPI Generator(OPENAPITOOLS,) para gerar a base do código para um servidor Flask(RONACHER, a). Apenas foi necessário codificar a interação das operações CRUD necessárias com o BD.

3.4.3.1 BANCO DE DADOS

O banco de dados foi projetado com 4 tabelas, sendo uma tabela, ItemCompra, uma tabela extra para representar uma relação muitos-para-muitos com dados. A tabela Item

especificamente é a única que não é modificável via o servidor, com a intenção que ela seja atualizada manualmente pelo gerente da loja utilizando um aplicativo de gerenciamento de banco de dados, como o *DB Browser for SQLite*.

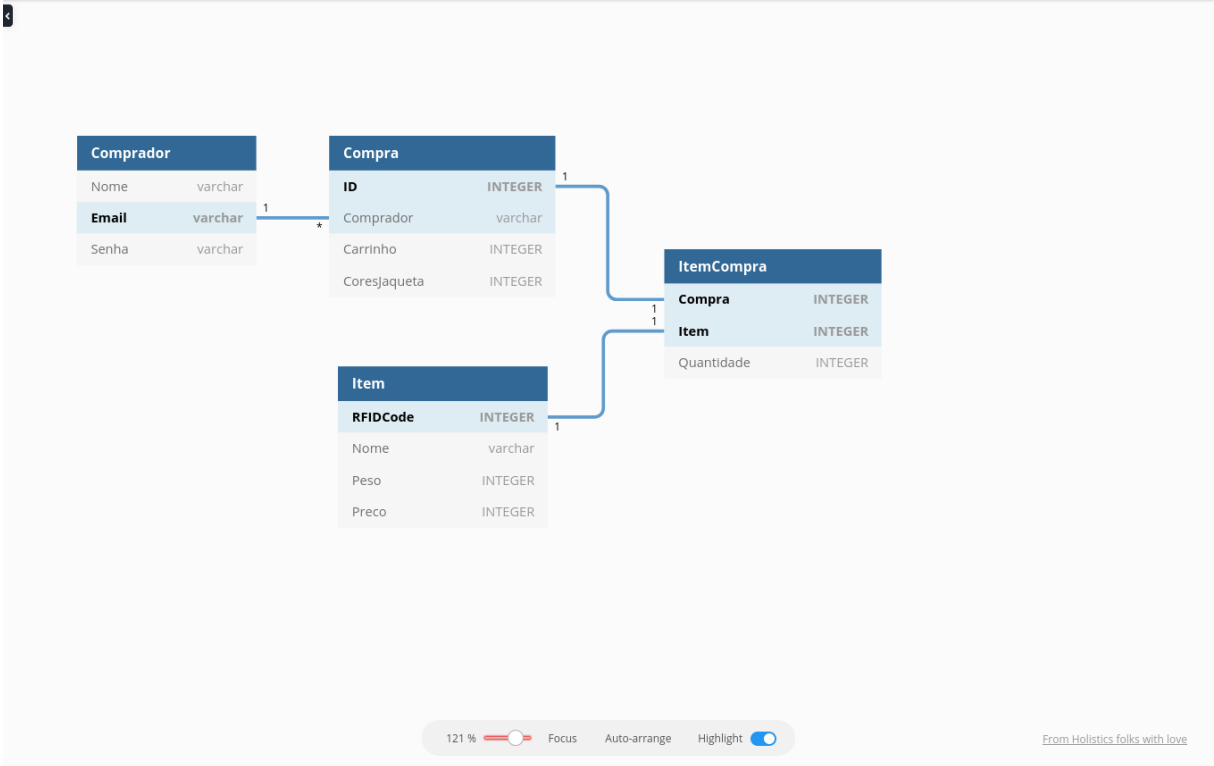

Figura 10: Diagrama Entidade-Relacionamento do Banco de Dados. Desenhada utilizando dbdiagram.io(HOLISTICS SOFTWARE,)

Esse banco de dados foi implementado declarativamente como parte do servidor utilizando a biblioteca SQLAlchemy (RONACHER, b) (BAYER,). Como o sistema só tem um servidor conectado ao banco de dados na mesma maquina e as necessidades de performance são relativamente pequenas(SQLITE,), optamos pela simplicidade do RDBMS SQLite(HIPP, 2020).

3.4.4 CARRINHO

O carrinho opera apenas sob direção do resto do sistema, estando parado esperando no começo, e apos termino da compra também para.

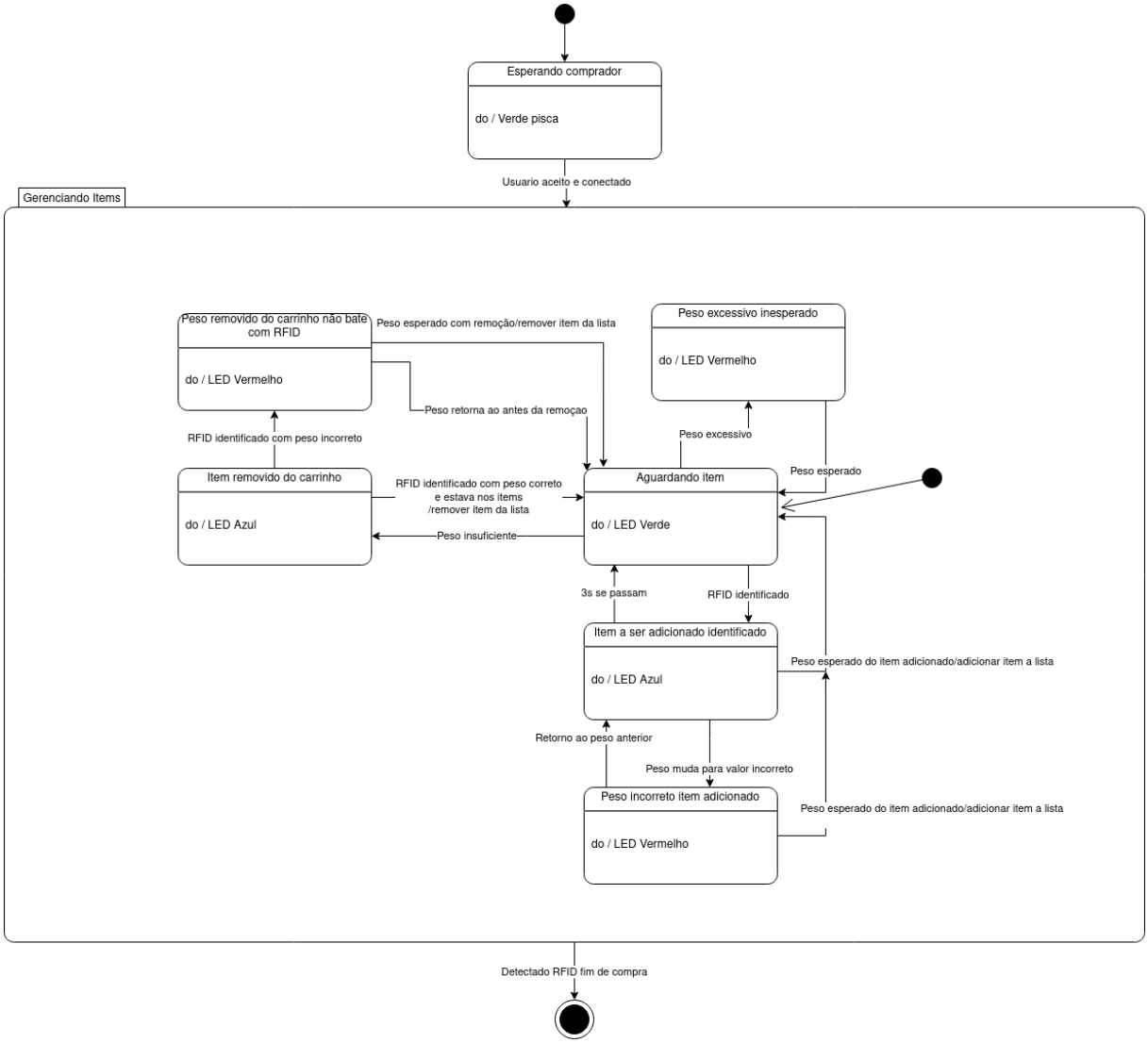

Figura 11: Diagrama de estados do carrinho. Desenhado usando draw.io (DIAGRAMS.NET,)

O carrinho foi projetado inicialmente com duas partes, uma parte em Python utilizando OpenCV para ler e processar imagens e o restante implementado utilizando Node-RED, com a comunicação entre as partes utilizando JSON.

Apos o controle de motor simples on-off implementado no Node-RED mostrar-se insatisfatório, reprojetamos o SW do carrinho com o objetivo de minimizar o tempo de leitura da câmera, utilizando *multithreading*, movendo o codigo de controle de motor do Node-RED para o Python para evitar o atraso de comunicação, e implementando um controle mais elaborado.

3.4.4.1 NODE-RED

Foi desenvolvido utilizando a plataforma visual Node-RED(OPENJS FOUNDATION,) e bibliotecas prontas para controle de módulos externos. Utilizamos o *dashboard* para facilitar

ajustes imediatos nos parâmetros do sistema para teste.

3.4.4.2 CÂMERA

A câmera opera buscando dois quadrados de uma cor configurável. Para conseguir encontrar apenas os alvos desejados, ha vários testes para checar se um dado contorno encontrado na imagem realmente é um dos quadrados sendo procurado. Utilizamos a biblioteca de processamentos de imagens OpenCV(BRADSKI, 2000) e a biblioteca de matrizes NumPy(HARRIS et al., 2020).

Utilizamos também uma media móvel ponderada para melhorar a estabilidade do resultado e evitar que um falso-positivo ou falso-negativo na aquisição de alvo momentâneo atrapalhe muito o funcionamento. Isso tem a desvantagem de diminuir o tempo de resposta do ciclo de controle a mudanças na posição do alvo. Um dos requisitos dessa parte do SW é um alto FPS, que foi atingida utilizando técnicas de *multithreading*.(ROSEBROCK, 2015b)

3.4.4.3 MOTOR

A implementação do motor leva em consideração, além da posição X e distancia do alvo obtidas da câmera, a distancia ate um obstaculo frontal medida pelo ultrassônico, medida utilizando a biblioteca Bluetin_Echo(HEYWOOD,).

Para o controle do motor em si, primeiro executa-se um algoritimo para determinar qual a direção que o motor deve se deslocar, que pode-se observar na tabela 1.

Alvo a vista	Ultrassônico	Distância	Posição X	Resposta
Nunca	X	X	X	Parar
Sim	X	Muito Perto	X	Trás
Não	X	X	X	Parar
Sim	X	Perto	Muito Esquerda	Girar Esquerda
Sim	X	Perto	Esquerda	Girar Esquerda
Sim	X	Perto	Centro Esquerda	Parar
Sim	X	Perto	Centro	Parar
Sim	X	Perto	Centro Direita	Parar
Sim	X	Perto	Direita	Girar Direita
Sim	X	Perto	Muito Direita	Girar Direita
Sim	Perto	Longe	Muito Esquerda	Girar Esquerda
Sim	Perto	Longe	Esquerda	Girar Esquerda
Sim	Perto	Longe	Centro Esquerda	Parar
Sim	Perto	Longe	Centro	Parar
Sim	Perto	Longe	Centro Direita	Parar
Sim	Perto	Longe	Direita	Girar Direita
Sim	Perto	Longe	Muito Direita	Girar Direita
Sim	Longe	Longe	Muito Esquerda	Girar Esquerda
Sim	Longe	Longe	Esquerda	Avançar
Sim	Longe	Longe	Centro Esquerda	Avançar
Sim	Longe	Longe	Centro	Avançar
Sim	Longe	Longe	Centro Direita	Avançar
Sim	Longe	Longe	Direita	Avançar
Sim	Longe	Longe	Muito Direita	Girar Direita

Tabela 1: Lógica de controle do motor

Em seguida, utiliza-se um controle PID para ajustar o PWM do motor. A implementação do PID só leva em conta a posição X (horizontal) do alvo, não a distancia ou o ultrassônico. O erro do controle PID então é a distancia entre o alvo e o centro da imagem.

Ambos os motores e PWM são controlados via a biblioteca recomendada RPi.GPIO (CROSTON,).

3.5 INTEGRAÇÃO

Além da integração entre os vários componentes de SW, HW e mecânicos, também utilizamos de forma integrada conhecimentos de varias disciplinas.

Primeiro temos as disciplinas de projeto de SW, como Análise e Projeto de Sistemas e Engenharia de Software, onde estudamos diagramas UML, definição de requisitos e outros elementos de projeto de SW. Como esse é um sistema distribuído com vários componentes de SW em uma arquitetura cliente servidor, utilizamos a arquitetura REST estudada em Desenvolvimento Integrado de Sistemas para realizar a integração entre componentes via rede.

A comunicação entre os elementos de SW é via HTTP, utilizando conhecimentos básicos da disciplina de Redes de Computadores. O conhecimento de *multithreading* de Sistemas Embarcados foi útil para implementar alguns aspectos do SW que lé a câmera. O servidor requereu conhecimentos de Banco de Dados. Todos esses elementos de SW então tiveram de ser implementados utilizando conhecimento e pratica das matérias de programação aplicada como Técnicas de Programação.

Quanto ao HW, o circuito quase todo é digital então usamos principalmente conhecimentos de Circuitos Digitais. Tivemos que levar em consideração também alguns aspectos não digitais como níveis lógicos, alimentação e *driver* de motor de motor utilizando conhecimentos de Circuitos Elétricos. Conforme necessário, esses circuitos foram desenhados utilizando conhecimentos de Desenho Técnico Aplicado.

Os conhecimentos anteriores sobre projeto e LaTeXde Oficinas 1 foram reutilizados.

Finalmente, utilizamos alguns conhecimentos de disciplinas não cursadas por nenhum integrante, Processamento Digital de Imagens para realizar a detecção de alvo na câmera e Controle 1 para utilizar essa informação para controlar o motor.

4 EXPERIMENTOS E RESULTADOS

Segundo o cronograma de entregas e atividades da disciplina tínhamos algumas semanas dedicadas a realização de testes e posteriormente correção de falhas e ajustes, processos fundamentais de qualquer projeto de integração.

4.1 HARDWARE E SOFTWARE

4.1.1 POC: RECONHECIMENTO DE IMAGENS

Desde o início do semestre a movimentação automática do carrinho por reconhecimento de imagens foi bastante questionada pelos professores quanto a viabilidade de execução, justamente pela dificuldade de implementação e complexidade de integração das tecnologias envolvidas.

Uma prova de conceito (PoC) foi realizada logo no início do semestre para testar o funcionamento do reconhecimento de imagens como proposto para o nosso projeto. Após o estudo dos diversos caminhos que poderíamos seguir, realizamos testes de reconhecimento de quadrados coloridos, executando o algoritmo desenvolvido em Python com a biblioteca OpenCV diretamente na Raspberry Pi. O resultado pode ser visualizado nas Figuras 12 e 13

Como mostra a Figura 12, o algoritmo foi capaz de reconhecer e diferenciar os dois quadrados azuis do resto da imagem, identificando inclusive a posição em coordenadas X e Y dos dois quadrados. A Figura 13 mostra um gráfico do valor de distância estimado dos quadrados azuis em relação a câmera, conforme os quadrados eram afastados. Diante dos resultados consideramos o teste como bem sucedido, validando a prova de conceito

4.1.2 SENSORES E ATUADORES

Os próximos testes realizados foram em relação ao funcionamento de todas as partes de hardware do carrinho - sensor ultrassônico, motores DC, sensor RFID e sensor de peso (módulo hx711). Esse processo é essencial pois dessa forma podemos identificar potencias problemas

Figura 12: PoC - Recohecimento de quadrados coloridos em uma imagem

Fonte: Autoria própria.

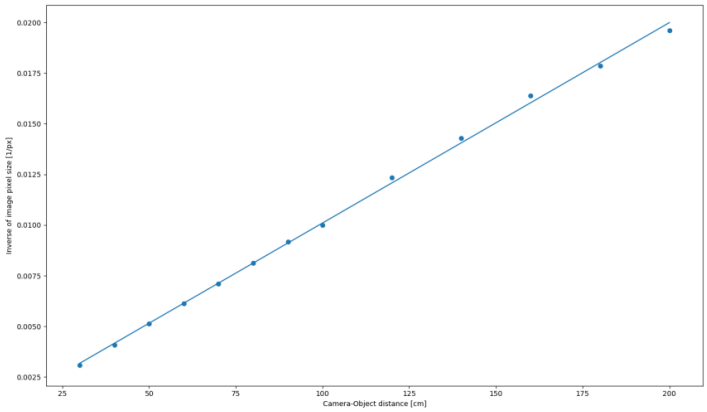

Figura 13: PoC - Gráfico de distância em função do tempo

Fonte: Autoria própria.

e dificuldades na hora de realizar a integração de todas as partes do projeto de forma isolada em cada componente. Cada periférico foi então conectado individualmente a Raspberry Pi e, utilizando-se da ferramenta de Dashboard do Node-RED, realizamos testes de funcionalidades básicas de cada componente. Os videos gravados durante a realização dos testes podem ser visualizados no blog da disciplina (CITTOLIN et al., 2021).

O sensor RFID não apresentou problemas na hora de identificar a presença de diferentes tags, como pode ser visto na Figura 14. Apesar do funcionamento constante e confiável durante os testes o sensor RFID também demostrou ter um alcance de detecção muito

curto (em torno de 5 cm), característica que poderia afetar negativamente a experiência de usuário na hora de utilizar o carrinho.

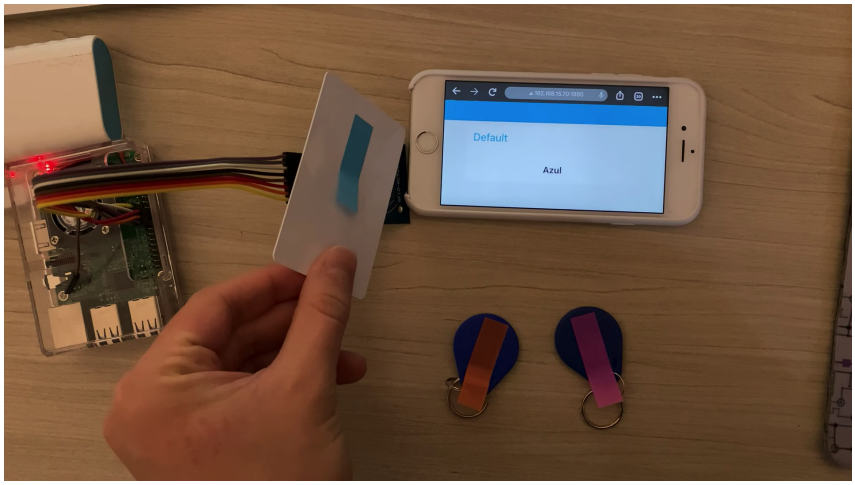

Figura 14: Testes de HW - Sensor RFID
Fonte: Autoria própria.

Os valores adquiridos inicialmente durante os testes do módulo Hx711 (sensor de peso) precisaram ser ajustados com um fator multiplicativo para que representassem valores mais próximos ao peso real dos objetos. Após a calibração com o fator multiplicativo o sensor de peso apresentou então resultados com precisão além da necessária para o escopo do nosso projeto. Na Figura 15 o objeto que estava sendo utilizado para os testes tinha 115 g. Na tela do computador é possível ver o valor medido pelo sensor.

Figura 15: Testes de HW - Sensor de peso (módulo Hx711)
Fonte: Autoria própria.

Em relação ao sensor ultrassônico, a Figura 16 mostra um gráfico com os valores de distancia detectados pelo sensor em função do tempo, enquanto o carrinho era movido sucessivamente para frente e para traz. O sensor apresentou resultados dentro do esperado pela equipe e suficientes para a aplicação de detecção de obstáculos do carrinho.

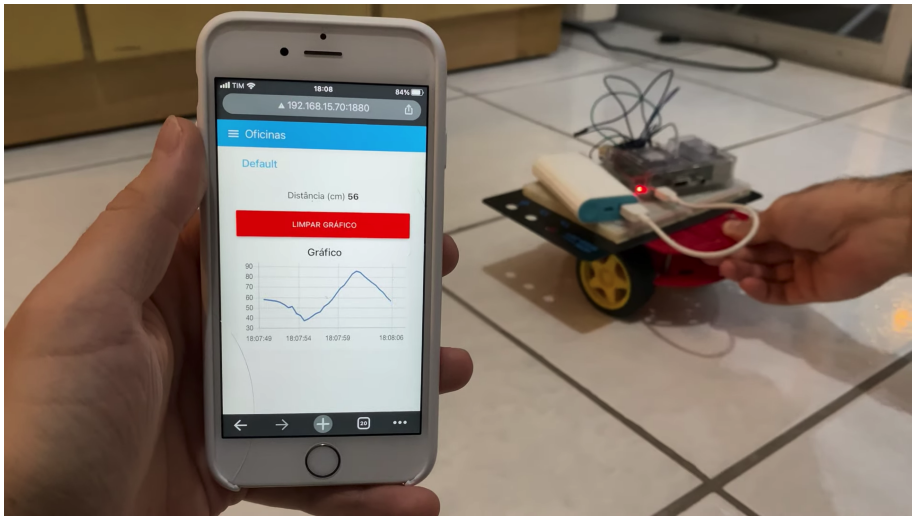

Figura 16: Testes de HW - Sensor ultrassônico

Fonte: Autoria própria.

Para o teste dos motores quatro comandos foram configurados - frente, ré, direita e esquerda - como pode ser visualizado na Figura 17. Todos os comandos funcionaram suficientemente bem para o escopo do nosso projeto, pelo menos durante essa etapa de testes isolados.

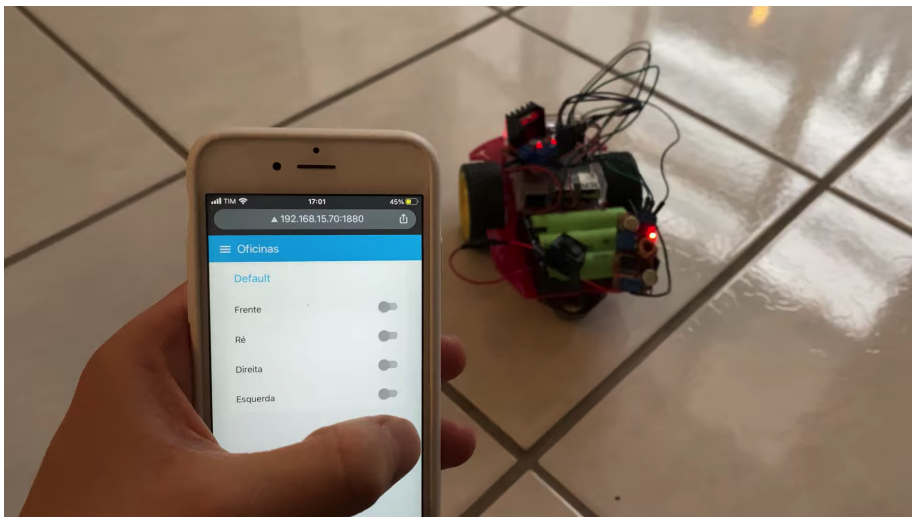

Figura 17: Testes de HW - Motores

Fonte: Autoria própria.

4.2 INTEGRAÇÃO

A última etapa de testes do semestre se iniciou com a fase de testes de integração. Principalmente no escopo de uma disciplina de integração essa etapa é a que de fato reflete todo o trabalho desenvolvido durante o semestre até então. Os resultados obtidos nessa parte do processo são diretamente relacionados a forma como o projeto foi concebido, planejado e validado durante o semestre. Também já esperávamos que esse seria o processo mais desafiador de todo o projeto, visto que a integração de todas as partes adiciona mais uma camada de complexidade ao projeto e introduz variáveis que afetam o desempenho do carrinho que antes não existiam

Os testes dessa etapa se iniciaram com a integração entre os elementos de software da Raspberry Pi e os sensores acoplados ao carrinho. Durante essa etapa de testes não foi considerado a funcionalidade de movimentação automática com reconhecimento de imagens e comunicação com o servidor. A Figura 18 é uma captura de tela do video gravado durante os testes. O maior desafio foi a sincronização entre o fluxo de interação do usuário com o carrinho e os dados obtidos pelos sensores. Percebemos que alterações mínimas nos valores das variáveis do código que rodava na Raspberry Pi tinham um grande impacto na experiência de usuário final na hora de utilizar o carrinho. Após alguns dias de testes e calibração chegamos a um resultado satisfatório para o nosso projeto.

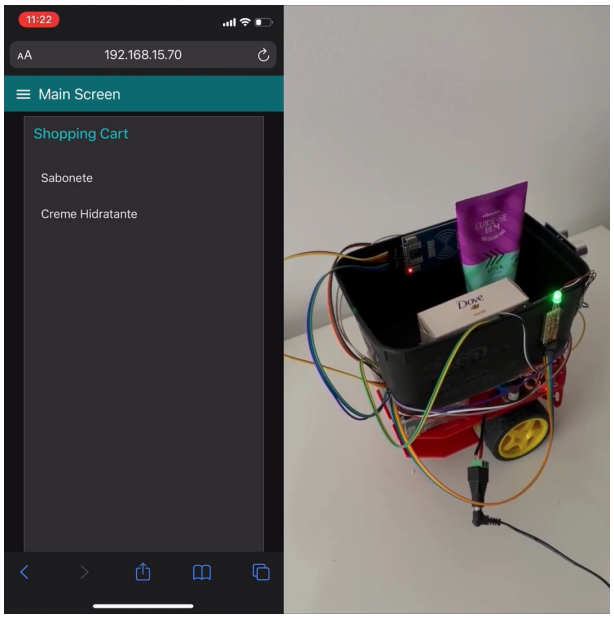

Figura 18: Integração - RPI e sensores

Fonte: Autoria própria.

Após isso iniciaram-se os testes de integração do algoritmo de reconhecimento de imagens e movimentação do carrinho. Essa foi a parte mais desafiadora de todo o projeto. A configuração inicial dessa fase de testes considerava alguns fatores que posteriormente percebemos ter grande impacto no desempenho do carrinho.

- Código de reconhecimento de imagens rodava em Python e tinha como saída apenas valores de posição dos quadrados coloridos.
- Esses valores eram capturados por um flow em Node-RED que acionava os motores pelo GPIO da Raspberry.
- O acionamento dos motores era feito por uma ponte H com um controle simples ON/OFF em malha aberta.

Os resultados dos testes com essa configuração inicial apresentaram diversos problemas. Como estávamos fazendo o controle em malha aberta e agora, com o carrinho devidamente montado, o motor tinha uma carga significativamente maior para suportar, a velocidade dos motores se mostrou completamente imprevisível e instável. O carrinho parecia começar a se mover na direção certa para seguir os quadrados coloridos na imagem mas perdia completamente a estabilidade logo em seguida, se movimentando de forma imprevisível. O carrinho também tinha dificuldade de se movimentar em terrenos que não eram perfeitamente lisos. Em diversos momentos até mesmo emperrava sem não conseguir mais sair do lugar.

Durante algumas semanas realizamos diversos testes e estudamos diferentes abordagens para resolver o problema. Após o estudo de alguns conceitos de controle e de conversa com os professores sobre os problemas encontrados, fizemos algumas modificações e ponderações sobre essa parte do projeto.

- Adição de multithreading no código em Python para acelerar a execução do algoritmo de reconhecimento de imagens.
- Acionamento dos motores deveria acontecer no código em Python, junto com o reconhecimento de imagens. Percebemos que o node-RED causava atrasos significativos na execução.
- Segundo regulador de tensão adicionado no carrinho exclusivamente para fornecer energia ao motor.
- Adição de 16 variáveis de controle no código em Python para calibração do movimento do carrinho.

- Uma luz forte era necessária para que o algoritmo de reconhecimento de imagens tivesse um melhor desempenho.
- O contraste da cor dos quadrados coloridos com o fundo da imagem também impactava significativamente o desempenho do algoritmo.

Na Figura 19 por exemplo é possível observar o flow em Node-RED que controlava todas as variáveis de controle adicionadas ao código em Python de reconhecimento de imagens. O nome de cada uma das variáveis pode ser visto nos blocos em azul da Figura 19.

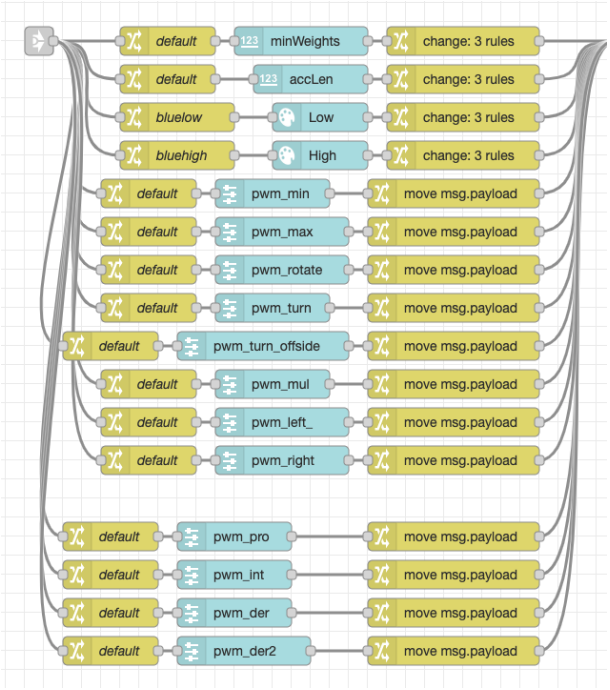

Figura 19: Integração - Variáveis de Controle

Fonte: Autoria própria.

4.3 RESULTADO FINAL

Após a realização de testes e ajustes durante todo o semestre nas partes de hardware e software do projeto, principalmente durante o período de testes integração, chegamos a versão final do EasyMarket.

As Figuras 20 e 21 são capturas de tela do video de entrega final do projeto. O video está também disponível na integra no blog da disciplina (CITTOLIN et al., 2021)

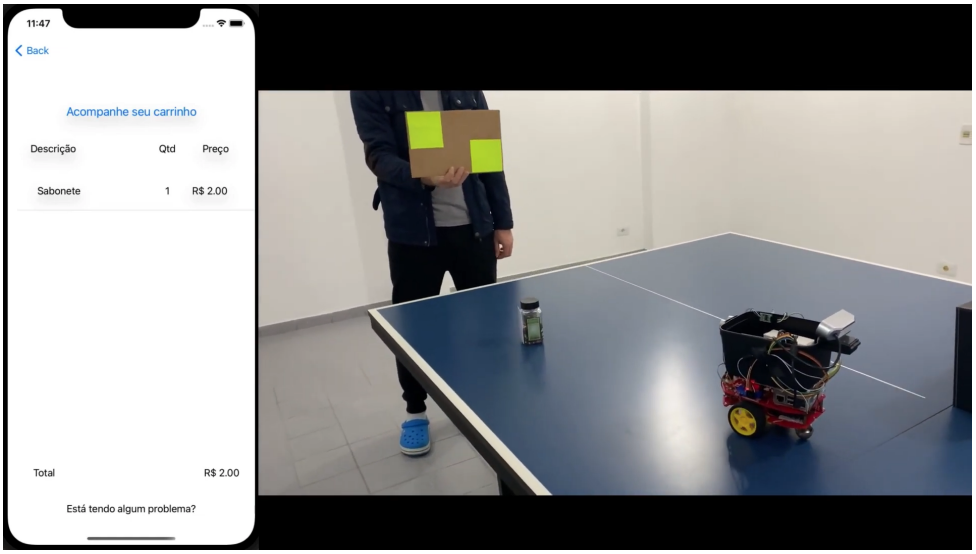

Figura 20: Captura de video - Entrega Final

Fonte: Autoria própria.

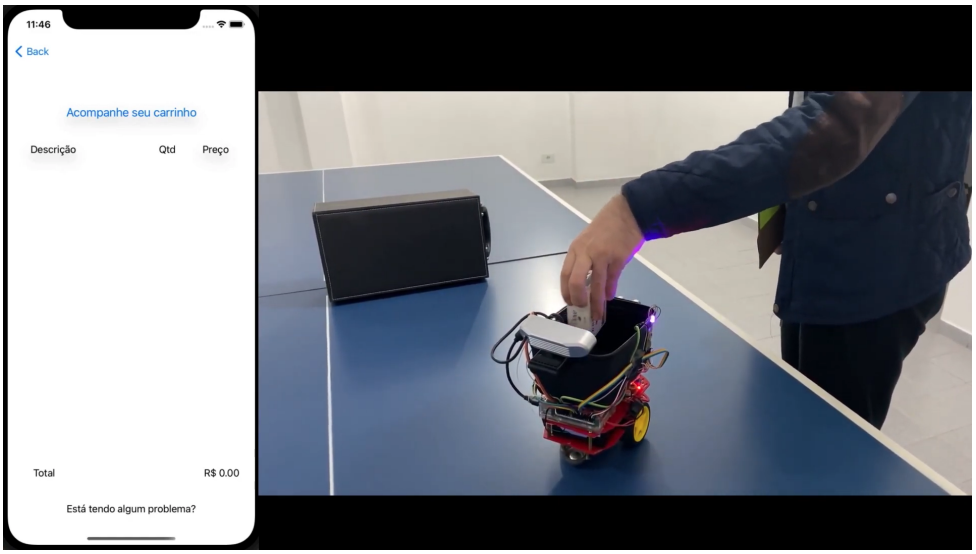

Figura 21: Captura de video - Entrega Final

Fonte: Autoria própria.

Como entrega final do projeto tivemos os seguintes resultados:

- O carrinho foi capaz de seguir o comprador automaticamente por reconhecimento de imagens
- O carrinho foi capaz de e comunicar com o servidor sobre início de compra e adição ou remoção de itens no banco de dados
- O carrinho foi capaz de identificar, por meio da combinação do sensor RFID e sensor de peso, que um item estava sendo adicionado ou removido do carrinho.
- O carrinho foi capaz de interromper sua movimentação frontal quando um obstáculo foi detectado a frente pelo sensor ultrassônico.
- O aplicativo foi capaz de se comunicar com o servidor para notificar o início de uma compra e consultar a lista de produtos do carrinho dinamicamente

Todos as versões e alterações dos códigos desenvolvidos e utilizados durante os testes podem ser consultadas no repositório GitHub do projeto (CITTOLIN et al., 2021).

5 CRONOGRAMA E CUSTOS DO PROJETO

5.1 CRONOGRAMA

Como pode ser visto nas figura 22, foi definido um cronograma de atividades bastante condensadas no período de 03 de março a 19 de maio de 2021. As atividades foram divididas em categorias: Geral, Projeto, Mecânico, Eletrônico, Software e Integração. Todas elas agendadas para cumprir o prazo dos entregáveis, datas em que os orientadores avaliaram o andamento do projeto, descritos na ultima seção do cronograma. Foram atribuídas horas previstas a cada atividade, totalizando 210 horas. Porém foi incluído uma folga de cerca de 30% a mais de horas sobre cada atividade. somado ao total, 275h. Até finalização deste relatório foram gastos cerca de 245h.

Para obtenção deste total foram somadas as horas gastas por cada integrante do projeto:

Tabela 2: Tabela mostrando a relação de horas gastas no projeto.

Membro	Horas Gastas
Pedro Francescon	105h
Ricardo Ardissone	80h
Viviane Moroni	60h
Total	R$ 245h

Fonte: Autoria própria.

A disparidade de horas entre os membros se deve principalmente a posse do carrinho para as tarefas de montagem e testes. Na figura 23 pode se ver a distribuição de atividades no tempo, bem como atrasos na inicialização (em vermelho) ou finalização (em lilás) de atividades.

	ID	Item	Duração	Duração	Pré-	Inicio	Fim	Status	Responsável	[horas atribuidas]				
Gerais	G01	Planejamento	25	21,0	-	03/03/2021	17/03/2021	Concluído (com a	Pedro	7,0	Viviane	6,0	Ricardo	8,0
	G02	Obter componentes	5	3,0	-	03/03/2021	31/03/2021	Concluído (com a	Pedro	3,0				
	G03	Atualização do site	25	16,0	-	03/03/2021	19/05/2021	Em andamento	Viviane	6,0	Pedro	8,0	Ricardo	2,0
	G04	Relatório final	10	11,0	I02	28/04/2021	19/05/2021	Em andamento	Viviane	3,0	Ricardo	4,0	Pedro	4,0
	G05	Editar vídeo para banca	8	10,0	I02	28/04/2021	12/05/2021	Concluído	Pedro	4,0	Viviane	5,0	Ricardo	1,0
Projeto	P01	Desenhar statechart	5	2,0	-	10/03/2021	17/03/2021	Concluído	Viviane	2,0				
	P02	Desenhar diagrama de blocos	5	2,0	-	10/03/2021	17/03/2021	Concluído	Ricardo	2,0				
	P03	Criar telas do app	10	4,0	P01	24/03/2021	07/04/2021	Concluído	Pedro	3,0	Viviane	1,0		
Mecânico	M01	Montar carrinho com motores	5	4,0	G02	24/03/2021	07/04/2021	Concluído	Pedro	3,0	Viviane	1,0		
	M02	Acoplar baterias, regulador de tensão e pont	3	3,0	M01	31/03/2021	07/04/2021	Concluído	Pedro	1,0	Ricardo	2,0		
	M03	Acoplar cesta	4	2,0	M02	31/03/2021	07/04/2021	Concluído	Viviane	2,0				
	M04	Acoplar à cesta a balança e o RFID	3	2,0	M03	31/03/2021	07/04/2021	Concluído	Viviane	1,0	Pedro	1,0		
	M05	Acoplar sensor ultrassonico	3	0,5	M04	31/03/2021	07/04/2021	Concluído	Viviane	0,5				
	M06	Acoplar camera	3	0,5	M04	31/03/2021	07/04/2021	Concluído	Viviane	0,5				
Eletrônico	E01	Fazer o carrinho andar	5	3,0	M02	31/03/2021	07/04/2021	Concluído	Pedro	2,0	Viviane	1,0		
	E02	Fazer o RFID funcionar	5	1,0	M04	31/03/2021	07/04/2021	Concluído	Pedro	1,0				
	E03	Fazer a balança funcionar	5	2,0	M04	31/03/2021	07/04/2021	Concluído	Pedro	1,0	Viviane	1,0		
	E04	Fazer sensor ultrassonico funcionar	5	1,0	M05	31/03/2021	07/04/2021	Concluído	Pedro	1,0				
	E05	Fazer a camera funcionar	5	3,0	M06	31/03/2021	07/04/2021	Concluído	Pedro	1,0	Ricardo	2,0		
	E06	Projeto de Hardware	5	2,0	M06	31/03/2021	07/04/2021	Concluído	Ricardo	2,0				
Software	S01	Programar celular	30	17,0	P03	07/04/2021	14/04/2021	Concluído	Viviane	15,0	Pedro	2,0		
	S02	Programar placa	20	45,0	G02	07/04/2021	14/04/2021	Concluído	Pedro	20,0	Ricardo	25,0		
	S03	Programar servidor	20	16,0	-	07/04/2021	14/04/2021	Concluído	Ricardo	16,0				
	S04	Prova de conceito (camera)	15	15,0	-	19/03/2021	24/03/2021	Concluído	Ricardo	6,0	Pedro	5,0	Viviane	4,0
Integração	I01	Conectar elementos	20	18,0	M, E e S	14/04/2021	28/04/2021	Concluído	Viviane	4,0	Pedro	10,0	Ricardo	4,0
	I02	Testes de integração	20	36,0	I01	28/04/2021	05/05/2021	Concluído	Pedro	28,0	Viviane	4,0	Ricardo	4,0
Entregáveis	#1	Plano de projeto	0,75	0,75	G01, P01	03/03/2021	17/03/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#2	Site de acompanhamento	0,75	0,75	G03	03/03/2021	24/03/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#3	Projeto/testes do hardware	0,75	0,75	#1, E	24/03/2021	07/04/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#4	Projeto do software	0,75	0,75	S	07/04/2021	14/04/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#5	Integração protótipo	0,75	0,75	I01	14/04/2021	28/04/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#6	Testes funcionais	0,75	0,75	I02	28/04/2021	05/05/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#7	Video para Banca	0,75	0,75	#6	05/05/2021	12/05/2021	Concluído	Pedro	0,25	Ricardo	0,25	Viviane	0,25
	#8	Relatorio final	0,75	0,0	G04	28/04/2021	19/05/2021	Em andamento	Pedro	0,0	Ricardo	0,0	Viviane	0,0

Figura 22: Cronograma - Lista de Atividades

Fonte: Autoria própria.

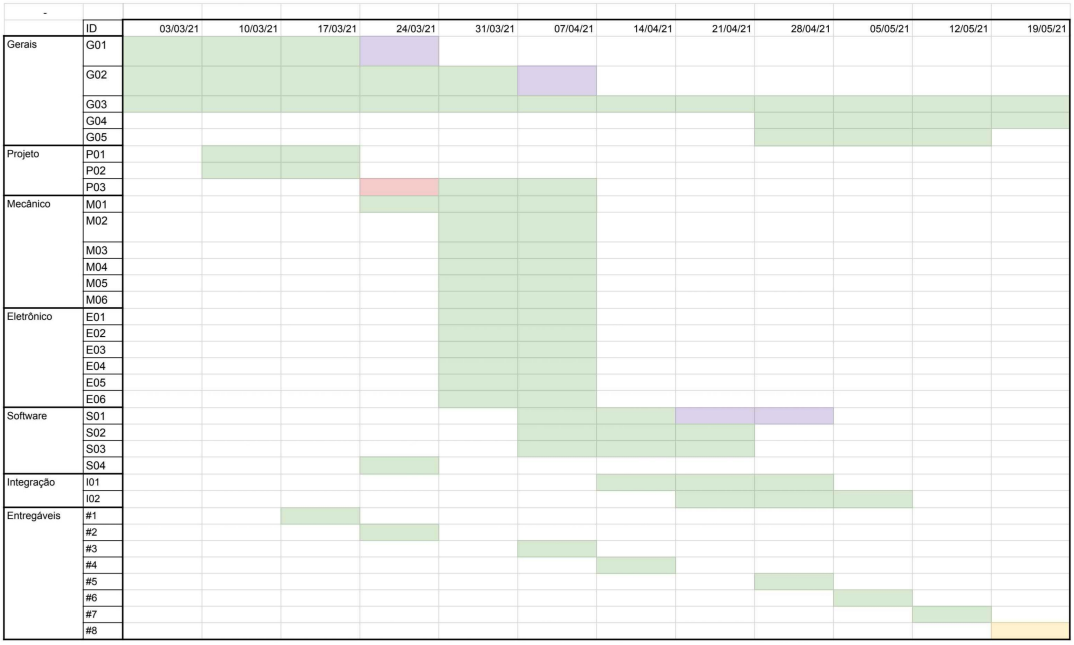

Figura 23: Cronograma - Diagrama de Gantt

5.2 CUSTOS

Para contabilizar os gastos financeiros do projeto, listamos na tabela 3 Os itens a serem adquiridos e seu custo ou ”NA”se por algum motivo não foi necessário sua compra.

Tabela 3: Tabela mostrando a relação de custos do projeto.

Item	Valor
Raspberry Pi 3 modelo B	NA
2 Motores DC 12V com rodas	NA
Sensor MPU 6050	R$ 12,30
Kit de baterias para o motor	R$ 40,00
3 Sensores de Distância HC-SR04	R$ 32,97
Chassi	NA
Câmera Microsoft Lifecam Vx-800	NA
Ponte H L298N	R$ 13,00
Custo de envio estimado	R$ 20,00
Custo de deslocamento de materiais	R$ 45,00
Total	R$ 163,27

Fonte: Autoria própria.

O item custo de deslocamento de materiais se refere ao transporte da posse do carrinho para condução de testes e melhorias entre os membros.

6 CONCLUSÕES

Constatamos no desenvolver do projeto que a implementação do Controle sem uso de *encoder* não é recomendável. Para implementar o Controle nos movimentos do motores do carrinho era necessária uma maior velocidade no ciclo, Outra constatação interessantes foi a dificuldade imprevista na construção de robôs com ampla movimentação. Uma dificuldade encontrada foi a integração das partes do software serem prejudicadas pela falta de tempo e alinhamento de desenvolvimento, já que toda a comunicação do grupo foi assíncrona. Acessibilidade e manutenibilidade, Alguns pequenos problemas como a necessidade de trocar de baterias facilmente tiveram impacto relevante. Altas abstrações e camadas de SW tem custos de processamento e podem serem imprevisíveis ou surtirem problemas caso sobrecarregas

6.1 TRABALHOS FUTUROS

Uma possibilidade de implementação futura, com um custo elevado, seria adequar o tamanho do carrinho para o tamanho real de um carrinho de compras de supermercado, com a necessidade de aumentar a força dos motores, tamanho da bateria, entre outros. Para isso seria adequado também haver um ambiente de teste igual ou muito semelhante ao de corredores de um supermercado. Outra melhoria mecânica seria melhorar a montagem, com menos fios expostos, com uma montagem mecânica pre-planejada e construída ou adaptada especificamente para esse projeto.

Um RFID mais potente, com alcance maior, poderia eliminar a necessidade de passar individualmente cada item pelo sensor RFID, bastando por o item no carrinho.

Em termos de controle, tanto a logica de controle quanto os sensores poderiam ser melhorados. Primeiro, o uso de *encoder* permitiria adicionar um outro ciclo de controle PID, com um ajuste preciso da velocidade. Segundo, o controle via câmera ainda cabe melhorias e ajustes, provavelmente requirindo um embasamento teórico mais avançado. Terceiro, a detecção da câmera pode ser melhorada de varias formas. Uma detecção menos sensível a iluminação, menos propensa a confundir outros objetos da mesma cor com o alvo. Acelerar

ainda mais a leitura e processamento da imagem deixaria o ciclo de controle mais curto, melhorando o comportamento do sistema. Isso poderia ser feito por exemplo movendo a implementação para o C++.

REFERÊNCIAS

APPLE INC. **Xcode**. Disponível em: <https://developer.apple.com/xcode/>.

AVIA SEMICONDUCTOR. **hx711F**. Disponível em: <https://cdn.sparkfun.com/assets/b/f/5/a/e/hx711F_EN.pdf>.

BAYER, F. M.; ARAúJO, O. C. B. d. **Controle Automático de Processos**. [s.n.], 2011. Disponível em: <https://www.ufsm.br/app/uploads/sites/413/2018/11/11_controle_automatico_processos.pdf>.

BAYER, M. **SQLAlchemy: Database Abstraction Library**. Disponível em: <http://www.sqlalchemy.org>.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, 2000.

BUILD a digital Raspberry Pi Scale (with Weight Sensor HX711). Section: Hardware & GPIO. Disponível em: <https://tutorials-raspberrypi.com/digital-raspberry-pi-scale-weight-sensor-hx711/>.

CITTOLIN, P. F.; ARDISSONE, R. L. S.; SOUZA, V. L. B. Moroni de. **EasyMarket**. maio 2021. Original-date: 2021-04-12T22:17:54Z. Disponível em: <https://github.com/pedrofrancescon/EasyMarket>.

CROSTON, B. **RPi.GPIO: A module to control Raspberry Pi GPIO channels**. Disponível em: <http://sourceforge.net/projects/raspberry-gpio-python/>.

DIAGRAMS.NET. **diagrams.net**. Disponível em: <https://app.diagrams.net/>.

ELEC FREAKS. **HCSR04**. Disponível em: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>.

GOODDAY451999. **Multiple Color Detection in Real-Time using Python-OpenCV**. abr. 2020. Section: Python. Disponível em: <https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/>.

GUS. **How to setup a Raspberry Pi RFID RC522 Chip**. out. 2017. Disponível em: <https://pimylifeup.com/raspberry-pi-rfid-rc522/>.

HANDSON TECHNOLOGY. **L298N Motor Driver module**. Disponível em: <https://www.handsontec.com/dataspecs/module/L298N%20Motor%20Driver.pdf>.

HAREENDRAN, T. **XL4015 Step-Down DC Module with CV/CC Control - Quick Review**. fev. 2020. Disponível em: <https://www.electroschematics.com/dc-module/>.

HARRIS, C. R. et al. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

HEYWOOD, M. A. **Bluetin-Echo: Raspberry Pi Python Library for Ultrasonic Module HC-SR04 Distance Measuring Transducer.** Disponível em: <https://github.com/MarkAHeywood/Bluetin_Python_Echo>.

HIPP, R. D. **SQLite**. 2020. Disponível em: <https://www.sqlite.org/index.html>.

HOLISTICS SOFTWARE. **dbdiagram.io - Database Relationship Diagrams Design Tool**. Disponível em: <https://dbdiagram.io/home>.

JECHLITSCHEK, C. Radio Frequency IDentification - RFID. 2006. Disponível em: <https://www.cse.wustl.edu/~jain/cse574-06/ftp/rfid/index.html>.

MOTOR DC 3-6V com Caixa de Redução e Eixo Duplo. Disponível em: <https://www.filipeflop.com/produto/motor-dc-3-6v-com-caixa-de-reducao-e-eixo-duplo/>.

NXP B.V. **MFRC522 - Contactless Reader IC**. 2007. Disponível em: <https://circuits4you.com/wp-content/uploads/2018/10/MFRC522-Datasheet.pdf>.

OPENAPI INITIATIVE. **OpenAPI Specification - Version 3.0.3**. Disponível em: <https://swagger.io/specification/>.

OPENAPITOOLS. **OpenAPI Generator**. Disponível em: <https://openapi-generator.tech/>.

OPENJS FOUNDATION. **Node-RED**. Disponível em: <https://nodered.org/>.

O'HANLON, M. **Going Straight with PID - Introduction | Raspberry Pi Projects**. Disponível em: <https://projects.raspberrypi.org/en/projects/robotPID>.

RASPBERRY PI FOUNDATION. **Raspberry Pi Documentation**. Disponível em: <https://www.raspberrypi.org/documentation/>.

RONACHER, A. **Flask: A simple framework for building complex web applications.** Disponível em: <https://palletsprojects.com/p/flask>.

RONACHER, A. **Flask-SQLAlchemy: Adds SQLAlchemy support to your Flask application.** Disponível em: <https://github.com/pallets/flask-sqlalchemy>.

ROSEBROCK, A. **Find distance from camera to object using Python and OpenCV**. jan. 2015. Disponível em: <https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>.

ROSEBROCK, A. **Increasing webcam FPS with Python and OpenCV**. dez. 2015. Disponível em: <https://www.pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv/>.

SKETCH B.V. **Sketch**. Disponível em: <https://www.sketch.com/>.

SMITH, S. W. **The Scientist and Engineer's Guide to Digital Signal Processing**. [s.n.], 1997. Disponível em: <http://www.dspguide.com/ch25/4.htm>.

SQLITE. **Appropriate Uses For SQLite**. Disponível em: <https://www.sqlite.org/whentouse.html>.

STMICROELECTRONICS. **L298 - Dual full-bridge driver**. [S.l.], 2000. Disponível em: <https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf>.

VIJAY, V. **Bloo Free Wireframe UI Kit**. Disponível em: <https://www.uistore.design/items/bloo-free-wireframe-ui-kit/>.