Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (EEX23) – S71 – 2020/1

Relatório Técnico
Medicine Deployer

Cássio G. Morales – cassiomorales@alunos.utfpr.edu.br
Cleverton Carneiro – clevertoncarneiro@alunos.utfpr.edu.br
Guilherme Rossi – guilhermerossi94@hotmail.com
Julio C. Penha – juliopenha@alunos.utfpr.edu.br
Luiz R. N. Agner – luizagnern@hotmail.com

Outubro de 2020

Resumo

A administração de medicamentos é uma das principais atividades de um profissional de enfermagem. Se o ato de ministrar os medicamentos não for realizado corretamente, consequências graves poderão ser acarretadas ao paciente. Com o intuito de minimizar esse erro, foi desenvolvido o Medicine Deployer, que é uma solução que visa o melhor gerenciamento do aprazamento de medicamentos, além da automatização do sistema de entregas de medicamentos entre a farmácia e as alas de um hospital. Utilizando as ferramentas fornecidas pelo Medicine Deployer, o usuário, a depender do nível de acesso, poderá fazer o cadastro de pacientes e medicamentos e enviar uma ordem de entrega para um robô controlado por uma Raspberry Pi. Esse robô seguirá uma linha colorida referente à ala em que o paciente se encontra e enviará notificações ao servidor a fim de alimentar um histórico de entregas e a quantidade disponível de cada medicamento. O projeto também é responsável por implantar um sistema de segurança compostos por gavetas e leitores de RFID, garantindo que apenas usuários autorizados possam ter acesso aos medicamentos.

1 Introdução

A omissão ou atraso na ministração de medicamentos é um dos erros mais registrados em estudos epidemiológicos sobre erros de medicação e em sistemas de notificação de incidentes hospitalares. Dependendo do tipo de patologia e de medicamento, esse tipo de erro pode causar consequências clínicas graves e até levar ao óbito do paciente. Por exemplo, a omissão de um antibiótico a um paciente com sepse ou de um anticoagulante a um paciente com embolia pulmonar pode ter consequências gravíssimas [1] [2].

Com o objetivo de reduzir esse comum erro, foi criado o Medicine Deployer, que é um sistema desenvolvido que facilita e automatiza a distribuição de remé-

dios entre as alas dentro de um hospital, visando a ministração dos medicamentos nos horário corretos e a redução de esforço dos enfermeiros com a tarefa de agendar o transporte dos medicamentos.

O sistema é composto de três partes principais, o software no qual os remédios de cada paciente são registrados (ilustrado na Figura 1), o robô, adaptado com um cofre, responsável pelo transporte dos medicamentos (ilustrado na Figura 2) e um servidor responsável por tratar de todas as requisições do software e do robô, possibilitando assim sua comunicação. Além disso, para o correto funcionamento, é necessário adaptar o hospital com linhas coloridas no chão para que o robô possa encontrar seu caminho até as alas.

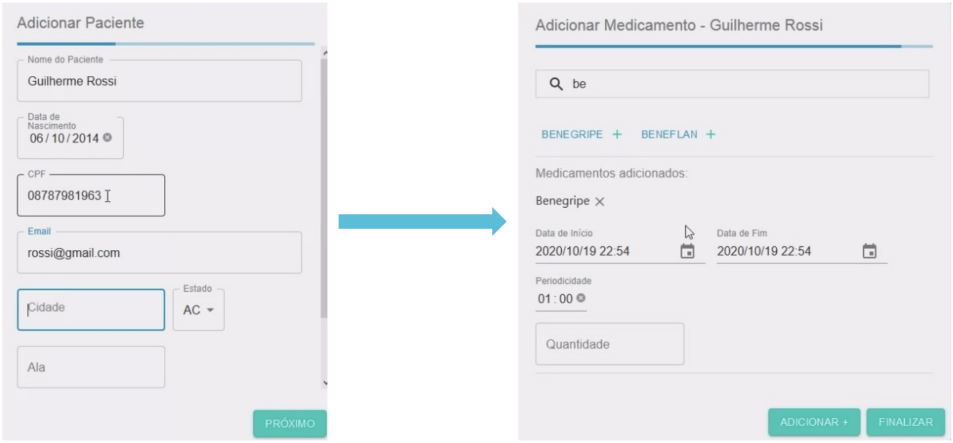

Figura 1: Cadastro de paciente e medicamento na aplicação web.

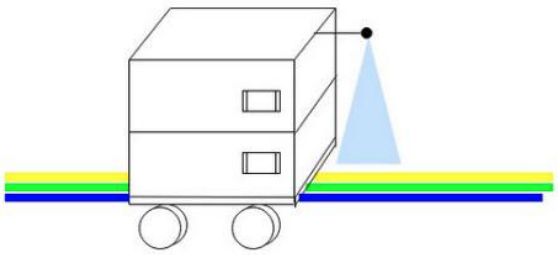

Figura 2: Robô seguidor de linha.

No software é possível fazer login com três tipos de usuário: administrador, farmacêutico e médico. O administrador é responsável por cadastrar médicos, farmacêuticos e fazer o controle de estoque dos medicamentos. Farmacêuticos são responsáveis por despachar medicamentos quando houver ordens de envio criadas. Por fim, os médicos são responsáveis por cadastrar pacientes e fazer a prescrição de medicamentos para eles. Cada uma dessas funcionalidades está

disponível através de telas CRUD (acrônimo para **C**reate-**R**ead-**U**pdate-**D**elete, que são operações realizadas em bancos de dados) na aplicação web.

O robô é responsável por transportar os medicamentos da farmácia do hospital até a ala designada. Para tal, ele consulta o servidor a cada 1 segundo para verificar se há alguma ordem despachada, ou seja, que já esteja presente dentro do cofre. Se houver, com base nas informações recebidas do servidor sobre os remédios que se encontram em seu cofre, o robô inicia sua corrida seguindo a linha de cor correspondente à ala designada.

O servidor é responsável por receber as requisições provenientes de todos os CRUDs anteriormente citados e também faz o cálculo dos horários em que cada remédio deve ser enviado, criando assim ordens que vão ser adicionadas ao robô. Além disso, também faz o cálculo de estoque dos medicamentos, subtraindo a quantidade de um determinado medicamento assim que a ordem é finalizada ou a adição de medicamentos quando o administrador cadastra uma nova quantidade.

A Figura 3 apresenta um diagrama geral do sistema, no qual é possível observar as relações entre as entidades anteriormente citadas.

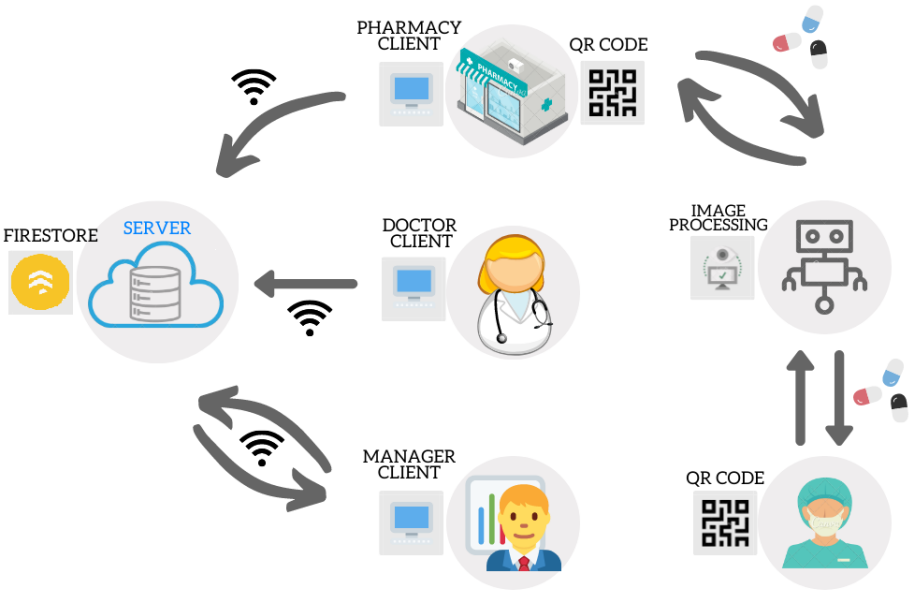

Figura 3: Diagrama geral do sistema

2 Servidor Base

O servidor foi desenvolvido para ser desacoplado da aplicação web e do software embarcado do robô, de tal forma que ambas aplicações possam interagir com o servidor sem que ele se preocupe com a implementação particular de cada módulo. Este modelo é útil, pois possiblita uma futura integração com outros sistemas e torna o projeto facilmente expandível no futuro. Para atingir este objetivo, foi desenvolvida uma API que responde a requisições HTTP, recebendo e enviando dados através de um formato de arquivo JSON [3]. A Tabela 1 mostra os requisitos que o servidor deve apresentar.

Tabela 1: Requisitos do servidor.

Requisito	Descrição
FR01	O servidor precisa ter uma camada de autenticação.
FR02	O servidor precisa diferenciar cargos de usuários.
FR03	O servidor precisa ter endpoints com funções CRUD para todas as entidades do banco de dados.
FR03.1	Estes endpoints precisam responder a chamadas HTTP.
FR04	O servidor precisar calcular lotes de medicamentos a serem enviados.
FR04.1	Este cálculo precisa ser feito toda vez que o endpoint é chamado.
FR04.2	Os lotes precisam ter informações sobre o medicamento e a quantidade a ser enviada, assim como a ala e a qual paciente se destina.
FR05	O servidor precisa ter um endpoint através do qual o robô irá alertar quando uma ordem tiver sido entregue.
NFR01	Firebase deverá ser utilizado como backend-as-a-service.

A API do sistema foi desenvolvida utilizando Node.js, que é um ambiente de execução JavaScript para o servidor. Ele fornece ferramentas para que um código JavaScript seja executado fora do ambiente de um navegador, ou seja, no próprio servidor, o que torna o desenvolvimento bastante fluido. A API é mantida utilizando os serviços do Firebase, que é uma plataforma de desenvolvimento disponibilizada pela Google, que engloba diversas ferramentas que auxiliam no desenvolvimento de aplicações web, como por exemplo análise de dados, autenticação de usuário, banco de dados, armazenamento de arquivos e a execução de código propriamente dito. Para tirar maior proveito do que a ferramenta do Google tem a oferecer, foi criado um banco de dados Cloud Firestore, que é um banco de dados NoSQL embutido na própria plataforma do Firebase. Em uma base de dados não relacional, "...os dados são representados como uma coleção de pares de valores-chave"[4]. Um conjunto de pares de valores-

chave são organizados em documentos representados por uma chave única gerada na adição de um documento a uma coleção. Esses documentos "...são geralmente armazenados em uma estrutura similar ao JSON"[4]. O banco de dados orientado a documentos foi escolhido pois os dados oriundo das consultas podem ser juntados manualmente, sem a necessidade de se utilizar JOINs, além de não haver a necessidade de um gerenciamento de múltiplas transações, já que a aplicação não requer isso.

Primeiramente, foram modeladas todas as entidades e as relações entre elas. Após analisar o problema no mundo real, foram definidas quatro entidades mínimas necessárias para a construção da aplicação: usuário, medicamento, paciente e ordem. O usuário é a entidade responsável por conter os dados dos usuários e suas diversas funções no sistema: médico, farmacêutico e administrador. O medicamento é responsável por guardar as informações dos remédios disponíveis, assim como a quantidade disponível e a quantidade já utilizada. O paciente, como o próprio nome já diz, guarda informações sobre o paciente, sendo as mais importantes a ala em que ele se encontra e a prescrição medicamentosa cadastrada pelo médico. Por último, a ordem carrega informações sobre o medicamento a ser entregue, assim como a qual ala e a qual paciente especifico. Tendo isso definido, foi construído um diagrama de entidade-relação (ER), mostrado na Figura 4. Apesar de se tratar de um banco de dados NoSQL, ainda faz sentido utilizar diagramas ER para melhor ilustrar as entidades, pois a princípio a modelagem de dados ainda continua a mesma, o que muda é a organização desses dados dentro dos arquivos no banco de dados [5] [6].

Após definida a estrutura do banco de dados, foi necessário implementar endpoints CRUDs para todas estas entidades, ou seja, funções acessíveis através de requisições HTTP que permitem criar, ler, atualizar e deletar dados. Além disso, foram criadas outras funcionalidades responsáveis por verificar se há medicamentos a serem entregues e também para atualizar a quantidade de medicamentos no estoque após a finalização de uma entrega. Todas estas funções são implantadas e disponibilizadas publicamente através da ferramenta Firebase Cloud Functions, que atua como um framework serverless e responde a eventos disparados através de requisições HTTP.

Para garantir a segurança do sistema como um todo, foi utilizado um serviço chamado Firebase Authentication, que está embutido no próprio Firebase. Esta ferramenta oferece suporte à autenticação utilizando email e senha, que fica integrado à API do sistema, fornecendo um sistema de geração de tokens de segurança para o gerenciamento de acesso à API.

3 Aplicação Web

A interface de usuário da aplicação web foi criada utilizando o React, que é uma biblioteca de JavaScript voltada ao desenvolvimento de interfaces de usuário e componentes gráficos. Escolheu-se utilizar o React para que componentes web,

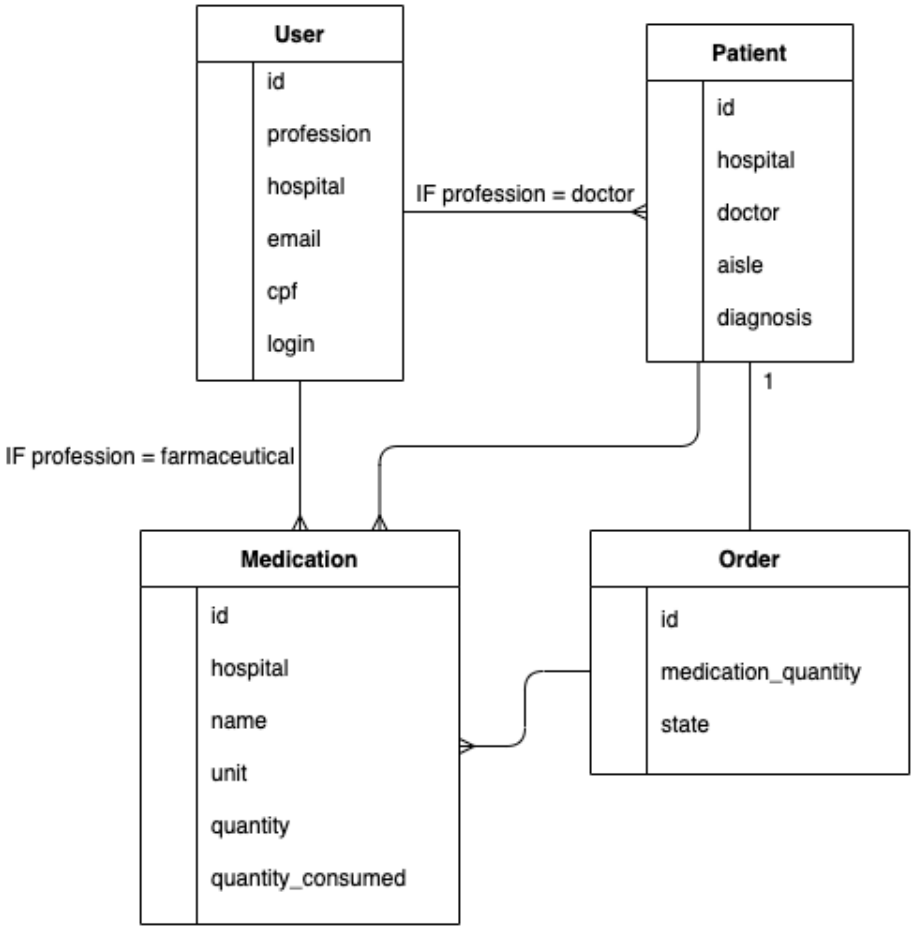

Figura 4: Diagrama entidade-relação do banco de dados

que podem gerenciar seus próprios estados, fossem reutilizados nos diferentes níveis de acesso dos usuários. O React também foi escolhido pois ele permite reconstruir o DOM (acrônimo para **D**ocument **O**bject **M**odel, que é uma representação orientada a objetos de uma página web) e publicar apenas as mudanças efetivas, sem a necessidade de se recarregar a página [7]. A Tabela 2 mostra os requisitos que a aplicação web deve apresentar.

Após a avaliação dos requisitos, foram implementados os seguintes componentes web:

- Tela de cadastro para médico, farmacêutico e administrador;
- Tela de login para médico, farmacêutico e administrador;
- Formulário para cadastro de paciente e adição de medicamentos para paciente;
- Tabela "Lista de Pacientes"para o médico;

Tabela 2: Requisitos da aplicação web.

Requisito	Descrição
FR01	A interface de usuário deve ser uma aplicação web.
FR02	A aplicação deve ter uma página de cadastro para o administrador cadastrar os usuários na plataforma.
FR03	A aplicação deve ter uma página de login para médicos.
FR04	A aplicação deve ter uma página de login para farmacêuticos.
FR05	A aplicação deve ter um dashboard para os farmacêuticos.
FR06	A aplicação deve possibilitar o envio de ordens criadas.
FR06.1	A aplicação deve ter uma mensagem de confirmação com um botão para evitar enviar uma ordem acidentalmente.
FR07	A aplicação deve permitir ao médico adicionar um paciente através de uma página de cadastro.
FR07.1	Esta página deve conter um formulário que permite ao médico inserir todas as informações pertinentes ao paciente.
FR08	A aplicação deve mostrar ao administrador as informações sobre quantidades de medicamentos.
FR09	A aplicação deve ter um sistema de notificação para o farmacêutico.
FR10	A aplicação deve mostrar histórico de ordens entregues.

- Tabela "Lista de Pedidos"para o farmacêutico;
- Tabela "Lista de Medicamentos"para o administrador;
- Tela "Envio de Pedido"para o farmacêutico;
- Tela "Log de Pedido"para o administrador;
- Navbar e sidebar;
- Tabela "Lista de Pedidos Entregues"para o administrador (Figura 5).

4 Estrutura Robótica

O robô foi idealizado, primeiramente, conforme os requisitos da Tabela 3 e seu modelo 3D está representado na Figura 6. Na primeira versão, o robô tinha três sensores ultrasônicos para detecção de obstáculos, mas, devido à pândemia, os requisitos FR01 e FR02 foram ajustados para possibilitar sua construção, além de que o requisito RF05 e NFR03 foram removidos.

O tamanho do robô (FR01) foi ajustado para 25 cm x 20 cm x 26 cm. O material (FR01) utilizado para fabricação da estrutura foi papelão, ao invés de MDF.

O robô se move utilizando um Kit Chassi 2WD (Figura 7). O kit tem duas rodas com motores DC e um rodízio giratório, que dá sustentação ao robô e

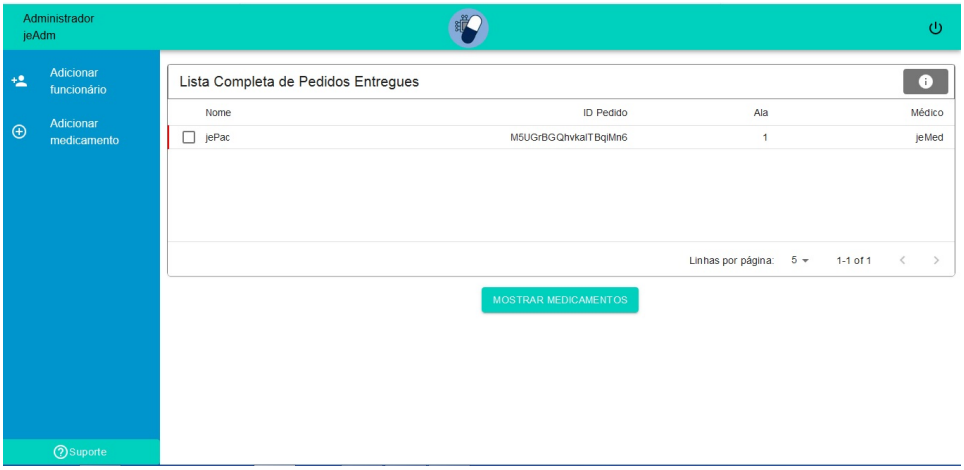

Figura 5: Tela de Lista de Pedidos Completos

permite que ele faça seus movimentos.

Nesta versão, como o mecanismo de trava não existe, o compartimento em que o farmacêutico deposita os medicamentos foi removido e apenas modelado em software, como apresentado anteriormente na Figura 6. O robô montado, sem o mecanismo de travas, está representado na Figura 8.

Na parte de superior do robô há um botão para que o robô possa começar ou finalizar uma entrega, e assim, ir até uma ala ou voltar para a farmácia. Há uma tela LCD para indicar o estado da entrega que pode ser vista na Figura 9.

Para fazer a captura de imagem, foi utilizado um módulo de câmera para Raspberry Pi v1.3. Esta câmera está acoplada a uma haste de papelão posicionada na parte frontal (Figura 10). A câmera foi posicionada de maneira a ficar apontada para o chão, para que o robô possa enxergar as linhas coloridas.

Tabela 3: Requisitos do robô.

Requisito	Descrição
FR01*	O tamanho do robô deve ser de 45 cm x 20 cm x 20 cm.
FR02*	A Estrutura do robô deve ser feita de MDF.
FR03	O robô deve ter duas rodas com motores DC e um rodízio giratório.
FR04	O robô deve ter uma câmera
FR04.1	A câmera precisa estar apontada para o chão.
FR05*	O robô deve ter um mecanismo de trava.
FR06	O robô deve ter um módulo Wi-Fi.
FR07	A fonte de alimentação do robô deve ser um Power Bank de 20000mAh, 5V e 3A, e a fonte de alimentação dos motores deve ser de 4 baterias de Li-Ion em paralelo .
FR08	O robô deve ter um botão.
NFR01	O computador que controla o robô deve ser uma Raspberry Pi 3B+.
NFR02	O módulo RFID deve ser baseado no chip MFRC522.
NFR03	As travas devem ser travas elétricas solenóides.

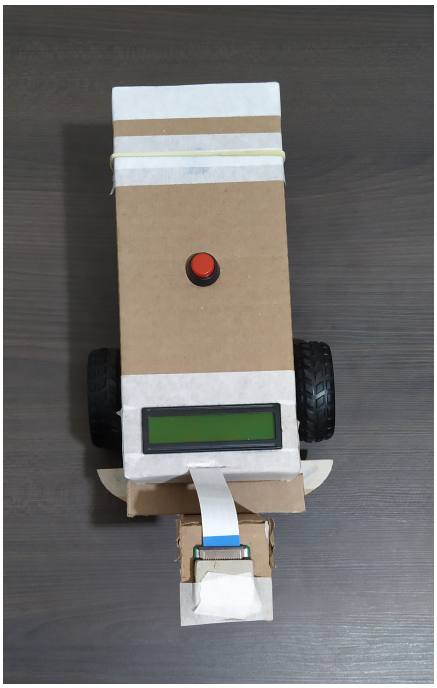

Figura 9: Vista superior do robô.

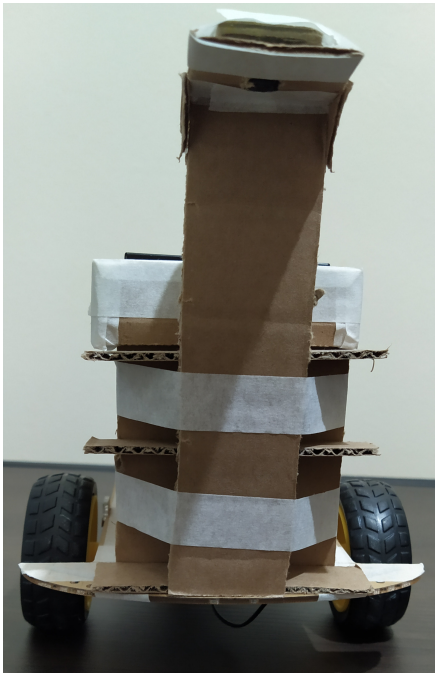

Figura 10: Vista frontal do robô.

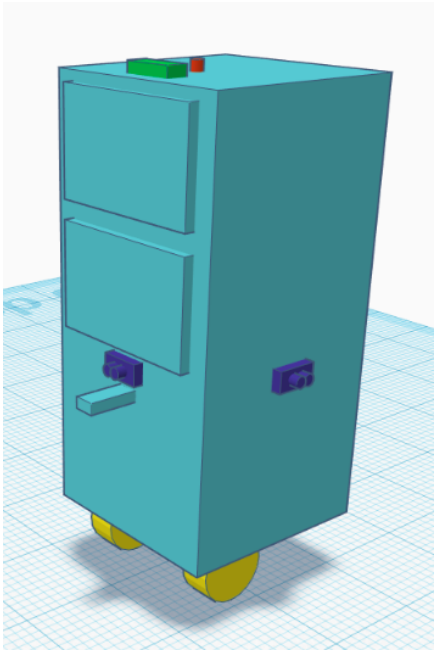

Figura 6: Primeira versão do Robô.

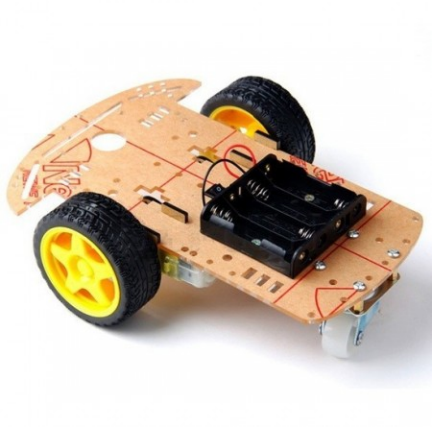

Figura 7: Kit Chassi 2WD.

Dentro do corpo do robô, delimitado por papelão, estão conectados o Power Bank para alimentação da Raspberry Pi, a Raspberry Pi, as baterias Li-Ion para alimentação do robô e ponte H para acionamento dos motores. Parte dos componentes pode ser vistos na Figura 11.

Na tampa do robô (Figura 12) estão conectados um leitor RFID, um buzzer para avisar que o robô chegou ao destino e um botão para iniciar ou finalizar o processo de entrega.

Também há um chicote elétrico para conectar os sensores e atuadores à Raspberry Pi, todos os componentes estão ligados conforme o diagrama de conexões da Figura 13.

A Figura 14 dá uma ideia do posicionamento dos componentes na parte interna do robô, vista pelo lado de fora.

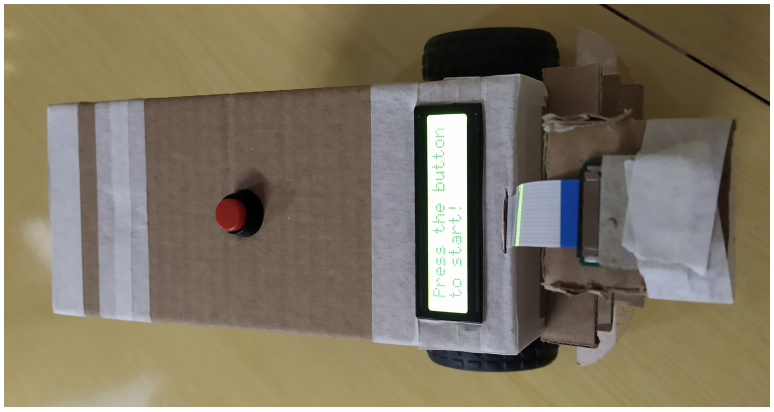

Figura 8: Robô montado.

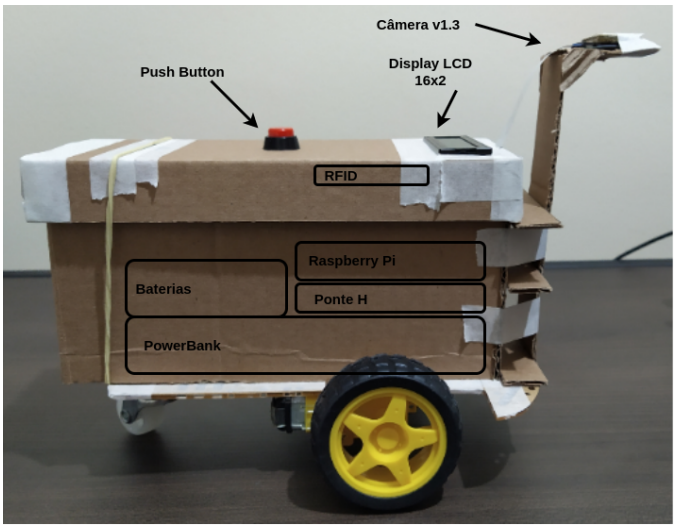

Figura 14: Posicionamento dos componentes.

Na versão pré-pandemia, foi desenhada uma placa de circuito impresso (PCB) utilizando o software de design de circuitos impressos Eagle, que pode ser visto na Figura 15 e 16. Devido às mudanças de escopo e redução do tempo disponível, a placa não foi produzida.

Por fim, com o robô montado, foi feita a pista que liga a farmácia às alas Figura 17. As linhas foram feitas utilizando tiras de papel colorido coladas em papel sulfite branco. No fim de cada pista há um QR Code para que o robô saiba se está na ala correta ou se chegou à farmácia.

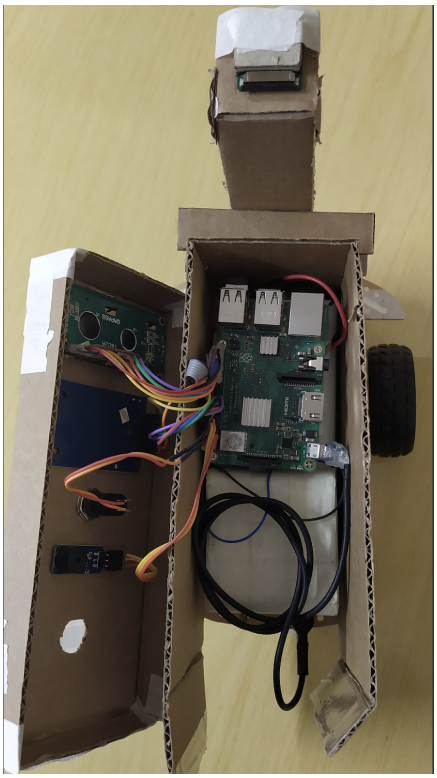

Figura 11: Robô com a tampa aberta.

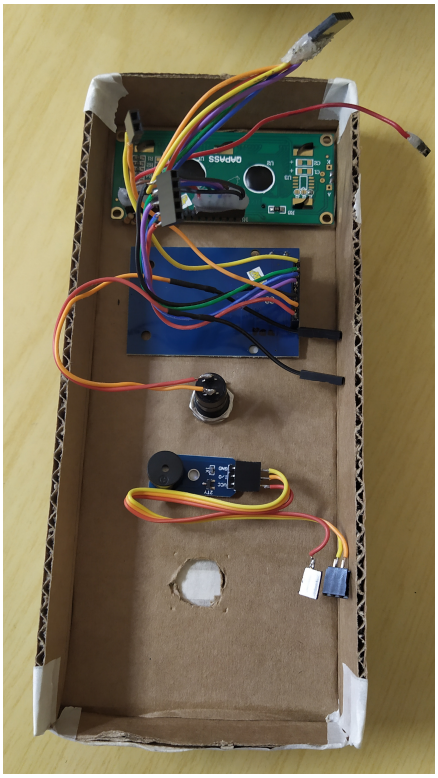

Figura 12: Componentes da tampa.

5 Visão Computacional

O algoritmo de visão computacional do robô é separado em três partes discutidas separadamente nas sub-seções seguintes

- Leitor de QR Code
- Código auxiliar para medir intervalos dos canais HSV
- Código seguidor de linha

5.1 Leitor QR Code

Depois de percorrido o trajeto, algo deve indicar que o robô já chegou ao final do percurso, em forma de um símbolo indicativo, ou seja, um método para indicar e informar a ala em que o robô se encontra, por isso a escolha do QR code. Além de conter informações em forma de texto e links, é um método com um grau de segurança contra danos, muito útil quando se encontra no chão onde circulam pessoas, equipamentos, macas, etc.

Com a utilização da biblioteca OpenCV, é possível fazer a chamada da fun-

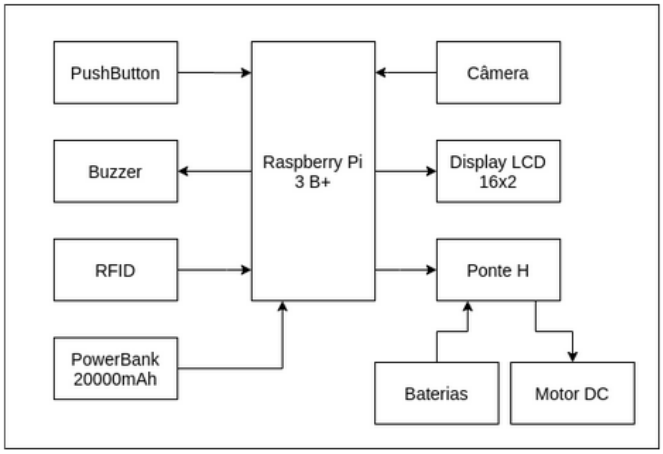

Figura 13: Diagrama de blocos das conexões do sistema embarcado.

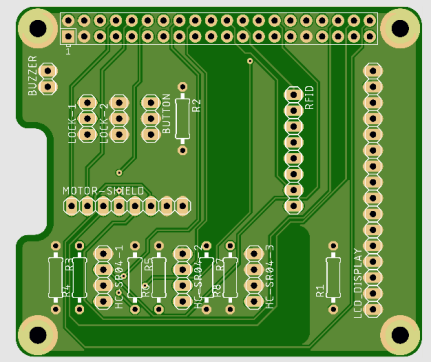

Figura 15: Vista superior da placa.

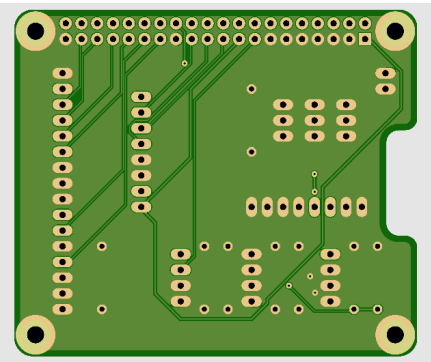

Figura 16: Vista inferior da placa.

ção "QRCodeDetector". A partir dessa chamada, o módulo de detecção é inicializado e retorna a informação contida nele. Com isto, pode-se fazer a leitura de um código, bastando portanto apenas tratar as informações dessa leitura para delegar o que deve ser feito em seguida.

5.2 Canais HSV

O HSV é uma representação alternativa ao modelo RGB (Red, Green and Blue) que foi desenvolvida em 1970 para que fosse mais próximo da forma como o olho humano processa imagens. O HSV separa a imagem em três canais, sendo eles o **H**ue, que modela a resultante de várias cores misturadas, **S**aturation, que modela a variedade de nuances referentes ao brilho em uma cor e **V**alue, que modela a mistura dessas cores com certas quantidades de preto e branco [8]. É possível observar a diferença entre os modelos RGB e HSV na Figura 18.

Para auxiliar na definição dos valores foi utilizado um algoritmo que permite

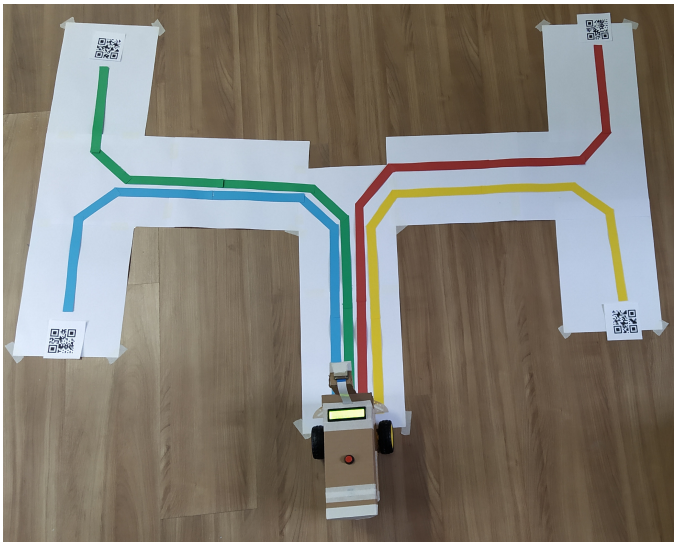

Figura 17: Pista completa com robô na farmácia.

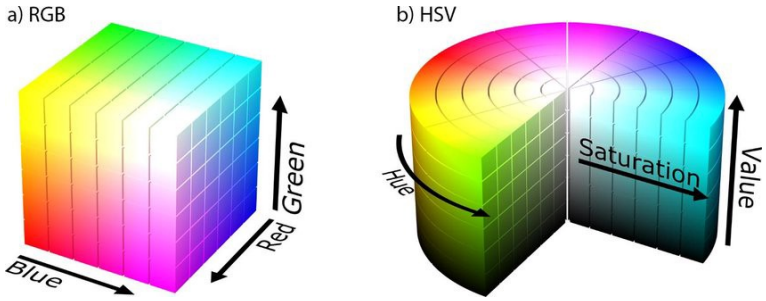

Figura 18: Representações utilizadas em RGB e HSV. Fonte: [9].

selecionar os intervalos dos canais Hue, Saturation e Value que serão aplicados na imagem. Esta ferramenta em questão permite visualizar a imagem original e a imagem com os canais aplicados, facilitando o processo de escolha dos valores. O código utilizado foi adaptado de [10] e os resultados obtidos a partir da Figura 19 podem ser observados na Figura 20. Após essa inspeção manual das imagens, os valores obtidos são codificados manualmente no algoritmo do robô.

É importante haver vários testes com várias faixas de valores de HSV para que o algoritmo não detecte cores erradas, como pode ser observado na Figura 21, em vez de detectar o que se espera, como mostrado na Figura 22.

5.3 Seguidor de linha

Este é o módulo que recebe estímulos do mundo externo e o processa a fim de auxiliar o robô na sua movimentação para alcançar seu objetivo.

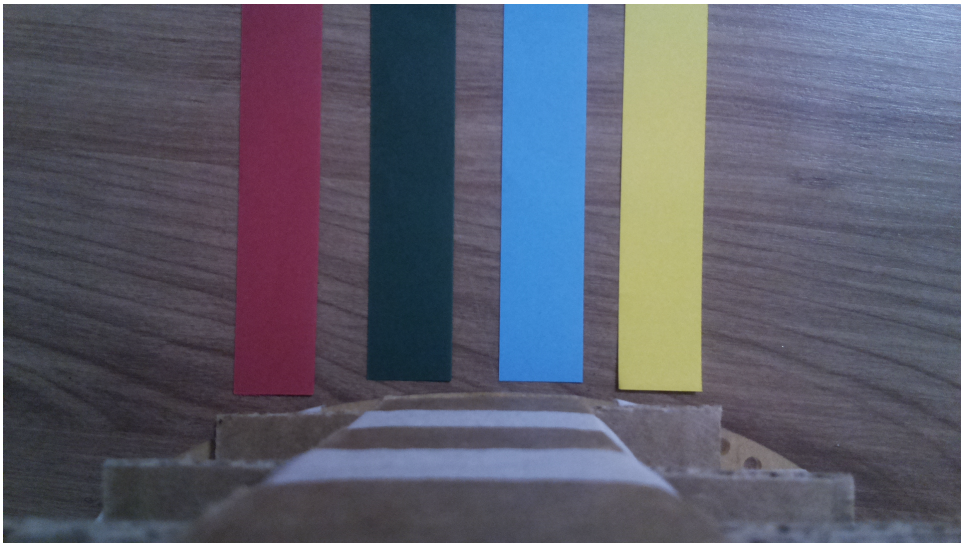

Figura 19: Visão do robô

(a) (b) (c) (d)

Figura 20: Imagens observadas das linhas com os filtros HSV aplicados. (a) Vermelho, (b) Verde, (c) Azul e (d) Amarelo.

Cada frame capturado pela câmera da Raspberry Pi passa por uma série de processamentos para que a imagem desejada seja gerada.

1. Inicialmente é feita uma cópia do frame nos canais HSV para construção de uma máscara. Máscaras são matrizes utilizadas para sobrepor e percorrer uma imagem original, fazendo operações pixel a pixel originando uma outra imagem. Por exemplo, se for aplicada uma máscara de dimensões 1x1 em uma imagem de dimensões 600x400 o programa faria 240000 iterações alterando a imagem pixel a pixel.

2. É criada a máscara a partir da imagem HSV colocando como parâmetros de entrada o intervalo HSV da cor desejada. Neste caso, a máscara é uma imagem binária da mesma dimensão que o frame. Ou seja, a máscara é uma imagem com valores "1"onde os píxels estão dentro do intervalo HSV, e "0"se não estiverem no intervalo.

3. É feita uma operação (Figura 23) "AND"do frame original com a máscara gerada (imagem binária). Assim, o resultado da operação pixel a pixel será

Figura 21: Intervalo HSV que detecta várias cores

Figura 22: Intervalo HSV que detecta apenas azul claro

outra imagem colorida apenas onde a máscara tem valor "1", ou seja, somente na linha de cor desejada, o restante dos pixels serão todos pretos, pois os valores na máscara são "0". A função em Python assume duas imagens de entrada para a operação, no caso deste projeto as imagens são as mesmas [11].

4. O frame resultante da operação anterior é um frame RGB. Este é convertido para uma imagem em níveis de cinza e aplicado a Convolução Gaussiana para suavizar as bordas e criar um contorno artificial. Convolução é um procedimento em que é criado uma matriz de linhas e colunas ímpares de "1's", pega-se o valor central dessa matriz e compara-se com cada pixel da imagem e também compara-se os valores das redondezas daquele pixel e por fim é feita uma média e o valor do pixel passa a ser este novo resultado [12], como observado na Figura 24.

5. É passada uma cópia do frame embaçado para uma função do OpenCV responsável por detectar os contornos da imagem. O retorno dessa função é um vetor com as coordenadas, então é aplicada uma lógica para detectar o centro do objeto, resultando na Figura 25 [13].

6. Nos passos finais do algoritmo o programa aciona os motores do robô dependendo de onde a linha estiver, ajustando o movimento do robô para andar em cima do lugar desejado. E se o robô, por acaso se perder e não detectar linha alguma (por exemplo, se ele chegar em uma curva muito rápido), ele dá ré até se encontrar novamente e poder seguir seu caminho.

$$\texttt{dst}(\mathrm{I}) = \texttt{src1}(\mathrm{I}) \wedge \texttt{src2}(\mathrm{I}) \quad \texttt{if } \texttt{mask}(\mathrm{I}) \neq 0$$

Figura 23: Funcionamento da operação "AND". "src1"e "src2"são a mesma imagem, o frame original

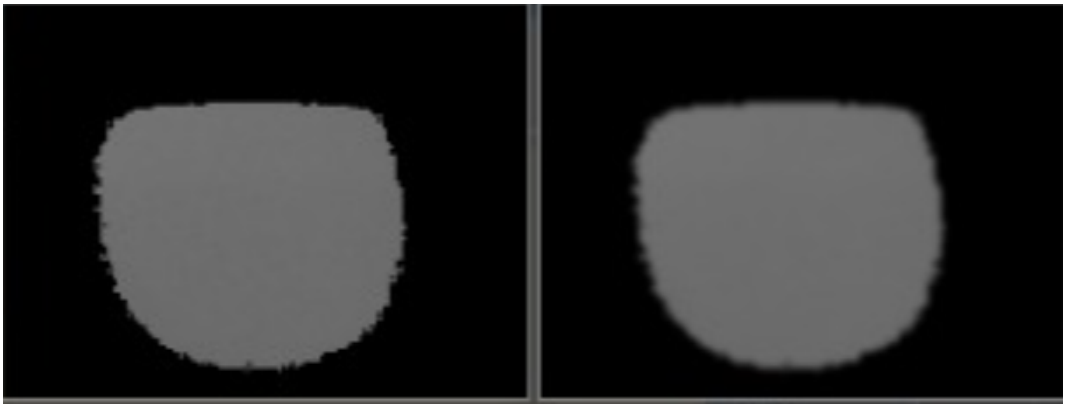

Figura 24: Frame antes e depois da Convolução Gaussiana

6 Sistema Embarcado

Para que o robô pudesse se comunicar com o servidor e também responder a estímulos do mundo externo foi necessário definir alguns requisitos para que um sistema embarcado fosse desenvolvido para atuar com a Raspberry Pi 3. Os requisitos definidos referentes ao sistema embarcado podem ser encontrados na Tabela 4.

Tabela 4: Requisitos do sistema embarcado.

Requisito	Descrição
FR09	O robô deve receber notificações do servidor e analisá-las.
FR10	O robô deve retornar à farmácia ao terminar uma entrega.
FR11	O robô deve ser capaz de ler QR Code para delegar sobre sua atual posição.
FR12	O robô deve notificar o servidor quando uma viagem é terminada.
FR12.1	A notificação deve conter qual ordem foi finalizada.
FR13	O robô deve aguardar o acionamento do botão para iniciar e finalizar uma viagem.

O algoritmo para que o robô possa fazer uma entrega completa tem as seguintes etapas:

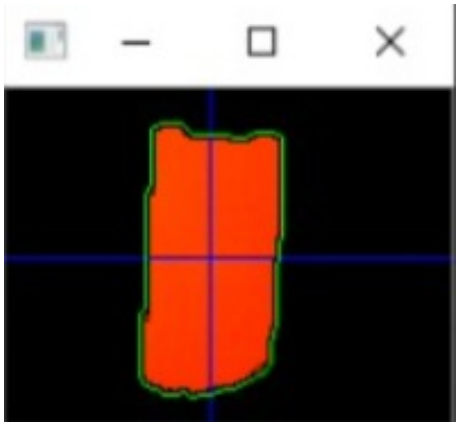

Figura 25: Imagem final processada

- ***getSentOrder()***: Recebe remédio a ser entregue;
- ***updt_ to_ delivering()***: Atualiza o status do pedido para "levando";
- ***lineFollowing()***: Segue a linha correspondente a cor da ala a ser levado o remédio;
- ***qr_ code_ found()***: Encontra o QR code da ala
- ***wait()***: Para e rotaciona 180°;
- ***button_ pressed()***: Espera o enfermeiro pressionar o botão;
- ***updt_ to_ delivered()***: Atualiza o status do pedido para "entregue";
- ***lineFollowing()***: Segue a linha de volta à farmácia;e
- ***qrCodeFound()***: Encontra o QR code da farmácia.

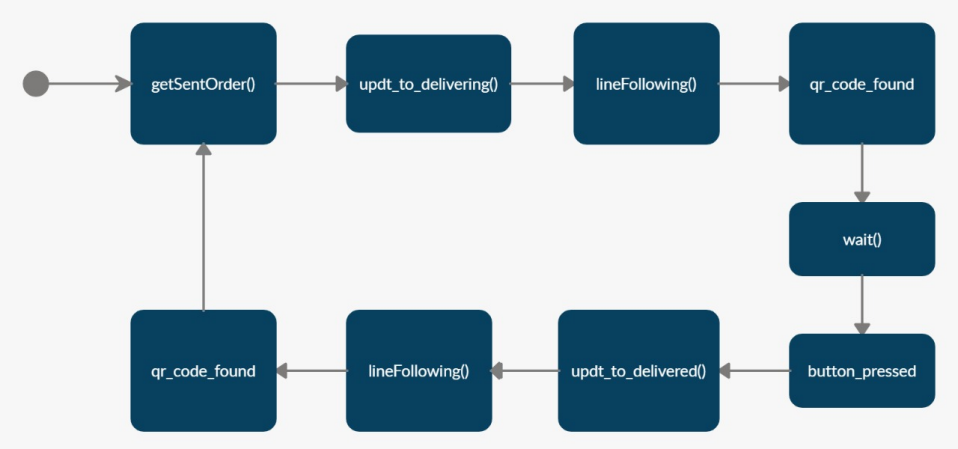

Figura 26: Fluxograma do funcionamento do sistema embarcado

7 Comunicação

Para que possa receber e atualizar a lista de pedidos a serem entregues, o robô faz requisições HTTP, conectando-se à rede local sem a necessidade de um dongle Wifi, pois o Raspberry Pi 3 já possui suporte nativo para se conectar a redes sem fios. O robô então, no estado em que está esperando por um novo pedido para ser entregue, faz um polling em um end-point da API do sistema a cada um segundo.

Após recebida a informação e tendo a entrega feita, o robô envia novamente outra requisição HTTP ao servidor, mas dessa vez em outro endpoint destinado a atualizar o status da entrega no sistema e alterar a quantidade de medicamentos consumidos.

8 Possibilidades de Expansão

No estágio atual de desenvolvimento do projeto, o robô faz apenas uma entrega de cada vez, o que significa que se duas alas precisarem de remédios na mesma hora, o robô precisa fazer uma entrega à uma delas, retornar à farmácia, coletar os novos medicamentos e então realizar a segunda entrega. O escopo pode ser expandido de modo que fosse implementado um sistema de mapa mental em que o robô soubesse onde ficam as alas, e, ao sair de uma, ele já se localiza e vai direto até a segunda ala que necessita de medicamentos.

O mapa mental é um conjunto de rotas que o robô teria consciência sobre. Por exemplo, se ele utilizasse a linha amarela para ir até uma ala (A) e depois precisasse seguir a linha azul para ir até outra ala (B), mas, na saída da ala A não enxergasse nenhuma linha azul, ele usaria o mapa mental para:

1. Realizar um backtrack até encontrar o ponto de divergência da linha azul com a amarela e, a partir daí, seguir a azul.

2. Seguir uma linha de outra cor para encontrar um ponto de convergência com a linha azul e, a partir daí, seguir a azul.

A expansão do projeto também passa por adotar os requisitos que foram definidos antes da pandemia, que incluiam a adição de três sensores ultrassônicos para que o robô possa desviar de obstáculos, o que seria corriqueiro em um corredor de hospital.

9 Conclusão

O projeto atendeu as expectativas, dadas as circunstâncias da atual conjuntura.

A parte de Software é robusta e segura, contando com serviços de autenticação e restrição de serviços dependendo do nível de acesso para médico, enfermeiro e farmacêutico. Enquanto que o Hardware é autônomo o suficiente para

entregar o medicamento até a ala específica no tempo determinado, permitindo que apenas o enfermeiro da respectiva ala consiga abrir o compartimento para retirar os medicamentos.

Antes da execução, a quantidade de horas estimada para a realização do projeto foi de 272 horas. Após concluir todas as etapas do desenvolvimento, somou-se 315 horas considerando o tempo gasto por todos os integrantes, 15% a mais do planejado. Os gastos estimados inicialmente eram de R$845,20, porém foram gastos R$785,20, pois foram usadas apenas quatro baterias para alimentação dos motores DC. A sistuação de pandemia fez com que houvesse um replanejamento das atividades, além da atualização do plano de riscos e do orçamento. O tempo reduzido para o desenvolvimento acarretou em um novo tipo de gerenciamento de projeto e acompanhamento de atividades. Julga-se que, tendo em vista a proposta da matéria de Oficina de Integração 3, os membros do grupo puderam desenvolver habilidades e competências compatíveis com o que o presente mercado de trabalho valoriza. Além disso, foi necessário ter contato com diversas ferramentas e técnicas que antes não eram familiares a todos os membros da equipe, como por exemplo o Azure DevOps para gerenciamento de projetos e técnicas Kanban e Scrum para desenvolvimento de tarefas individuais.

Referências

[1] ISMP España. Recomendaciones para la prevención de errores de medicación, boletín 41. *Instituto para el Uso Seguro de los Medicamentos*, December 2015.

[2] ISMP Canada Safety Bulletin. Aggregate analysis of dose omission incidents reported as causing harm. *The Institute for Safe Medication Practices Canada*, March 2013.

[3] API Restful conceito princípios e como criar. `https://www.hostgator.com.br/blog/api-restful/`.

[4] Qual a diferença entre base de dados relacional e não relacional? `https://debugeverything.com/diferenca-base-de-dados-relacional-e-nao-relacional/`.

[5] How To Draw NoSql Data Model Diagram? `https://www.techighness.com/post/how-to-draw-no-sql-data-model-diagram/`.

[6] Equivalent of ERD for MongoDB? `https://stackoverflow.com/questions/6010408/equivalent-of-erd-for-mongodb`.

[7] How React works under the hood. `https://www.freecodecamp.org/news/react-under-the-hood/`.

[8] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing, Page 429*. Pearson, 2007.

[9] Vencislav Popov, Markus Ostarek, and Caitlin Tenison. Inferential pitfalls in decoding neural representations. *NeuroImage*, page 10, October 2017.

[10] opencv_python_object_detection.py. `https://gist.github.com/pknowledge/aa1469b7ba8cd652adb652d4359ef4f0`.

[11] bitwise_and Operations on Arrays. `https://docs.opencv.org/2.4.13.2/modules/core/doc/operations_on_arrays.html?highlight=bitwise#bitwise-and`.

[12] Image Blurring Image Smoothing. `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html#image-blurring-image-smoothing`.

[13] Find the Center of a Blob (Centroid) using OpenCV (C++/Python). `https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/`.