



UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

PROJETO INTEGRADOR - DAS 5104

Entrega Final

Acadêmicos:

Daniel Regner
Igor Althoff Vidal
Ivan Doria

Matrícula:

15204839
13104313
13204824

Sumário

1	Introdução	3
1.1	Seashell	3
1.1.1	A Empresa	3
1.1.2	O produto	5
2	Requisitos	6
2.0.1	Requisitos funcionais	6
2.0.2	Requisitos não-funcionais	7
2.1	Plataforma utilizada	7
2.2	Cenário Escolhido	8
2.3	Casos de Uso do Sistema	8
2.4	Identificação de funcionalidades	9
2.5	Diagrama em árvore	9
2.6	Diagrama sequencial	11
2.7	Modelo conceitual	12
3	Estrutura Física	14
3.1	Análise Estrutural	15
3.2	Círculo Elétrico	15
3.3	Instalação Sensores	18
4	Movimentação	21
4.1	Cinemática Direta	21
4.2	Máquina de Estados	23
5	Simulação V-REP	25
5.1	Modelagem do robô	25
5.2	Software da simulação	26
6	Desenvolvimento do Código para o ESP-32	27
6.1	Estrutura do código	27
6.2	Comunicação entre Estação de Operação e a plataforma Robótica	29
6.2.1	Interface de comunicação Wi-Fi via WebSockets	29
6.2.2	Dados Transmitidos	29
6.3	Implementação da Máquina de Estados	31
6.4	Execução paralela via Threads	32
6.5	Interface de Usuário (UI)	33
6.6	Updates Over-The-Air	35
7	Implementação do movimento	37

8 Resultados obtidos	40
9 Conclusão	42

1 Introdução

Este documento é um registro de funcionalidades desenvolvidas sob o projeto do robô quadrupede a ser controlado de forma manual, definindo o sentido da trajetória de cada passo, e de forma automática respondendo através de medições de sensores qual será o próximo movimento a ser efetuado. Encontram-se aqui, como solicitado, as atividades desenvolvidas pela equipe desde o começo do projeto e já registradas, porém com eventuais modificações decorrentes de correções implementadas ao longo do projeto.

Após a discussão com a empresa e implementação da nova estrutura utilizando servo motores de maior torque para suportar cargas maiores e consequentemente ter uma robustez maior para futuras implementações sobre a estrutura. A sequência deste documento retrata a implementação física da estrutura com a parte de eletrônica embarcada, comunicação com o cliente para acessar suas funcionalidades e medições dos sensores acoplados e a implementação do diagrama de classe descrito em relatórios anteriores para efetuar a movimentação através de uma máquina de estados.

Dentre os principais problemas encontrados neste projeto, a implementação da movimentação foi um dos maiores desafios enfrentados para implementação em vista pelo curto período à ser implementada. Desta forma o problema foi simplificado de uma formulação matemática da cinemática inversa da estrutura para estados e posições previamente definidos para efetuar as sequências de movimentos da caminhada conforme descrito nos relatórios anteriores.

1.1 Seashell

1.1.1 A Empresa

É uma empresa formada por ex-alunos do curso de Engenharia de Controle e Automação da UFSC e que foi selecionada pelo programa Sinapse da Inovação. Tem como objetivo democratizar o acesso a tecnologias embarcadas de ponta, facilitando o desenvolvimento de aplicações complexas, e contribuindo para o surgimento de produtos e negócios inovadores.

Atualmente estão incubados no Celta parque em Florianópolis.



Figura 1: Logotipo da empresa Seashell

1.1.2 O produto

É uma plataforma de DevOps para software embarcado baseada em Linux e containers Docker. A plataforma possui como principais features:

- Distribuição Linux “enxuto” (< 200MB);
- Runtime para execução embarcada de containers Docker;
- Compilação cruzada out-of-the-box com suporte a múltiplas arquiteturas de hardware;
- APIs para acesso a periféricos, comunicação, etc;
- Atualização/configuração remota (OTA);
- Suporte nativo a aplicações críticas de tempo-real;
- Plataforma web para gerenciamento dos dispositivos;

O principal produto consiste em um framework de desenvolvimento de software embarcado que inclui infraestrutura e ferramentas capazes de facilitar o desenvolvimento de aplicações para sistemas envolvendo aeronaves não-tripuladas, máquinas agrícolas, equipamentos industriais, automóveis, sistemas robóticos, dispositivos de IoT, entre outros.



Figura 2: Representação do produto da empresa Seashell, demonstrando todas as etapas da aplicação de um sistema embarcado, do baixo ao alto nível

2 Requisitos

Ao início do projeto foram definidos alguns requisitos a serem alcançados para conclusão do mesmo. Os requisitos previamente definidos foram:

- Definir um cenário de demonstração da plataforma utilizando o Robô Quadrupede juntamente com os sensores fornecidos pela empresa.
- Implementar em Linux o cenário de demonstração do Robô.
- Documentar detalhadamente o processo desenvolvido, incluindo detalhes da montagem física, cálculos matemáticos envolvidos e código implementado, a ser entregue como um artigo informativo.
- Desenvolver um manual para facilitar a utilização da plataforma robótica desenvolvida em futuras demonstrações.
- Realizar testes com a plataforma para validar sua funcionalidade.

2.0.1 Requisitos funcionais

ID	Descrição
R.F. 1	Controlar o movimento do robô
R.F. 2	Manter distância de qualquer objeto/pessoa que esteja na sua proximidade
R.F. 3	Mensurar variáveis relevantes para a compreensão espacial do robô
R.F. 4	Apresentar os dados mensurados para um usuário
R.F. 5	Disponibilizar uma interface para que o usuário possa controlar o robô

2.0.2 Requisitos não-funcionais

ID	Descrição
R.N.F 1	Documentar cada etapa de desenvolvimento
R.N.F 2	Utilizar comunicação wi-fi para realizar a interface com o usuário
R.N.F 3	Codificar a cinemática inversa do robô
R.N.F 4	Utilizar dos materiais oferecidos pela empresa Seashell
R.N.F 4.1	Utilizar sensores ultrasônicos
R.N.F 4.2	Utilizar estrutura impressa em impressora 3D com servo motores
R.N.F 4.3	Utilizar Raspberry Pi
R.N.F 4.4	Utilizar IMU
R.N.F 4.5	Utilizar ESP32
R.N.F 5 (opcional)	Utilizar plataforma da Seashell para fazer a integração com o robô

2.1 Plataforma utilizada

Em um primeiro momento, a plataforma robótica utilizada foi a um robô quadrupede desenvolvido por Emanuele Santellani, disponibilizada como projeto aberto no Thingiverse (<https://www.thingiverse.com/thing:1677703/>) e ilustrada na Figura 3.

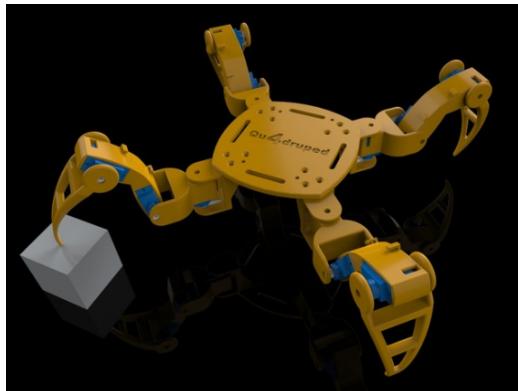


Figura 3: Robô quadrupede utilizado.

Entretanto, ao longo do projeto, foi necessária a mudança desta para uma plataforma maior e mais robusta, tendo em vista que o robô irá carregar outros componentes e com a escolha inicial ele já operaria em seu limite, mesmo sem carga. A nova estrutura é apresentada no próximo capítulo e ilustrada na Figura 9.

2.2 Cenário Escolhido

O cenário escolhido pelo grupo é fazer com que o robô se afaste dos objetos posicionados ao seu redor utilizando como referência as medidas de distância obtidas por sensores ultrassônicos. O controle em alto nível irá calcular as distâncias nas 4 direções do robô e posicioná-lo de modo a respeitar uma distância mínima dos objetos a seu redor, ou o mais afastado possível caso não seja possível respeitar a distância mínima. Este cenário é ilustrado na Figura 4.

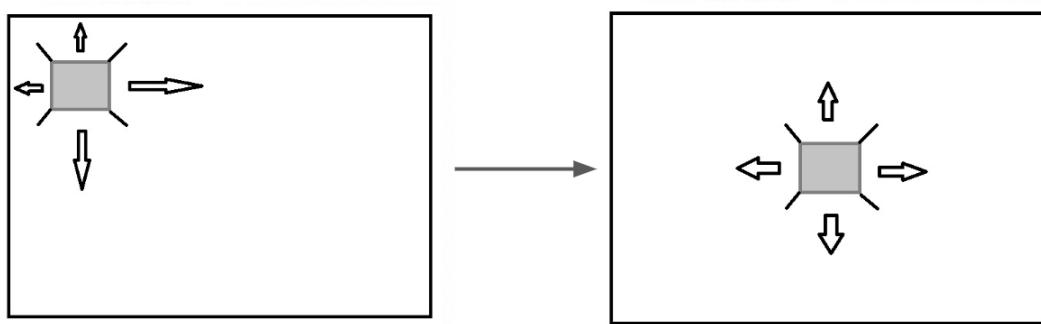


Figura 4: Cenário escolhido pelo grupo.

2.3 Casos de Uso do Sistema

Com base nas especificações do projeto descritas acima, foi desenvolvido um Caso de uso do sistema apresentado na figura 5.

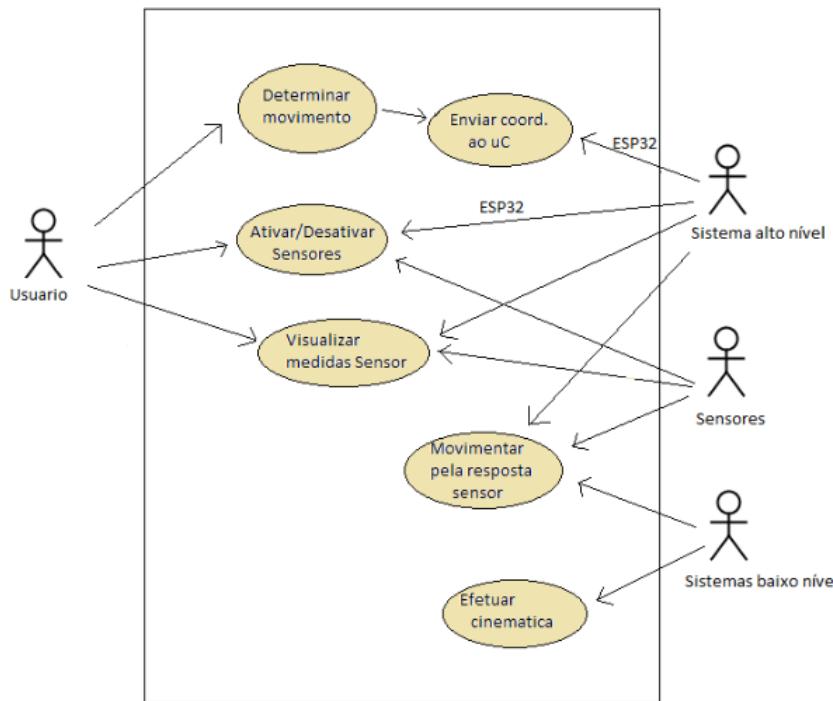


Figura 5: Casos de Uso do Sistema

2.4 Identificação de funcionalidades

A aplicação requer que o robô possa se movimentar através de uma referência pelo usuário ou que o robô tenha autonomia para decidir seus movimentos tendo em vista respeitar uma regra de segurança informada também pelo usuário. Para que isso seja possível, é necessário que o robô seja capaz de se movimentar e de reconhecer o ambiente ao seu redor. Além disso, é necessário que exista uma interface para permitir a comunicação e a operação remota do protótipo.

A partir disso, podemos extraír três funcionalidades principais para o sistema. Primeiramente, temos a movimentação do robô, seguida pelo reconhecimento de ambientes e por último a interface de comunicação.

2.5 Diagrama em árvore

Para elaboração do diagrama em árvore, é necessário elistar as sub-funcionalidades do sistema. Começando pela função de Movimentação, temos que o robô irá precisar desenvolver os cálculos de cinemática inversa para posicionar corretamente cada uma

de suas juntas, de modo a desenvolver seu movimento. Também é necessário que o mesmo execute uma máquina de estados, para correta coordenação do movimento das pernas.

Para a funcionalidade de reconhecimento de ambientes, é necessário que o sistema realize a leitura dos sensores ultrassônicos acoplados a ele. Entretanto, para a utilização destes dados, também é necessário realizar os cálculos para obter as distâncias a partir das leituras de cada sensor.

A interface do sistema irá possibilitar a seleção de um dos modos de operação do robô, entre automático e manual, e também irá possibilitar a passagem de referência. Além disso, a interface será responsável por realizar a comunicação entre a lógica de alto-nível, implementada na Raspberry Pi, com a lógica do microcontrolador ESP-32, de baixo nível.

Sendo assim, obtemos a seguinte lista de funcionalidades e sub-funcionalidades:

- *F.1: Movimentação*
 - *S.F.1.1: Cálculo da Cinemática Inversa.*
 - *S.F.1.2: Execução da Máquina de Estados.*
- *F.2: Reconhecimento de Ambientes*
 - *S.F.2.1: Leitura dos Sensores.*
 - *S.F.2.2: Cálculo das Distâncias.*
- *F.3: Interface*
 - *S.F.3.1: Passagem de Referências.*
 - *S.F.3.2: Transição entre modos de operação.*
 - *S.F.3.3: Comunicação entre Níveis.*

A partir das funcionalidades apresentadas acima, chegamos no diagrama de árvore apresentado na Figura 6.

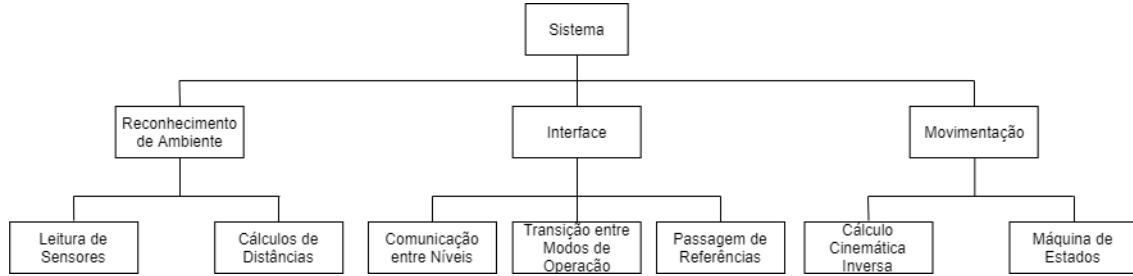


Figura 6: Diagrama em Árvore com a apresentação das funcionalidades e sub-funcionalidades do sistema

2.6 Diagrama sequencial

Através das especificações do sistema, foi desenvolvido o diagrama sequencial descrito na figura 7, representando a sequência de processos (mais especificamente, de mensagens passadas entre objetos) sobre o sistema. O sistema é constituído de uma interface a ser implementada em um módulo Raspberry Pi, o qual tomará ações sobre o controle automático, efetuando o controle através dos valores lidos sobre os sensores, implementados na camada de baixo nível, ESP 32.

Inicializando o sistema pelo usuário através da interface, o sistema realiza um autoteste para checar funcionamento de sensores e movimentar os servos de cada perna para uma posição inicial. Efetuada a inicialização do sistema, o usuário deve escolher entre as opções de movimentação manual ou automática baseada na leitura de sensores. O Diagrama de Sequências obtido é ilustrado na Figura 7.

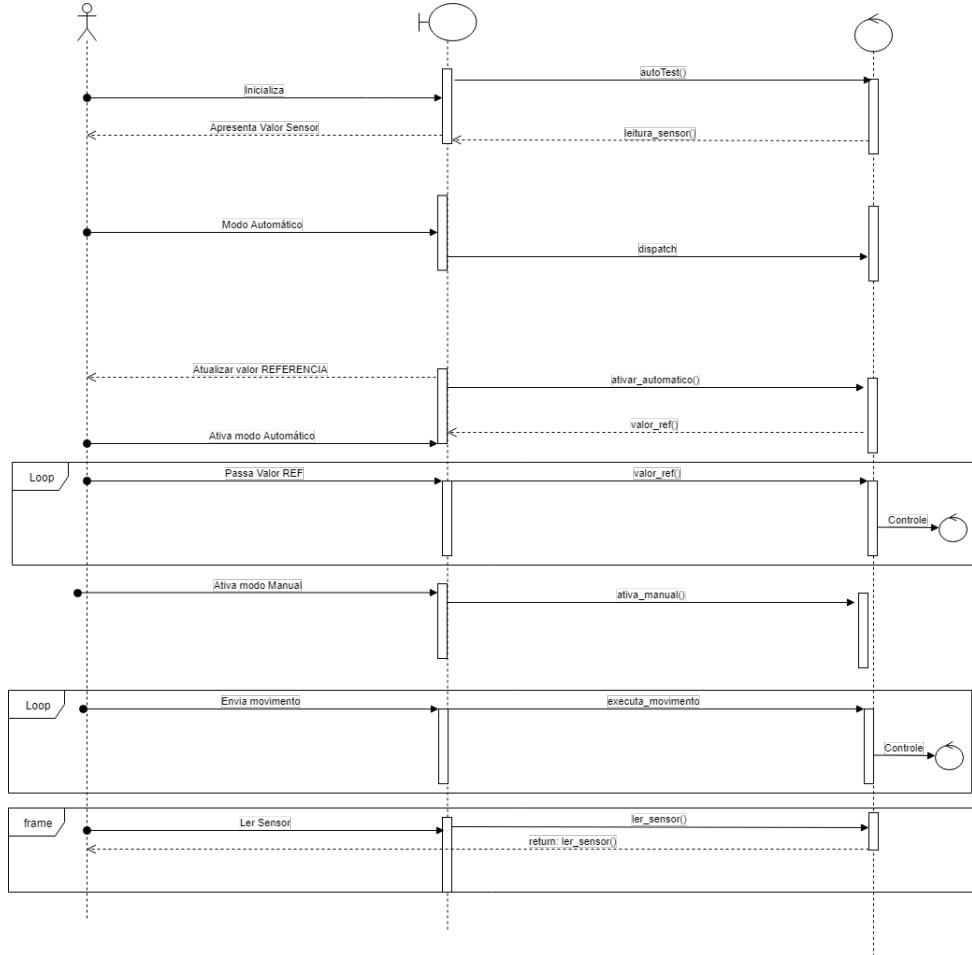


Figura 7: Diagrama sequencial

2.7 Modelo conceitual

O Modelo Conceitual é um conjunto de suposições baseadas no mundo real que indicarão como os componentes do projeto estão relacionados. De acordo com as funcionalidades do nosso projeto foi elaborado um modelo conceitual, com as principais classes do sistema, como apresentado abaixo na figura 8.

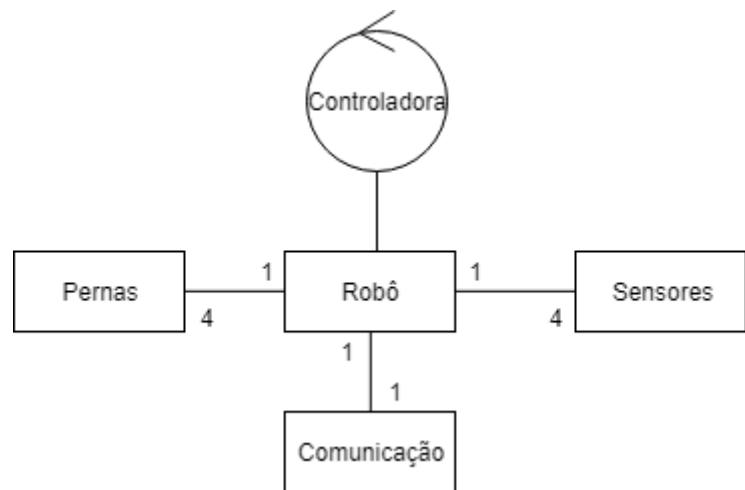


Figura 8: Modelo conceitual

3 Estrutura Física

Através da solução para o problema estrutural ocasionado pela falta de torque dos Servomotores 9g de 1.1 Kgfcm, foi utilizado o projeto **”A 3D Printed Quadruped Robot”**, utilizando Servomotores MG955 de 11 Kgfcm. Com a estrutura montada, ilustrada na Figura 9, foi dada a sequência de instalação dos sensores para realização das leituras e consequentemente permitindo a atuação sobre os valores lidos.



Figura 9: Estrutura nova Robô Quadrupede

3.1 Análise Estrutural

Para comprovação de que a nova estrutura teria condições de efetuar o movimento e sustentação da estrutura foi desenvolvido um cálculo estrutural sobre os momentos resultantes em cada perna, sendo a força peso de cada componente medida ao longo da montagem e apresentada na tabela 1. Utilizando da imagem 10 para calcular o momento resultante sobre os motores *A* e *B*.

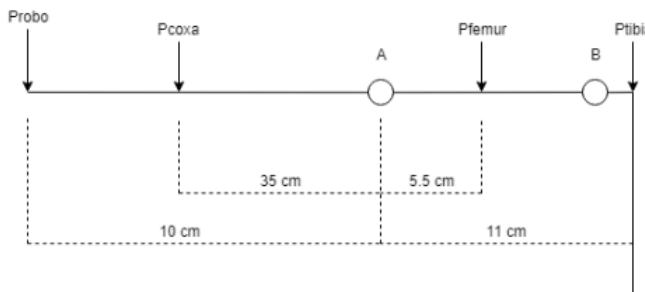


Figura 10: Diagrama Estrutural para cada perna

Efetuado os cálculos para os momentos resultantes sobre os motores representados na imagem, resultando em um momento aplicado sobre os motores de aproximadamente 6 e 8 kgfcm, inferior ao torque nominal de um Servo-motor MG955 com tensão nominal de 5 V.

Componente	Peso
Base	230 g
Coxa	130 g
Fêmur	180 g
Tibia	80 g
Total perna	390 g
Total robo	1790 g

Tabela 1: Tabela de Pesos da Estrutura

3.2 Circuito Elétrico

De modo a atender os requisitos do projeto quanto a utilização de componentes eletrônicos, listados na tabela 2 e suas ligações, foi desenvolvido um circuito impresso através do Software de prototipagem EAGLE. Na figura 12 pode ser visualizado o esquemático de ligação do microcontrolador aos sensores ultrassônicos e IMU, assim como a ligação ao módulo através de comunicação i2c.

Componente	Quantidade
Microcontrolador ESP32	1
Módulo PWM 16 Canais	1
Servo motor MG955	12
Sensor Ultrassônico HC-SR04	4
IMU MPU6050	1

Tabela 2: Tabela de Componentes Eletrônicos

Para a utilização dos sensores ultrassônicos disponíveis no mercado, modelo HC-SR04, juntamente com a placa ESP-32, necessitamos utilizar um divisor resistivo para reduzir a tensão proveniente do pino de retorno *ECHO* do sensor devido as portas lógicas do microcontrolador utilizarem como nível lógico 3.3V e os sensores necessitarem de uma tensão de alimentação de 5V. A imagem 11 descreve o esquemático utilizando juntamente com os cálculos para diferença entre resistores.

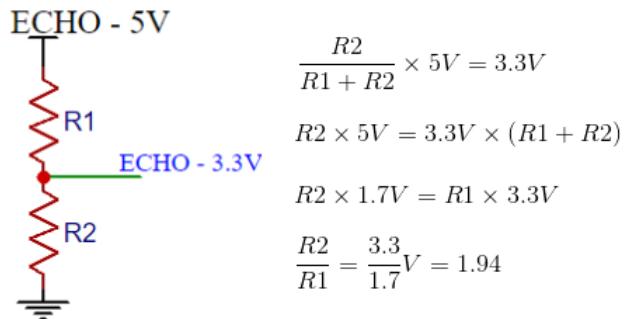


Figura 11: Divisor resistivo para leitura sensor Ultrassônico

Com a proporção entre R1 e R2 definida, utilizamos valores de resistores para R1 e R2 de $1k\Omega$ e $2k\Omega$ respectivamente utilizando três resistores em série de $1k\Omega$.

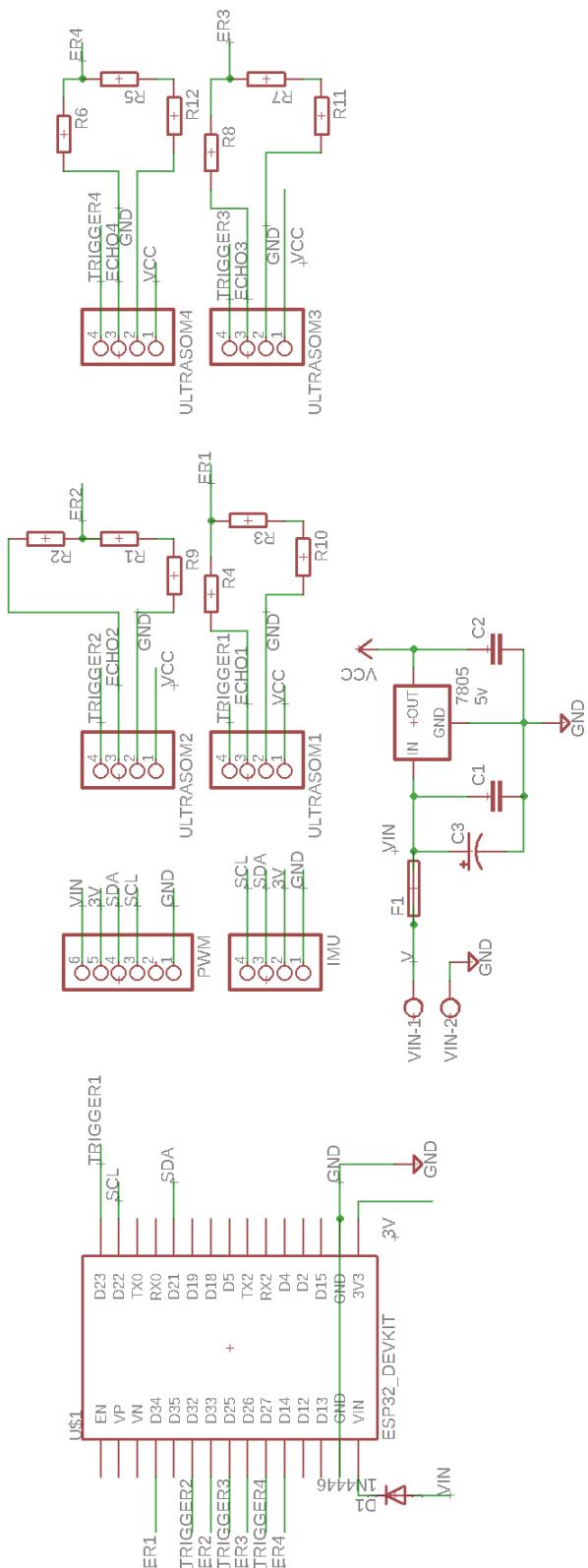


Figura 12: Esquemático circuito elétrico desenvolvido EAGLE
17

Após validado o esquemático apresentado na figura 12 foi desenvolvido para prototipagem o esquemático da PCB para impressão e implementação. A Figura 13 descreve a placa impressa desenvolvida para conexão do mico controlador aos sensores e módulos.

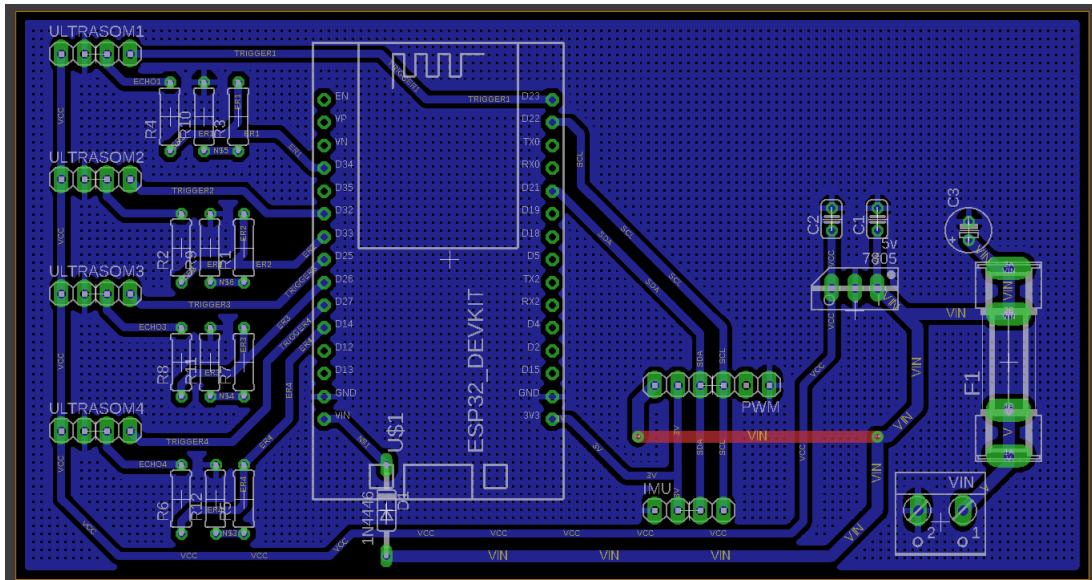


Figura 13: Prototipagem desenvolvido EAGLE

3.3 Instalação Sensores

Efetuada a prototipagem da placa impressa para conexão dos sensores ao micro-controlador, basta agora determinar os posicionamentos dos sensores ultrassônicos e a unidade de medição inercial (IMU) para efetuarem leituras coerentes da estrutura. Para a fixação dos sensores ultrassônicos, foi desenvolvido uma estrutura fixada aos parafusos da carcaça superior de forma a não danificá-la e ser de fácil remoção caso tenham interesse de alterar o sistema de medição futuramente conforme figura 14.



Figura 14: Suporte modelado via INVENTOR para Sensores Ultrassônicos



Figura 15: Estrutura com suporte Sensores Ultrassônicos



Figura 16: Fixação módulo IMU



Figura 17: Estrutura Completa

4 Movimentação

Para desenvolver o movimento do robô quadrupede, através de uma discussão entre os integrantes e a empresa *Seashell*, foi determinado em primeiro momento a utilização da formulação matemática para desenvolver a cinemática inversa através da inversão da matriz de Denavit-Hartemberg. O desenvolvimento matemático pode ser visualizado nas subseções seguintes.

Ressalta-se aqui que, devido ao curto período de tempo disponível para a implementação do projeto, a cinemática direta e inversa foi realizada porém não implementada, com a movimentação sendo implementada de maneira ad-hoc como mostrado na próxima seção.

4.1 Cinemática Direta

O desenvolvimento da cinemática direta tem como base os parâmetros de Denavit-Hartemberg. São quatro parâmetros associados a uma convenção para fixar sistemas de referência aos elos de uma cadeia cinemática espacial, ou manipulador robótico. Nesta convenção, sistemas de coordenadas são fixados a articulações entre dois elos, de forma que uma transformação seja associada à articulação $[Z]$, e a segunda transformação seja associada ao elo $[X]$. As transformações de coordenadas ao longo de um robô em série consistindo de n elos resulta nas equações cinemáticas do robô. Estes parâmetros são definidos como descrito abaixo, a Figura 18 representa os parâmetros a serem definidos.

- d : distância ao longo do z_{n-1} anterior até a normal comum do z_n .
- θ : ângulo de rotação do z_{n-1} anterior, iniciando de x_{n-1} até x_n
- α : ângulo de rotação entre z_{n-1} e z_n
- a : distância entre origem $n-1$ e n .

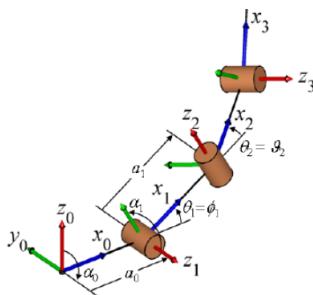


Figura 18: Parâmetros de Denavit-Hartemberg.

Após determinado os parâmetros, foi desenvolvido uma tabela de valores sob o manipulador de três graus de liberdade de cada perna, apresentada na Tabela 3.

	a	θ	α	d
1	$\frac{\pi}{2}$	θ_1	a1	0
2	0	θ_2	a2	0
3	0	θ_3	a3	0

Tabela 3: Parâmetros de Denavit-Hartenberg.

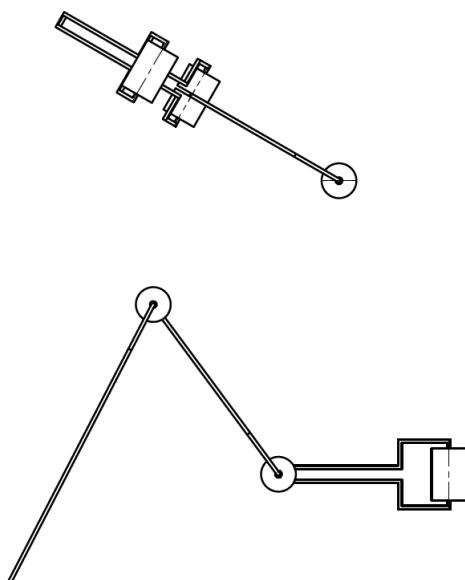


Figura 19: Vistas superior e lateral da perna.

4.2 Máquina de Estados

Para simplificar a movimentação do robô e dar continuidade ao desenvolvimento do projeto, concluímos que para curto prazo seria mais interessante desenvolver a cinemática do sistema com base em uma máquina de estados, de maneira ad-hoc.

Dado que, para sustentação do robô, é necessário que não haja deslizamento entre as pernas e a superfície de trabalho, o robô deve levantar cada perna que deva ser movimentada. Entretanto, levantar mais de uma perna ao mesmo tempo pode resultar na perda da estabilidade do sistema, sendo assim recomendado para robôs de mesma configuração, que somente uma perna seja removida da superfície por vez.

De modo a garantir a estabilidade do robô mesmo quando uma de suas pernas não está apoiada na superfície de trabalho, é necessário que em todos os instantes o Centro de Gravidade do sistema esteja dentro do triângulo formado pelos pontos de sustentação do mesmo. Esta situação é ilustrada na Figura 20

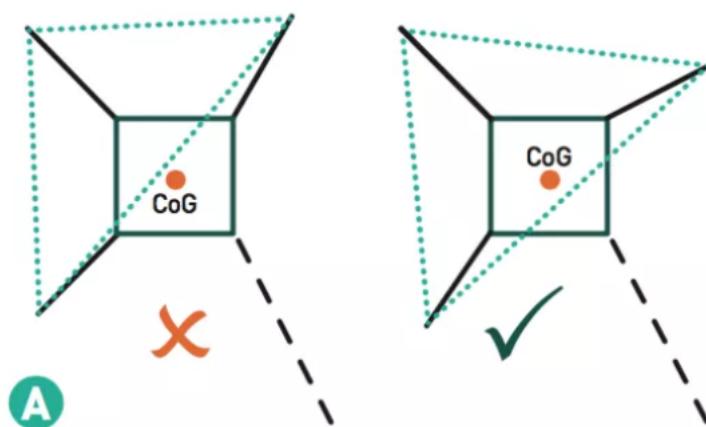


Figura 20: Detalhe do Centro de Gravidade dentro do Robô dentro do triângulo formado pelos pontos seus pontos de sustentação.

Fonte: <https://makezine.com/2016/11/22/robot-quadruped-arduino-program/>

Tendo definidas as condições que devem ser respeitadas para correta movimentação do robô, definiu-se a sequência de movimentação a ser implementada para o mesmo. Esta implementação foi escolhida dada a sua simplicidade de implementação, além de ter uma maior garantia de funcionamento, apesar de ser menos eficiente do que métodos como a cinemática inversa. Isto acontece pois na primeira os movimentos são desenvolvidos em sequência, perna a perna, e na segunda todos os servos das pernas são ajustadas ao mesmo tempo.

A imagem 21 representa os passos a serem tomados para iniciar e desenvolver uma caminhada, sendo as pernas representadas por uma letra de A a D.

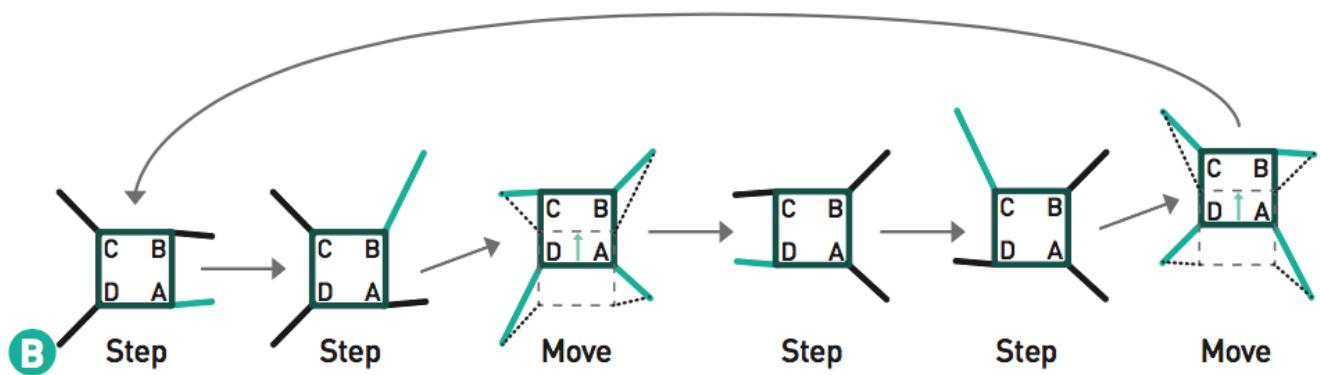


Figura 21: Sequência dos movimentos de perna realizados pelo robô de modo a se deslocar para frente.

Fonte: <https://makezine.com/2016/11/22/robot-quadruped-arduino-program/>

Abaixo, na Figura 22, ilustra-se a máquina de estados obtida.

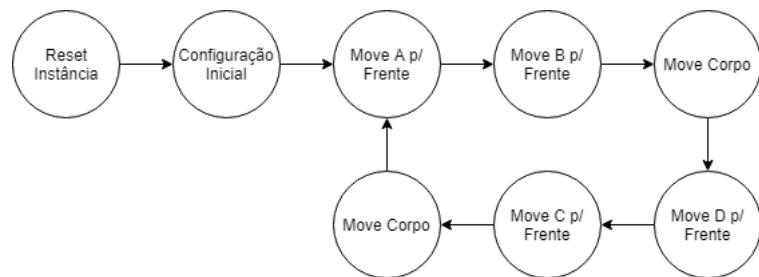


Figura 22: Máquina de Estados para andar

5 Simulação V-REP

A movimentação do robô é uma das questões centrais deste projeto. A disposição das pernas para se alterar o centro de massa e consequentemente o andar do robô não é algo simples. Existe portanto a necessidade de validar a movimentação que já foi descrita teoricamente em relatórios passados. Para tal é necessário simular o robô com o auxílio de ferramentas computacionais. A partir de uma pesquisa encontrou-se dois possíveis simuladores:

- Gazebo;
- V-rep;

O Gazebo é um software de simulação Open-Source utilizado por diversos grupos de pesquisas ao redor do mundo, porém ele somente é acessível em ambiente Linux, além disso a modelagem de novos robôs não é fácil. O V-rep, é gratuito para fins acadêmicos e possui diversas ferramentas que são úteis para o nosso projeto, alguns integrantes do grupo já possuíam familiaridade com essas ferramenta, dessa forma foi escolhido a plataforma do V-rep.

5.1 Modelagem do robô

O ambiente de simulação V-rep permite importar os arquivos diretamente do SolidWorks, dessa forma, o modelo físico do robô será o mesmo utilizado na simulação.

O V-rep otimiza a simulação permitindo a criação de uma Layer invertida com o intuito de tornar o cálculo do modelo mais simples. Essa Layer invertida é uma versão simplificada do robô, com peças sólidas mais simples, pois, quanto mais peças possui o robô mais complexo é o cálculo realizado pelo simulador. Conforme pode-se observar abaixo:

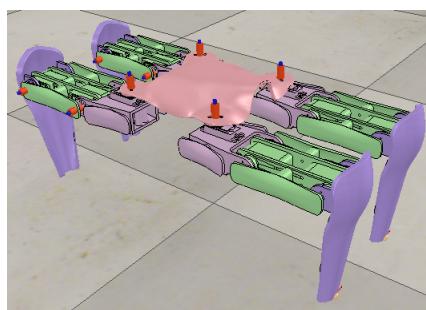


Figura 23: Modelo físico do robô

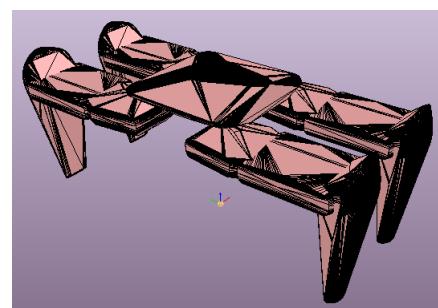


Figura 24: Modelo invertido do robô

5.2 Software da simulação

Com o ambiente de simulação V-rep a programação do robô é feito com linguagem Lua, porém é uma linguagem na qual os integrantes deste trabalho não possuem afinidade, dessa forma o V-rep permite a integração com outras linguagens de programação, tais como:

- Python
- C++
- Matlab
- Octave
- Java

Foi escolhido dessa forma a linguagem Python para o software do robô. Assim sendo utiliza-se no ambiente de simulação do código:

```
1 simRemoteApi.start(19999)
```

Abrindo assim uma porta para comunicação via Socket em 19999. E dentro do código em python escreve-se então:

```
1 vrep.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
```

Possibilitando então a interação entre o ambiente simulado e um código externo. No código disponível no repositório do projeto é então adicionado toda a sequencia de comandos para fazer o robô se mover.

A partir da janela de comando do python é possível inserir a direção e sentido do movimento do robô :

- W - frente;
- S - trás;
- A - gira esquerda;
- D - gira direita;

6 Desenvolvimento do Código para o ESP-32

6.1 Estrutura do código

De modo a garantir a boa organização do código e facilitar a continuação do desenvolvimento, o código para o ESP-32 foi programado utilizando C++ e aproveitando-se das vantagens da orientação a objetos. O diagrama de classes do código implementado é ilustrado na Figura 25.

A controladora, através da Interface de Usuário, recebe comandos para o robô. O robô então realiza as chamadas para as funções *setDefaultPose()* e *setAnotherPose()*. Estas funções foram criadas para testar o funcionamento do código e da máquina nesta arquitetura, além de validar a integração com os outros componentes que serão explicados nas próximas seções, e posteriormente serão substituídas pelo código que já possui a implementação da movimentação em um arquivo separado.

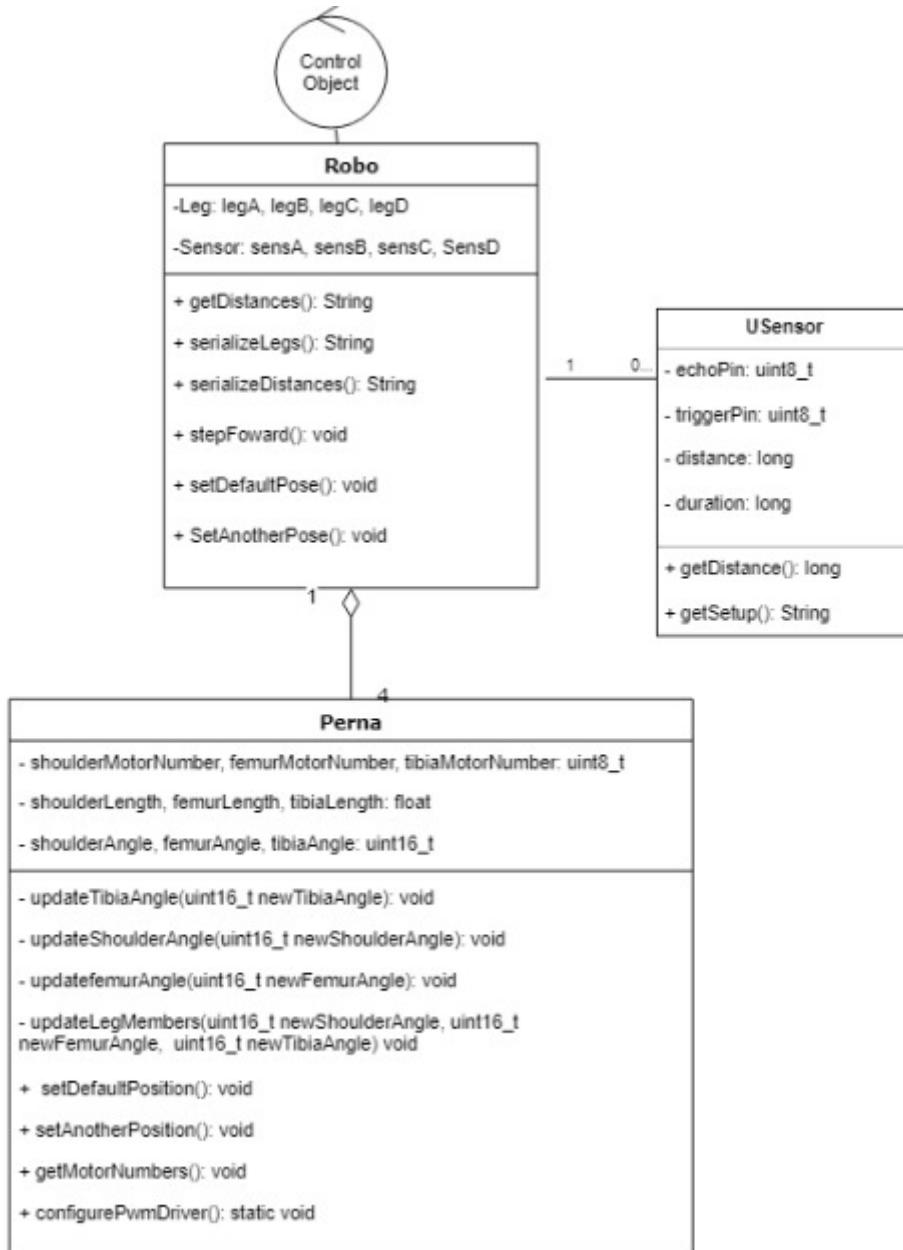


Figura 25: Diagrama de classes do código implementado, que possui 4 pernas e 4 sensores privados, além dos respectivos métodos para cada classe e seus parâmetros.

6.2 Comunicação entre Estação de Operação e a plataforma Robótica

6.2.1 Interface de comunicação Wi-Fi via WebSockets

De modo a permitir a operação remota da plataforma, mostrou-se necessária a implementação de uma interface de comunicação entre o microcontrolador ESP-32 e a estação de operação. Conforme já mencionado anteriormente, o microcontrolador ESP-32 foi escolhido pois este já possui um chip para comunicação *Wireless* com base nos padrões da norma IEEE 802.11.

Devido a prévia experiência do grupo com a utilização de Soquetes de Redes (popularmente conhecido por Sockets) na Raspberry Pi, optou-se por primeiramente tentar utilizar-se desta solução para resolver o problema de comunicação entre o ESP e a Raspberry Pi. Testes foram feitos com implementações dos protocolos TCP e UDP no ESP, porém sem sucesso satisfatório. Esta solução mostrou-se demorada devido a baixa documentação sobre a implementação dos Sockets no ESP-32 e o baixo suporte da comunidade, e por isso novas alternativas foram procuradas.

Após discussão do problema com a empresa Seashell, o grupo foi recomendado a implementar o protocolo WebSockets, dado a sua simplicidade e a sua grande utilização. Por se tratar de um protocolo mais documentado para o ESP, sua implementação foi feita de maneira rápida e com uma performance aceitável.

Ainda aproveitando-se da vasta documentação do uso dos WebSockets, foi criado um *script* em Python que constantemente envia mensagens para o ESP-32 através dos WebSockets e recebe a resposta do mesmo. Este *script* foi até o momento foi testado em um notebook com Linux e por isso não deve apresentar problemas ao ser utilizado em uma Raspberry Pi que também dispõe de Linux e a mesma versão do Python.

Através de uma conversa com a empresa, foi decidido que a interface de usuário seria implementada por eles, dado o maior conhecimento das ferramentas necessárias e também por facilitar uma eventual integração com o produto da empresa. Resultados sobre a interface de usuário são apresentados na subseção 6.5

6.2.2 Dados Transmitidos

Para garantir que o usuário tenha uma boa experiência ao operar a plataforma robótica, foi realizado um brainstorm, com os membros do grupo e da empresa, de modo a definir quais seriam os dados que deveriam ser disponibilizados para o usuário e quais informações este iria transmitir para o robô.

A informação mais importante que o robô deve mandar para o usuário é a leitura dos sensores ultrassônicos, para que o operador possa ter uma ideia do meio onde o robô se encontra e validar as decisões a tomadas quanto aos movimentos a serem desenvolvidos. Sendo assim, esta informação é útil na parte de desenvolvimento e aperfeiçoamento do produto e também no produto final.

Além disso, mostrou-se interessante disponibilizar ao usuário os dados relativos a posição de cada uma das juntas do robô. Assim, é possível recriar um modelo da plataforma na UI e permitir um acompanhamento do status do robô remotamente.

Por último, seria interessante também adicionar um dispositivo de medição por inércia (IMU), para permitir a visualização da orientação da plataforma nos 3 eixos, melhorando a experiência do usuário.

De modo a permitir que o robô desenvolva as tarefas desejadas pelo operador, chegou-se a conclusão da necessidade de enviar a plataforma o modo de operação desejado (Automático ou Manual), para definir se o robô deve tomar suas próprias decisões quanto ao movimento a ser executado ou se este deve aguardar a passagem de uma referência para se movimentar.

No caso da operação manual, foi definido que a passagem de referência será a direção para a qual o robô deve movimentar-se. Como o robô não possui um *header*, ou seja, qualquer lado pode ser sua frente, este pode desempenhar um padrão de movimentação *headerless*, mudando sempre sua definição de frente quando uma referência solicitando movimentação em outra direção é recebida.

Esta característica ainda facilita o desenvolvimento do código de movimentação, pois é necessário somente desenvolver a sequência de movimentos para cada perna de modo com que o robô ande para frente, restando somente alterar a ordem de movimentação de cada perna quando este tiver que se movimentar em outra direção, sem a necessidade de rotacionar o corpo do robô. Esta situação é melhor ilustrada na Figura 27.

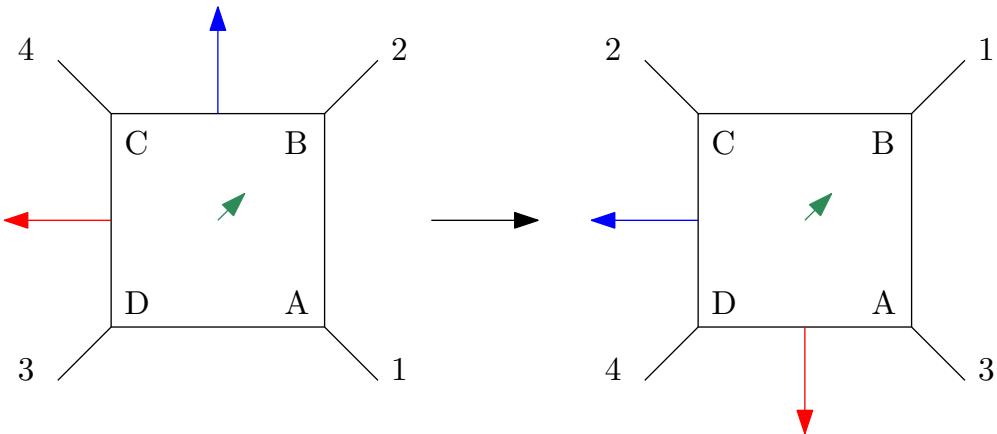


Figura 26: Exemplo da mudança de orientação do robô. Com o robô se movimentando para a frente, temos a ordem de movimentação A,B,D,C. Quando solicita-se movimento para a esquerda, teremos a ordem B,C,D,A.

Foi requisitado pela empresa que os dados trocados entre a estação de operação e a plataforma robótica fossem formatados no padrão *JSON*, facilitando sua legibilidade por parte dos usuários, além de possuir um bom suporte da comunidade e possuir vários métodos disponíveis para o desenvolvimento.

6.3 Implementação da Máquina de Estados

Para a implementação da sequência de movimentação do robô, foi desenvolvida no ESP-32 a lógica da movimentação para a frente. Como ilustrado na Figura 22, o robô possui 8 estados em que ele irá desenvolver algum tipo de movimentação. Uma função *fsm.run()* foi implementada para executar as mudanças de estados, e de modo a permitir uma melhor organização do código, optou-se pela utilização da biblioteca **”Arduino-FSM”** ao invés do *Switch:Case* previamente desenvolvido. Esta biblioteca permite a criação de uma máquina de estados mais robusta, permitindo a chamada de funções entre a transição dos estados e durante os estados, além de já contar com uma implementação de transições temporais prontas, que são bastante utilizadas durante a movimentação ad-hoc.

Primeiramente, Como o robô ainda não encontrava-se montado para a realização dos testes, um pequeno circuito foi montado com 1 LED representando cada perna do robô. Assim, toda vez que uma perna deveria ser movimentada, o LED correspondente acenderia. O código foi implementado e executado com sucesso, usando como transição entre os estados um certo período de tempo transcorrido.

Utilizando-se da estrutura do robô pronta, foi testada a implementação da máquina de estados com a movimentação do robô.

Com a finalização da montagem do robô, o próximo passo é colocar a máquina de estados para operar as pernas e verificar a necessidade de outros estados não levantados até agora, além da determinação da condição de transição entre os estados, garantindo que o robô tenha tempo suficiente para executar o movimento de uma perna antes de iniciar o movimento de outra.

6.4 Execução paralela via Threads

Para garantir que o robô possa desenvolver suas funções continuamente sem ter que parar para receber comandos do usuário, tirando proveito dos dois núcleos de processamento do ESP, foi definido que 2 *threads* seriam executadas paralelamente: Uma para a movimentação e outra somente para comunicação com a interface de usuário.

Para isto, optou-se pela utilização da biblioteca *ArduinoThread*, dado a experiência prévia com a mesma. Para sua utilização, são definidas as funções que serão executadas paralelamente ao código principal, depois a thread é inicializada recebendo como parâmetros a respectiva função e o requisito temporal para que a função seja executada novamente. Por último, é criado o controlador, que tem as threads criadas atribuídas a ele.

A primeira thread criada é referente a comunicação, tendo em vista que já existia um protótipo do código da comunicação funcionando. Assim, a função que previamente era chamada no *loop* do Arduino para escutar novas mensagens e respondê-las, agora foi associada a thread *handleCommunication()*.

A segunda thread é a relativa à máquina de estados. O código mencionado na Seção 6.3 foi implementado como uma thread. O novo estado do sistema é sempre setado com a finalização do estado anterior, e somente quando a thread é executada novamente as ações do novo estado são desenvolvidas. Para garantir a execução contínua das chamadas para posicionamento dos servos, a thread será responsável somente por realizar o cálculo das novas posições, enquanto a atualização das posições irá acontecer repetidamente no *loop* principal.

A implementação destas duas threads em paralelo ocorreu com sucesso, demonstrando ser capaz de atender os requisitos temporais do sistema. Entretanto deve-se tomar cuidado com chamadas do tipo *delay* dentro do código das threads, já que o Arduino bloqueia todas as execuções durante o período informado.

Um problema da implementação atual é que não existe nenhum tipo de proteção

quanto as variáveis que são manipuladas pelas Threads, o que pode causar inconsistências. Para resolução deste problema, primeiramente pensou-se em implementar um mutex, que garanta que a escrita dos valores nas variáveis críticas seja realizado por uma operação atômica e por consequência garanta que seu valor não será acessado durante uma operação de escrita. Entretanto, a biblioteca do Arduino não possui suporte a biblioteca de mutex da linguagem C. Portanto, a solução escolhida é uma eventual implementação de um semáforo, prevenindo a leitura durante a escrita, mas garantindo que a thread de leitura tenha acesso a um valor anterior da variável que está sendo modificada.

6.5 Interface de Usuário (UI)

Como mencionado anteriormente, na seção 6.2, por opção da empresa a UI foi desenvolvida pela mesma. Ela é executada em NodeJS e roda em distribuições de Linux e outros sistemas operacionais capazes de executar JavaScript. Apesar de ter sido testada em notebooks convencionais, uma Raspberry Pi deve ser capaz de executar a interface pois se trata de um microcomputador que executa uma versão customizada do Linux. Testes serão feitos no futuro para garantir a compatibilidade da interface com a plataforma alvo.

Vale ressaltar que, por ter sido desenvolvida e fornecida pela própria empresa, a interface é aceita pela empresa como uma maneira de atender os requisitos de operação, mesmo que a mesma não tenha sido testada em uma Raspberry Pi.

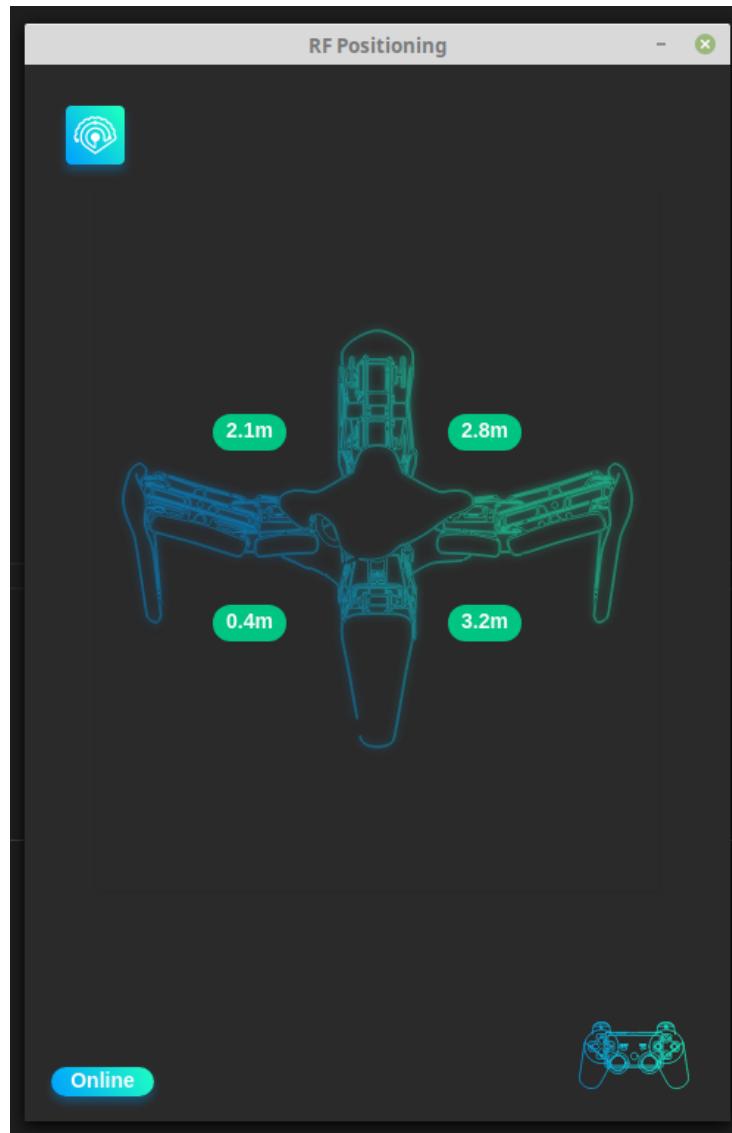


Figura 27: Interface de Usuário implementada pela empresa Seashell.

A operação da interface é realizada através de um joystick do tipo *DualShock 3*, da Sony, e sua operação foi validada pelos membros do grupo. A comunicação via WebSocket com o ESP32 também foi validada, com as duas pontas recebendo e enviando os dados já mencionados anteriormente.

6.6 Updates Over-The-Air

Ao testar o ESP-32 em uma versão do shield desenvolvido pela equipe sem a devida alimentação externa, o shield acabou drenando muita corrente do pino Vin do ESP-32, responsável por alimentar o mesmo. Com isso, o chip USB-Serial CP2102 da placa acabou sendo danificado, impossibilitando a comunicação com a placa da maneira correta. Este incidente acabou por atrasar uma parte do desenvolvimento do projeto, dado que, com somente um ESP-32 disponível em mãos, uma solução teria que ser encontrada para a continuação do projeto.

Após extensa procura nos datasheets do ESP-32, foi levantada a possibilidade de programar a placa diretamente via os pinos TX e RX do ESP-32, ignorando o chip serial danificado. Entretanto, os chips USB-Seriais convencionais não seriam capazes de executar tal tarefa devido a sequência necessária de boot do ESP para acesso de seu bootloader. Assim, através de um conversor FTDI (USB-Serial) especial, que disponibiliza acesso aos pinos RTS e DTR que irão informar ao ESP que sua rotina de programação está sendo chamada. Utilizando-se então da ferramenta de programação esptool.py e de algumas soldas realizadas diretamente no core do ESP-32 para acesso de pinos que não são disponibilizados na placa de desenvolvimento, conforme figura 28, foi possível realizar a programação do ESP novamente.

De modo a garantir que o ESP pudesse ser programado novamente no futuro, implementou-se uma biblioteca para updates over-the-air do próprio ESP-32. Assim, qualquer usuário conectado a sua rede pode fazer o upload dos códigos que serão executados na placa direto para a mesma, enquanto a mesma salva o código em sua memória e executa as rotinas de inicialização necessárias para utilizar o novo código. Ainda como vantagem dessa implementação, não se mostra mais necessário ter o robô conectado ao computador para programação, podendo este ser programado desde que esteja no alcance do computador do usuário.

Automatic bootloader

`esptool.py` can automatically enter the bootloader on many boards by using the RTS and DTR modem status lines to toggle GPIO0 and EN automatically.

Make the following connections for `esptool.py` to automatically enter the bootloader:

ESP32 Pin	Serial Pin
EN	RTS
GPIO0	DTR

Figura 28: Detalhe do datasheet sobre a programação via pinos TX/RX e os pinos adicionais necessários.

7 Implementação do movimento

O movimento do robô foi validado previamente na teoria e por simulações, conforme apresentado no capítulo "Simulação V-rep".

Para permitir o desenvolvimento paralelo com o código desenvolvido para o ESP-32, a movimentação foi em um primeiro momento realizada em um Arduino Mega, sendo depois somente necessário copiar o código no código do ESP-32 e realizar pequenos ajustes de integração para garantir a compatibilidade com a abstração de objetos.

Entretanto, ao transportar essa movimentação validada em simulação para o robô físico real foram necessárias certas adaptações, pois certos detalhes não foram levadas em conta nas simulações. As principais situações não consideradas na simulação e que influenciaram na implementação foram:

- O atrito da perna com o chão não é muitas vezes o suficiente para o movimento proposto;
- Os servo-motores perdem o passo com o desenrolar do movimento;
- A força dos servo-motores não é suficiente para realizar certos movimentos, principalmente ao encostar no chão após estar levantada;
- A queda do robô acontece com mais facilidade do que na simulação, necessitando ter ângulos bem ajustados em cada servo-motor. A velocidade do movimento de cada perna também é considerável.

Para contornar essas situações foram realizados testes para encontrar a posição ideal de cada ângulo de cada servo-motor. Individualmente foram levantados esses parâmetros e continuamente validadas colocando o robô no cenário real de aplicação. Para isso foi desenvolvido um código em Arduino que mapeia todos os ângulos definidos e atualizar individualmente o valor do ângulo de acordo com o ID de cada servo-motor. As posições¹ encontradas para cada servo podem ser visualizadas na tabela 4.

¹As posições na tabela 4 são dadas por quantidade de passos internos do servo-motor. Apesar de que no relatório foi utilizado a nomenclatura "ângulo"

	Perna 1			Perna 2			Perna 3			Perna 4		
Posição\Servo	C	F	T	C	F	T	C	F	T	C	F	T
COXA_DEFAULT	0	1	15	2	3	4	5	6	8	9	10	12
COXA_UP	400			390			400			390		
FEMUR_DEFAULT		270			550			300			440	
FEMUR_UP			270			480			520			480
TIBIA_DEFAULT				510			340			500		
TIBIA_UP				350			350			600		
FORWARD_COXA_1 (B)	510			470			470			500		
FORWARD_COXA_2 (A)	250			210			210			240		
FORWARD_COXA_3 (C)	260			230			230			250		
FORWARD_COXA_4 (D)	550			470			500			500		

Tabela 4: Tabela com as posições utilizadas em cada servo-motor. Os dados coloridos são interpretados de cima para baixo, cada cor faz parte de uma sequência de movimentos que o robô faz até completar um passo. (C - coxa, F - femur, T - tibia)

A melhor posição foi manter o conjunto de fêmur do robô levemente arqueadas para cima, assim como a tibia de cada robô mantendo um ângulo um pouco menor do que 90º, evitando com que o atrito abra as pernas.



Figura 29: Posicionamento ideal do robô para minimizar o efeito do atrito sobre as pernas

Na figura 32 é possível ver uma atualização quanto à movimentação previamente apresentada. As pernas não mais esticarão para fazer o movimento MOVE. Esse movimento aumentaria a complexidade, visto que mais dois servo-motores teriam

que ser acionados, e não traria ganhos significativos, uma vez que o robô é de uma dimensão bastante grande.

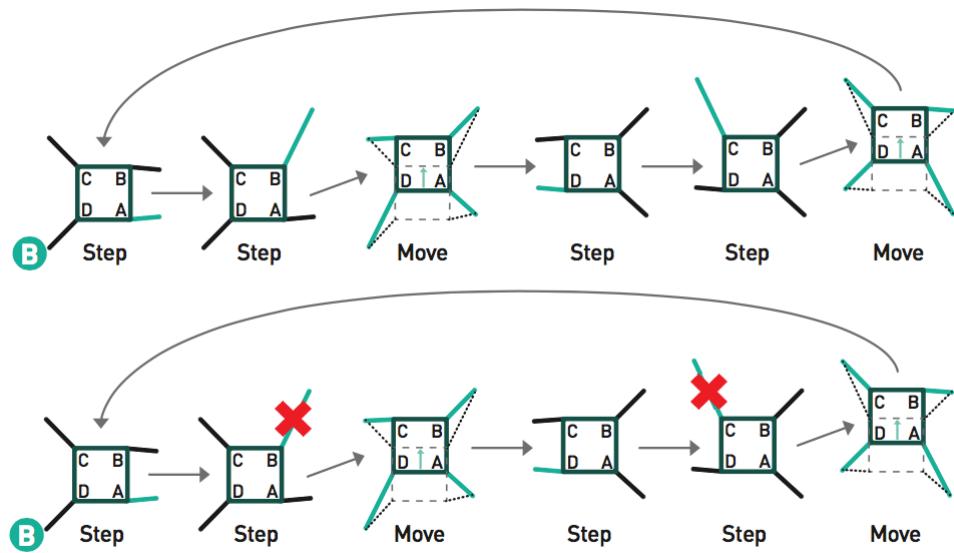


Figura 30: Movimentação final do robô

8 Resultados obtidos

Ao longo do projeto foram percebidas várias etapas do desenvolvimento e vários problemas que não eram esperados e que acabaram atrasando o desenvolvimento do projeto. Sendo assim, através dos diagramas de Gannt ilustrados abaixo, é possível realizar uma comparação dos períodos de desenvolvimento de cada uma das macro-atividades.

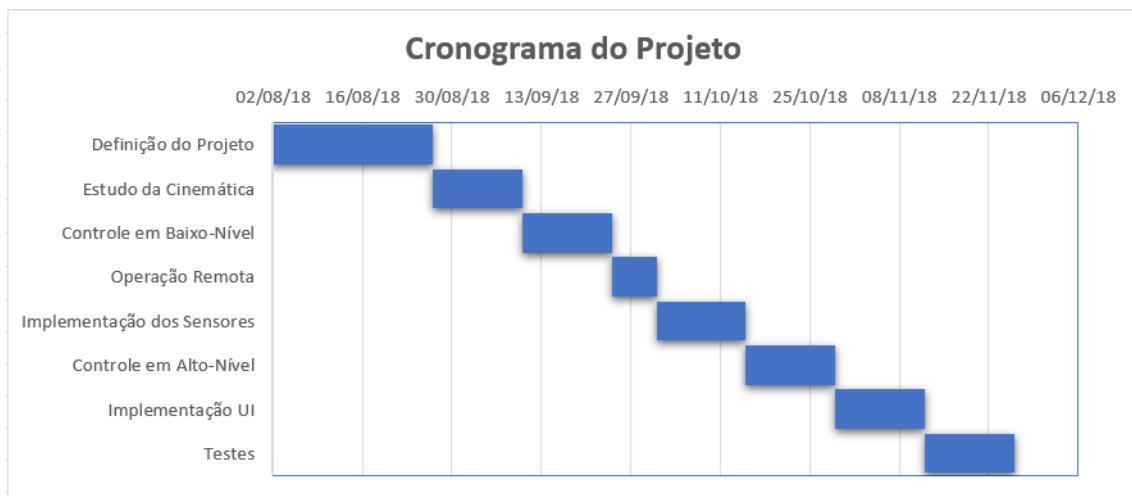


Figura 31: Diagrama de Gannt inicial

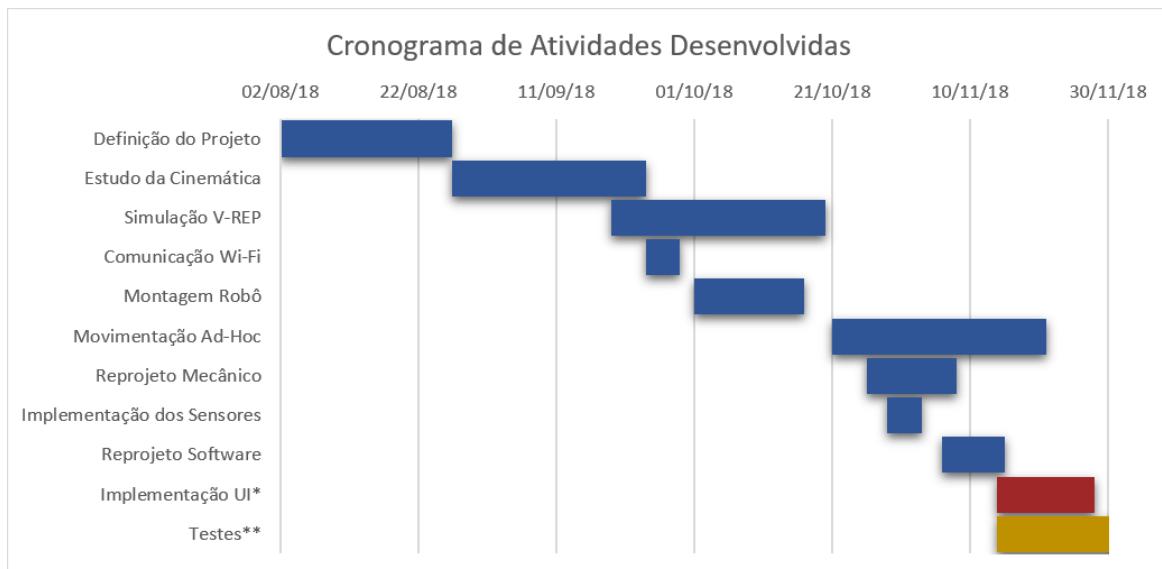


Figura 32: Diagrama de Gannt das atividades realizadas. *: Atividade realizada pela empresa seashell. **: Atividade em desenvolvimento.

Do que foi planejado, ficou faltando concluir a integração entre os códigos de movimentação e o código que realiza a comunicação e a leitura dos sensores, que mostrou-se mais complicado do que o planejado. A expectativa é que mais duas semanas de trabalho sejam suficientes para entregar o trabalho concluído para a empresa. Esta situação já foi explicada para a mesma, que retornou um feedback muito positivo do trabalho realizado e que espera manter contato com os membros do grupo para continuação do projeto.

9 Conclusão

Com o escopo do projeto definido inicialmente em conjunto com a empresa de desenvolver uma plataforma de testes físicas do software implementado pela empresa *Seashell* no qual necessitaria de uma integração com usuário para leitura de variáveis e ações de controles, foi determinado o desenvolvimento de uma plataforma de robô quadrupede que atendesse os seguintes requisitos ao longo do semestre.

- Movimentar o robô;
- Implementar o sensoriamento;
- Desenvolver uma resposta de movimento para a leitura de sensores;
- Realizar a comunicação entre as interfaces do sistema;
- Controlar a plataforma através de uma interface remota.

Dentre as tarefas listadas acima, o grupo obteve sucesso em atendê-las, apesar de um tempo curto para implementação otimizada dos algoritmos de movimentação, foi desenvolvido uma lógica a partir de uma sequência de estados para efetuar a movimentação para os diferentes direções da estrutura.

Através da utilização de um micro controlador ESP-32 para implementação da movimentação, leitura dos sensores e comunicação com um cliente externo sob a interface gráfica desenvolvida com auxílio da empresa utilizando comunicação Wi-Fi via WebSockets é possível atuar sobre a estrutura remotamente.

Observando o escopo inicial do projeto, podemos concluir que os objetivos iniciais da empresa foram atendidos pelo grupo, porém ainda existe pequenos aperfeiçoamentos a serem efetuadas para uma movimentação mais suave e otimizadas.

A empresa acompanhou o projeto durante todas as etapas e mostrou-se bastante satisfeita com os resultados obtidos, esperando manter contato para finalização e entrega do projeto nas próximas semanas.