Federação das Indústrias do Estado da Bahia

CENTRO UNIVERSITÁRIO SENAI CIMATEC

Engenharia Elétrica

Projeto Theoprax de Conclusão de Curso

Desenvolvimento do robô de inspeção.

Apresentada por: Carlos Alberto Pereira
Cleber Couto Filho
Davi Costa
Ícaro Nascimento

Orientador: Prof. Marco Reis, M.Eng.

Setembro de 2018

Carlos Alberto Pereira
Cleber Couto Filho
Davi Costa
Ícaro Nascimento

Desenvolvimento do robô de inspeção.

Projeto Theoprax de Conclusão de Curso apresentada ao , Curso de Engenharia Elétrica do Centro Universitário SENAI CIMATEC, como requisito parcial para a obtenção do título de **Bacharel em Engenharia**.

Área de conhecimento: Interdisciplinar

Orientador: Prof. Marco Reis, M.Eng.

Salvador
Centro Universitário SENAI CIMATEC
2016

Resumo

Este documento contempla a descrição das etapas do desenvolvimento do projeto de Theoprax de Conclusão de curso, ELIR, robô autônomo de inspeção de linhas de transmissão, atendendo aos objetivos gerais e específicos e aos requisitos estabelecidos pelo cliente, tendo em vista a necessidade do projeto num cenário tanto comercial quanto acadêmico. Durante o desenvolvimento do projeto foi necessário realizar o estudo de conceitos de robótica, bem como os softwares necessários para implementação das funcionalidades, também estudadas e definidas pelo grupo. Em paralelo ao desenvolvimento das funcionalidades diversos testes foram realizados para validação dos conceitos e verificação de erros, em etapas de testes individuais partindo para a etapa de testes integrados. Os conceitos estudados e desenvolvidos pelo grupo durante todo o projeto fazem parte de uma grande contribuição tecnológica para a área de robótica e engenharia, sendo um projeto enriquecedor tanto para a equipe envolvida no desenvolvimento quanto para as gerações futuras interessadas no desenvolvimento tecnológico em robótica.

Palavras-chave: Palavra-chave 1, Palavra-chave 2, Palavra-chave 3, Palavra-chave 4, Palavra-chave 5

Abstract

This document contain a description of the development stages of the Theoprax end of course project, the ELIR, a autonomous robot for inspection transmission lines, meeting the general and specific objectives and the requirements established by the client, considering the need of the project in a scenario both commercial and academic. During the development of the project, it was necessary to carry out the study of robotics concepts, as well as the software required to implement the functionalities, also studied and defined by the group. In parallel with the development of the functionalities, several tests were carried out to validate the concepts and verify errors, in individual test stages, starting to the integrated testing stage. The concepts studied and developed by the group throughout the project are part of a great technological contribution to the area of robotics and engineering, being a project enriching both for the team involved in in development and for the team involved in development and for future generations interested in the technological development in robotics.

Keywords: Keyword 1, Keyword 2, Keyword 3, Keyword 4, Keyword 5

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

THEOPRAX
WWW ..... World Wide Web

Lista de Simbolos

∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble
∂	Bla bla bla
$\prod$	ble ble ble

Capítulo Um

Introdução

No Brasil, a eletricidade é gerada por hidrelétricas, termoelétricas, parques eólicos e usinas nucleares. Na maioria dos casos, devido a condições geográficas e de segurança, a energia gerada nem sempre é utilizada ou consumida no local de sua geração. Portanto, há a necessidade do uso de linhas de transmissão para transportar energia gerada na fonte geradora para a carga do consumidor (**??**). O mercado consumidor brasileiro é composto de cerca de 47 milhões de unidades. Em termos de linhas de transmissão de energia, são cerca de 98.648,3 km, que devem estar operando 24 horas por dia, 7 dias por semana, 365 dias por ano e em perfeito estado de manutenção, para garantir eletricidade para os consumidores (**??**)

No Brasil, há uma quantidade considerável de linhas de transmissão de alta tensão que já ultrapassaram a vida útil as quais foram destinadas. Com o envelhecimento dos cabos, a inspeção para manutenção preventiva é um fator de extrema relevância para garantir o perfeito funcionamento dos sistemas elétricos. De um modo geral, as inspeções nas linhas de transmissão de alta tensão são realizadas regularmente de forma visual, a fim de identificar a necessidade da realização de manutenções preventivas. As inspeções buscam verificar a integridade física dos componentes das linhas, em termos de fissuras, corrosão e eventuais danos que venham a prejudicar o fornecimento de energia elétrica. Essas inspeções envolvem a análise da integridade estrutural das torres, da condição dos isoladores, das conexões das linhas de transmissão, dentre outros, a fim de se verificar a existência de eventuais pontos de ruptura.

Um dos métodos empregados para detecção de pontos quentes nos cabos é o imageamento térmico, que é capaz de identificar uma elevação de temperatura nos cabos, o que é um indício de possíveis pontos de ruptura. A inspeção através de câmera térmica é uma importante ferramenta no campo das inspeções para manutenções preventivas. Outros pontos a serem inspecionados envolvem as condições do local onde as torres são instaladas, pois a vegetação e eventuais construções devem ser mantidas a uma distância mínima segura, tal que não ocorra nenhum contato entre quaisquer estruturas e as torres ou cabos de transmissão, evitando assim interferências no funcionamento da linha.

Além disso, é essencial a garantia de dispor-se de um terreno em condições de trânsito de veículos para o transporte do pessoal de manutenção, transporte de ferramentas, dentre outros fatores. Durante vários anos, a inspeção de linhas de transmissão de alta tensão tem sido feita regularmente através de aeronaves tripuladas. As aeronaves executam vôos

em baixa altitude e muito próximos das linhas de transmissão conforme mostrado nas Figuras 1.1 e 1.2.

Figura 1.1: Inspeção de linhas de transmissão feita por aeronaves tripuladas.

Figura 1.2: Interação humana durante a inspeção de linhas de transmissão.

Em alguns casos, devido às características geográficas da região, condições climáticas e outros fatores que venham a dificultar o sobrevôo, há uma grande exposição dos tripulantes a riscos associados à tarefa. Além dos perigos aos quais os tripulantes são expostos, a inspeção feita com aeronaves tem um custo bastante elevado. Outra forma alternativa de inspeção é o uso de veículos terrestres, porém essa forma é muito limitada, pois boa parte das linhas de transmissão está localizada em áreas de difícil acesso terrestre, muitas vezes restritas pelas características geográficas da região. Além disso, o ângulo de visão é, muitas vezes, desfavorável para a realização da inspeção.

Outra maneira de inspecionar as linhas de transmissão é através de eletricistas que literalmente caminham sobre os cabos de linhas de transmissão de alta tensão (Figura

Figura 1.3: Realização de inspeção em linhas de transmissão através da observação humana.

1.3), realizando inspeção visual e termográfica. Esse tipo de inspeção é lenta e não é viável, tendo em vista que o país possui milhares de quilômetros de linhas de transmissão.

Neste contexto vários robôs de inspeção de linhas de transmissão foram desenvolvidos, porém poucos deles consistiram em projetos de engenharia que sejam aplicáveis no mundo real, além disso a maioria eram robôs tele-operados, ou seja robôs controlados por seres humanos. Um dos pontos diferenciais deste projeto de tese é a proposição de um desenvolvimento de uma navegação autônoma utilizando técnicas de aprendizagem de máquinas até então não utilizadas em robôs de inspeção de linhas de transmissão de alta tensão.

1.1 Objetivos

Desenvolver um protótipo de um robô de inspeção de linhas de transmissão

1.1.1 Objetivos Específicos

Desenvolver 4 funcionalidades para o robô, sendo elas:

- Actuation

  Funcionalidade responsável por atuar os motores que movimentam o robô.

- Motion Planning

Tem como objetivo planejar a movimentação das partes do robô

- Power Management

  Responsável pelo gerenciamento e distribuição de energia para os componentes elétricos e eletrônicos do robô.

- System Integrity Check

  Checagem da integridade do sistema antes do robô entrar em funcionamento.

  content...

1.2 Justificativa

1.3 Requisitos do cliente

Foi especificado pelo cliente que o robô ELIR realize tais funções:

- Transpor obstáculos e cadeia de isoladores;
- Deslocar-se através do consumo de baterias;
- Deslocamento/movimento realizada por servomotores;
- Realizar as funções de forma autônoma.

1.4 Organização do Projeto Theoprax de Conclusão de Curso

Este documento apresenta x capítulos e está estruturado da seguinte forma:

- **Capítulo 1 - Introdução**: Contextualiza o âmbito, no qual a pesquisa proposta está inserida. Apresenta, portanto, a definição do problema, objetivos e justificativas da pesquisa e como este projeto theoprax de conclusão de curso está estruturado;
- **Capítulo 2 - Conceito do Sistema**: Conceitua o sistema por meio de diagramas que representam as arquiteturas do robô em diferentes níveis de abstração, abordando o estudo do estado da arte, desdobramento dos requisitos de qualidade e funcionamento do projeto. ;
- **Capítulo 3 - Materiais e Métodos**: Mostra os materiais e métodos que foram utilizados durante o projeto, contendo a especificação de componentes, sendo eles

softwares e dispositivos, assim como os esquemas elétricos e eletrônicos - e a descrição das funcionalidades;

- **Capítulo 4 - Resultados**: Exibe os resultados obtidos durante a confecção do projeto, apresentando os testes unitários e integrados, assim como as datas e quem o efetuou;
- **Capítulo 5 - Conclusão**: Apresenta as conclusóes, contribuições e algumas sugestões de atividades de pesquisa a serem desenvolvidas no futuro.

Capítulo Dois

Conceito do Sistema

Quanto maior for a rapidez de transformação de uma sociedade, mais temporárias são as necessidades individuais. Essas flutuaçõess tornam ainda mais acelerado o senso de turbilh da sociedade.

(Alvin Toffler)

Quanto maior for a rapidez de transformação de uma sociedade, mais temporárias são as necessidades individuais. Essas flutuações tornam ainda mais acelerado o senso de turbilhão da sociedade.

(Alvin Toffler)

2.1 Estudo do estado da arte

flkjasdlkfjasdlkfjs

2.2 Descrição do sistema

lasdjflsadjf

2.2.1 Especificação técnica

lakjfldksjfdslakjf

2.2.2 Arquitetura geral do sistema

lkasjdflksdajflk;

2.2.2.1 Arquitetura do sistema de movimentação

De forma a garantir uma movimentação efetiva do robô é necessária a integração de diversos ferramentas físicas e de software, como a estrutura de movimentação adequada, sistema ordenamento de missão, controle de potência,demandando assim um *framework* e um sistema operacional.

A inspeção de linha foi denominada missão, para cade vez que o robô começar a realizar a inspeção, será considerado o início de uma nova missão.

Para garantir a execução correta da missão e ultrapassagem dos obstáculos de forma efetiva, se dividiu o sistema em 4 principais subsistemas, sendo elas : *Motion Planning* , *Actuation*,*System Integrity Check* e *Power Management*. A arquitetura geral do sistema de movimentação está mostrada na figura 2.1, ilustrando os subsistemas e suas funcionalidades. A estrutura física do robô foi projetada para que sejam realizados movimentos

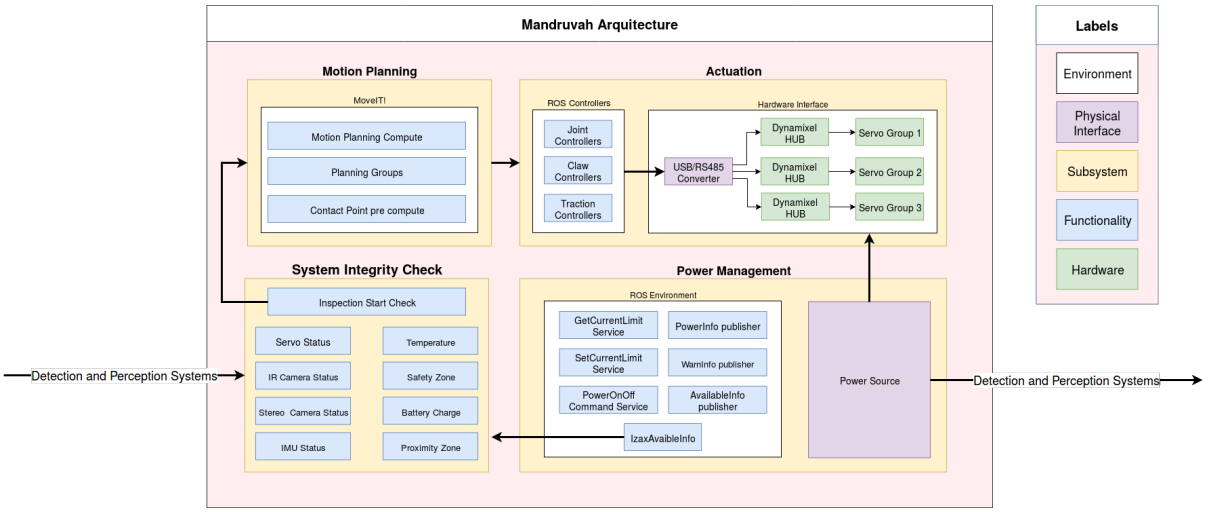

Figura 2.1: Arquitetura Geral do sistema de movimentação

Fonte: Própria

de translação e transposição de obstáculos presentes na linha de transmissão,consistindo de unidades de tração para a translação na linha e juntas nos braços e garras para a realização da ultrapassagem de obstáculos. O controle da estrutura física do robô está relacionado com a *Actuation*. A transposição dos obstáculos é um grande desafio para essa aplicação, visto que será necessário a aplicação da cinemática inversa no robô. A cinemática inversa consiste num conjunto de equações que definem o movimento do robô para a movimentação de um ponto à outro, tal modelo é extraído à partir da estrutura do robô. A funcionalidade responsável por calcular esse modelo e encontrar como será feita a movimentação foi denominada *Motion Planning*. Para garantir a execução correta da missão e preservar a integridade do robô foi estipulada uma funcionalidade que checa os dispositivos antes de cada missão, denominada *System Integrity Check*. E com a finali-

dade de realizar o controle da potência no robô foi será utilizado o projeto de uma placa específica para esse papel, assim todos os aspectos relacionados à alimentação do robô, assim como consumo e monitoramento estão atrelados ao *Power Management*.

O *framework* ROS possibilita a integração de todas essas funcionalidades, sua estrutura baseada em nós facilita a identificação dos problemas e possibilita a modularização do código. Fornecendo também diversas ferramentas como o *MoveIt!*, que será utilizada para o *Motion Planning*, assim como drivers de compatibilização para os servo motores adotados no projeto.

2.2.3 Arquitetura de software

2.3 Desdobramento da função qualidade

asdfsdafsf

2.3.1 Requisitos técnicos

Foi determinado pelo cliente os seguintes requisitos técnicos.

- Desempenho de deslocamento: Percorrer 15km por dia
- Velocidade de deslocamento: Velocidade média sem obstáculos será de 0.5 m/s
- Ultrapassagem de obstáculos: Volume máximo dos obstáculos 410x330x150mm
- Autonomia de Potência: 2 horas de autonomia
- Sistema Operacional: Linux
- Backend: C++ e Python
- Framework: ROS Kinetic Kame

2.3.1.1 Funcionamento do robô

O funcionamento do robô será comprovado por um teste realizado dentro da instituição, em um modelo reduzido de linha de transmissão. Sem condições ambientais adversas, realizando a parada baseada no sinal da câmera de detecção de obstáculos e

com o teste operado manualmente. Serão desenvolvidas rotinas de software para: início da missão; simular detecção de obstáculo e parada emergencial. As especificações para os testes são:

- Condutor: LINNET e diâmetro: 18,3mm;
- Obstáculos: Grampo de suspensão e amortecedor de vibração;
- O robô será colocado manualmente na linha;
- A operação se iniciará à uma distância de 1 metro do obstáculo;
- A parada será realizada com base no sinal do sistema de detecção, a uma distância de 50cm do obstáculo;
- O comando para início da missão será feito por meio do terminal do Linux , por meio de acesso remoto;
- A estrutura para o teste será fornecida pela a empresa;

As etapas para realização do teste são:

- O robô será manualmente posicionado na linha, à uma distância de 1 metro do obstáculo;
- O comando para iniciar a inspeção será enviado via acesso remoto, por meio do terminal Linux;
- Após receber o comando, o robô iniciará um movimento na linha em direção ao obstáculo;
- Ao receber o sinal de obstáculo detectado, o robô irá parar e esperar o pós-processamento do sistema de detecção;
- Após o processamento, irá fazer a ultrapassagem referente ao tipo de obstáculo, e continuar se deslocando na linha;

2.3.1.2 Códigos da programação disponíveis em repositório online

Os códigos produzidos serão disponibilizados na plataforma GitHub, onde os pacotes produzidos durante o projeto estão organizados em 4 repositórios referentes às funcionalidades do robô.

2.3.1.3 Documentação técnica de final de projeto

A documentação foi definida como um relatório denominado Conceptual and Design Report , Databooks com as informações e logbooks com os testes.

2.3.1.4 Protótipo do robô

O cliente disponibilizou as partes mecânicas do robô, sendo entregue pela equipe o protótipo do robô montado. É incluída na montagem a disposição dos cabos e unidade de processamento do robô.

Capítulo Três

Materiais e Métodos

Esta seção destaca o que é necessário para construção do projeto, contendo a lista de materiais, especificação dos componentes, funcionalidades, modelo esquemático de comunicação e de alimentação.

3.1 Especificação dos componentes

Serão detalhados os componentes utilizados para confecção do protótipo, sendo eles físicos ou digitais.

3.1.1 Estrutura analítica do protótipo

3.1.2 Lista de componentes

3.1.3 Servomotores

Os servomotores são responsáveis pelo atuação do robô, realizando os movimentos das juntas dos braços e das garras, além de atuarem as rodas que deslocam o robô na linha. São utilizados os servomotores da Robotis, *Dynamixel* MX-106R e MX-28. Esses motores foram escolhidos pois apresentam drivers prontos para o *ROS* , que possibilitam uma integração mais fácil com as ferramentas de controle, apresentando bom torque, peso reduzido e fácil integração para controle conjunto.

Figura 3.1: Motor Robotis *Dynamixel* MX-28R.

Robotis Dynamixel MX-28R	
Peso (g)	153
Dimensões (mm)	40.2 x 65.1 x 46
Torque (N.m)	8.0 (em 11.1V), 8.4 (em 12V) e 10.0 (em 14.4V)
Temperatura de operação (°C)	-5 até +80
Tensão de operação (V)	10 até 14.8 (Tensão recomendada: 12V)
Baud rate	8000bps até 4.5Mbps
Protocolo de comunicação	RS485
Resolução	0.088°
ID	254 ID (0 até 253)
Feedback	Posição, temperatura, carga, tensão de alimentação, etc.

Tabela 3.1: Especificações Motor Robotis *Dynamixel* MX-28R

Robotis Dynamixel MX-28R	
Peso (g)	153
Dimensões (mm)	40.2 x 65.1 x 46
Torque (N.m)	8.0 (em 11.1V), 8.4 (em 12V) e 10.0 (em 14.4V)
Temperatura de operação (°C)	-5 até +80
Tensão de operação (V)	10 até 14.8 (Tensão recomendada: 12V)
Baud rate	8000bps até 4.5Mbps
Protocolo de comunicação	RS485
Resolução	0.088°
ID	254 ID (0 até 253)
Feedback	Posição, temperatura, carga, tensão de alimentação, etc.

Tabela 3.2: Especificações Motor Robotis *Dynamixel* MX-106R

Figura 3.2: Motor Robotis *Dynamixel* MX-106R

3.1.4 Placa de Gerenciamento de Energia (Power Management)

A placa de gerenciamento de energia é responsável pela distribuição de corrente e de tensão para todos os componentes elétricos e eletrônicos do robô, além de monitorar os níveis de tensão e corrente demandados durante a operação.

Além de realizar o monitoramento do consumo em cada porta individualmente, a placa possui um sistema de proteção, cortando a alimentação em casos de surto de corrente. A placa funciona através da alimentação de 14.4 Volts provenientes da placa multiplexadora, responsável por transmitir a carga de duas baterias *Li-Íon* de 14.4 Volts e 6Ah.

Na placa de *Power Management* existem conversores DC/DC responsáveis por fazer a conversão dos níveis de 14.4 Volts para 12 Volts em cada porta de saída da placa. As 7 portas de saída possuem sensores de tensão e corrente individuais, feitos com amplificadores de instrumentação INA226. Existem duas portas de saída que disponibilizam tensões em valores menores, de 5 Volts.

O monitoramento dos níveis de tensão e corrente se dá principalmente pela inteligência do sistema, um firmware embarcado em um microcontrolador Atmega32u4, responsável por fazer as leituras dos parâmetros em cada uma das portas, verificando se os seus níveis estão de acordo com os limites configurados, cortando a alimentação via relés digitais caso esses valores sejam ultrapassados.

3.1.5 ROS (Robot Operating System)

Framework, no ambiente de programação, é um espaço onde compatibiliza códigos comuns a fim de otimizar o trabalho e tempo, muito utilizado na área de desenvolvimento. A abstração de hardware, códigos de baixo nível, drivers de sensores, simuladores, etc - são as grandes vantagens de se utilizar essa aplicação, podendo assim,fazer com que o desenvolvedor foque somente nas soluções de problemas específicos do seu projeto.

Foi utilizado durante todo o desenvolvimento do *ELIR* o *framework ROS*, já que

reúne uma série de ferramentas importantes para o desenvolvimento de um robô. Como colocado no site oficial do *ROS* temos que o Sistema Operacional de Robótica é um flexível *framework* para escrita de softwares para robótica. É uma coleção de ferramentas, bibliotecas e convenções que serve para simplificar a tarefa de criar complexos e robustos comportamentos de robôs diante a uma variedade de plataformas.

A grosso modo, cada câmera, motor ou periférico ligado ao *ROS*, estão associados ao um nó. A comunicação entre os nós se dá através de tópicos ou de serviços, a diferença é que o primeiro a informação é trocada de forma constante com certo intervalo de tempo e o segundo somente quando solicitado.

Assim é feita toda a comunicação e interligação entre os periféricos no *ROS*, forma simples de integração dos componentes.

3.1.6 MoveIt!

Durante a inspeção de linha, é necessário que o robô realize a ultrapassem dos diferentes tipos de obstáculos que existem nas linhas de transmissão. O *MoveIt!* é um ferramenta que funciona de forma integrada com o *ROS*, apresentando funcionalidades de planejamento de movimento, percepção 3D,controle , manipulação e cinemática inversa.

A cinemática é o estudo do movimento, no âmbito da robótica designa o estudo do controle da posição do robô no espaço. Esse controle pode representar do robô como um todo, sua posição geográfica, ou controle de alguma parte sua em específico, como seu braço e a posição relativa desse braço e o robô. A cinemática direta é o cálculo onde se encontra a posição do robô para determinado valor de velocidade ou posição de suas juntas. Analogamente, na cinemática inversa, se encontra os valores de velocidade ou posição das juntas para uma posição no espaço,onde essa posição é denominada *end-effector*, geralmente sendo definido como a parte do robô que interage com o mundo, como por exemplo a garra no caso de manipuladores. O cálculo da cinemática inversa envolve equações complexas e retorna diferentes soluções, assim sendo necessário encontrar a solução que melhor atende às diretrizes do movimento, o *MoveIt!* já realiza esse cálculo e fornece uma trajetória otimizada baseada em parâmetros do usuário.

Outra das suas vantagens é utilizar o modelo *URDF* do robô. *URDF* é uma sigla para Unified Robot Description Format, e designa um arquivo com extensão `.urdf` e sintaxe em XML. É um dos tipos de modelos mais utilizados na robótica atual,sendo escolhido pois apresenta uma sintaxe simples e dinâmica, proporcionando conversões em outros formatos de forma fácil. Define o robô como um conjunto de partes, chamadas de *links* onde a união entre essas partes é uma junta. Onde cada *link* vai ter um *link*

pai, que é determinado pela definição da junta, assim o modelo apresenta uma estrutura em árvore, onde todos os *links* vão ter um pai até chegar ao *link* da raíz,essa definição é importante pois a partir disso é feita a cinemática inversa do robô. Um erro no modelo *URDF* acarreta em uma mudança no comando que é mandado para o robô original.

Durante o desenvolvimento do projeto, foi possível realizar a integração do *MoveIt!* com o robô real, sendo possível realizar movimentos físicos utilizando a ferramenta de visualização para posicionamento de end-effector pelo usuário. Porém, ao tentar se enviar um comando com o valor de posição para o end-effector se mover, o programa falhou em encontrar soluções para a cinemática. Mesmo se utilizando os diversos solucionadores providos e enviando valores possíveis de se calcular, o programa sempre estava falhando em encontrar uma solução.

O *MoveIt!* é designado para funcionar com robôs de 6 graus de liberdade, onde cada grau de liberdade indica uma coordenada que o end-effector pode se mover e um dos 3 eixos de referência (x,y,z) que ele pode girar. A grande quantidade de graus de liberdade faz com que os solucionadores utilizem formas de cálculos complexas, assim robôs que possuem menos que 6 graus de liberdade precisam ser compatibilizados, já que a solução leva em consideração todas as direções e giros. O Elir possuI somente 2 graus de liberdade, e as soluções que antes funcionavam não estavam se utilizando especificamente da solução de cinemática inversa provida pelo *MoveIt!*. Assim, decidiu-se realizar o cálculo da cinemática inversa por meio de um código python, que utiliza a equação de cinemática inversa específica para o tipo de braço do robô e fornece os ângulos de junta necessários para o end-effector especificado.

Com o ângulo de junta em mãos, é feita a integração do robô com a ferramenta, de forma que o planejamento de movimento ocorre, só que com o software recebendo um ângulo desejado. Conhecendo os possíveis valores de end-effector, o que pode ser encontrado pela ferramenta de visualização, é possível realizar o movimento no robô enviando somente um comando de coordenadas.

Implementar o controle de movimento nessa plataforma possibilita uma série de implementações futuras que aumentam a autonomia do robô e robustez do sistema, como odometria, ferramentas de percepção e mapeamento.

3.1.7 Gazebo

O software *Gazebo* é software utilizado para simulação de robôs. Tem uma licença de uso livre e apresenta diversas formas de integração com o *ROS*, sendo o principal simulador utilizado em conjunto com essa plataforma, possibilitando a inserção de plugins

como câmeras e sonares, que se comunicam com o *ROS* de forma fiel a dispositivos reais.

Nele é possível simular também o ambiente do robô, definindo parâmetros físicos como aceleração da gravidade e vento. Oferece suporte para a inserção de modelos 3D de softwares CAD, assim podendo ser inseridas diversas estruturas que já foram modeladas para outros propósitos no software.

3.1.8 Visual Studio Code

Para que todas as funcionalidades do robô sejam configuradas e desenvolvidas de forma correta a nível de software, é necessário o desenvolvimento de diversos códigos em diferentes linguagens para a configuração de aspectos específicos do projeto.

Foi utilizada durante o desenvolvimento do projeto a ferramenta *Visual Studio Code*. Trata-se de um editor de códigos open source desenvolvido pela *Microsoft* em 2015, sendo possível desenvolver códigos em diversas linguagens como C++, C, Python entre outros. Durante todo o desenvolvimento do *ELIR* a ferramenta fora utilizada para o desenvolvimento de arquivos nas extensões .py, .yaml, .launch e .urdf.

Por ter uma interface simples e amigável, o *VSCode* mostrou-se uma ferramenta extremamente útil para a escrita e desenvolvimento de códigos durante todas as fases do projeto.

3.1.9 PlatformIO

Na interface do existem diversas extensões que podem ser instaladas para adicionar novas funcionalidades na plataforma.

Uma das extensões utilizadas fora o *PlatformIO*, um ecossistema desenvolvido especificamente para o desenvolvimento de códigos e firmwares em plataformas microcontroladas, sendo extremamente versátil, tendo suporte para diversas plataformas como STM, MSP430, Arduino entre outras, tornando desnecessário o uso de uma IDE específica para se realizar a configuração e desenvolvimento de firmwares durante o projeto.

A necessidade de se utilizar essa extensão se deu principalmente pela necessidade de se embarcar o firmware da Power Management na placa. O *PlatformIO* possui as funcionalidades de debug e gravação, sendo assim, todos os procedimentos necessários para atualizar o firmware são atendidos na extensão.

3.2 Diagramas mecânicos

3.3 Modelo esquemático de alimentação e comunicação

3.3.1 Diagramas elétricos

Devido à quantidade de motores presentes no robô e a forma com que eles estão distribuídos na estrutura, foram desenvolvidos dois modelos de hubs para a conexão dos motores na rede.

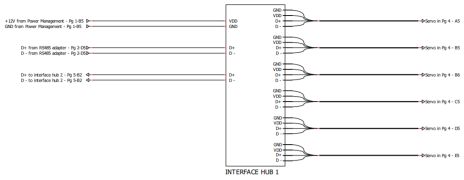

Figura 3.3: Esquema HUB.

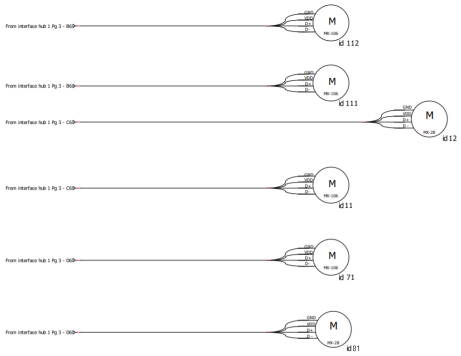

Figura 3.4: Esquema HUB2.

3.3.2 Esquemas eletrônicos

Nas unidades de tração do robô, os hubs contam com um conector de alimentação, um para a entrada de dados e 6 de saída para os motores. IMAGEM

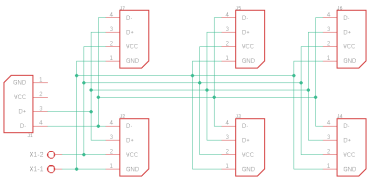

Figura 3.5: Esquema HUB2.

Já na unidade central, o hub além de conectar os seis motores ali presentes, também

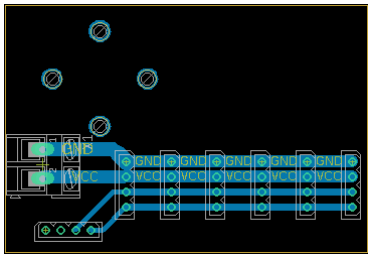

Figura 3.6: Esquema HUB2.

é responsável pela conexão dos hubs das unidades de tração e do conversor rs485 que está ligado à *NUC*.

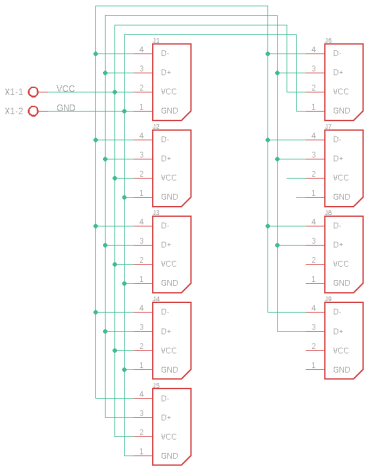

Figura 3.7: Esquema HUB2.

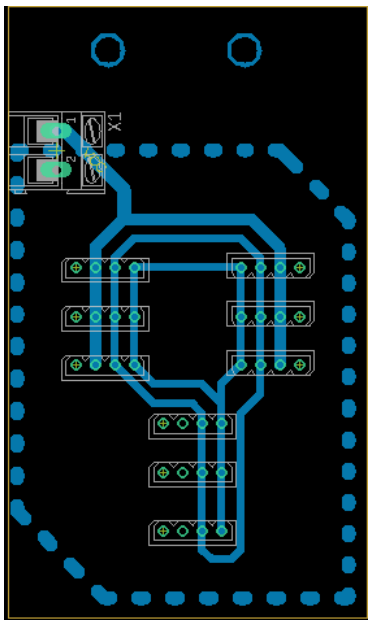

Figura 3.8: Esquema HUB2.

3.4 Especificação das funcionalidades

3.4.1 Fluxo das informações

3.4.2 Motion Planning

3.4.2.1 Definição da funcionalidade

A funcionalidade de *Motion Planning* é responsável por realizar o planejamento da trajetória do Robô, utilizando o software *MoveIt!* que realiza o cálculo da cinemática inversa para encontrar a melhor forma de ultrapassar os obstáculos.

3.4.2.2 Dependências

O software *MoveIT!* pode utilizar o modelo matemático da cinemática inversa do robô ou um arquivo do tipo *URDF*. O nome *URDF* é uma sigla para *Unified Robot Description Format*, esse arquivo é uma especificação em `XML` utilizada para descrever robôs. Modelos em *URDF* apresentam uma simplicidade na descrição do robô, e para o caso do Robô *Elir*, utilizar o modelo *URDF* possibilitará uma aproximação fiel ao modelo real do robô, assim para o cálculo da cinemática inversa será utilizado o seu modelo *URDF* e não o seu modelo matemático.

3.4.2.3 Premissas Necessárias

Para o correto funcionamento dessa funcionalidade as seguintes premissas são necessárias:

- A configuração dos limites de giro das juntas do robô estarão compatíveis com os comandos enviados
- O modelo *URDF* do robô estará adequado com o modelo físico
- O pacote gerado pelo *MoveIt! Setup Assistant* estará configurado adequadamente

3.4.2.4 Descrição da Funcionalidade

A movimentação do robô na linha acontecerá por movimentos de translação e transposição de obstáculos. A translação na linha será feita por controladores de torque nas rodas do robô, enquanto a transposição do obstáculos utilizará o *MoveIT!*. Por meio da ferramenta *MoveIt! Setup Assistant*, se utiliza o modelo do robô para criar um pacote do *ROS* com os principais arquivos pelo *MoveIT!*. A configuração correta do *MoveIT!* possibilita que se utilizem as funções da sua biblioteca para o cálculo da trajetória, levando em consideração também obstáculos no caminho.

O *MoveIT!* fornece uma *user interface* que recebe o end-effector,a nomenclatura atribuída ao node feito em python que recebe o *end-effector* é `moveit_commander`. O *node* responsável por fazer a integração da user interface com os parâmetros recebidos pelo *ROS Parameter Server* com o *end-effector* para fazer os cálculos é denominado `move_group`. O *node* `move_group` também pode receber parâmetros como leituras dos sensores do robô e nuvens de pontos.

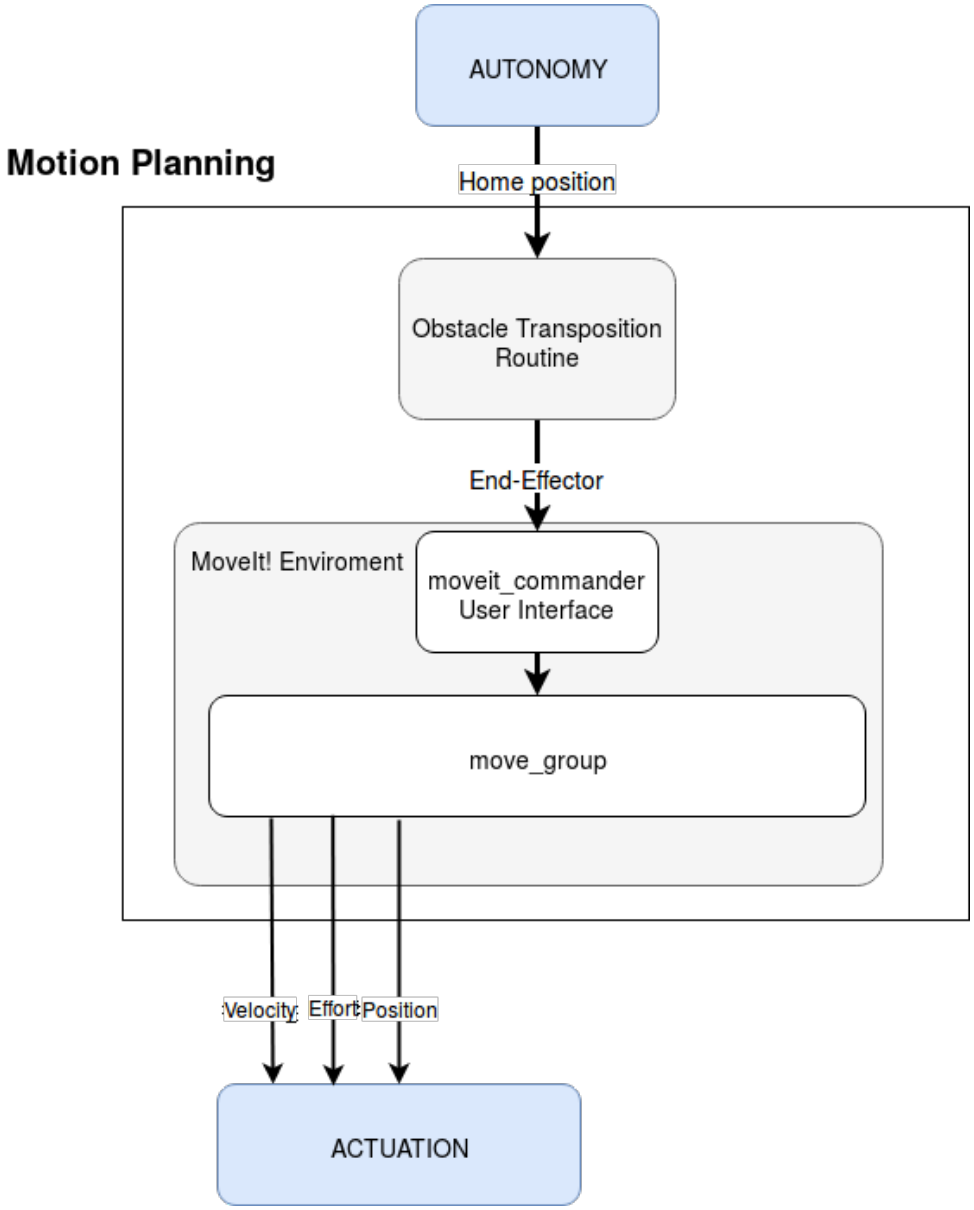

Figura 3.9: Fluxograma de funcionamento da funcionalidade de Motion Planning

Fonte: Própria

3.4.2.5 Saídas

Por meio da compatibilização do *MoveIt!* com o *ROS*, a saída dessa funcionalidade são os comandos de velocidade, esforço e posição para cada junta do robô.

3.4.3 Actuation

3.4.3.1 Definição da funcionalidade

A funcionalidade de Actuation tem como objetivo mover a estrutura física do robô, possibilitando o controle dos movimentos das juntas, garras e unidades de tração.

3.4.3.2 Dependências

Essa funcionalidade depende das funcionalidades de *Power Management* e *Motion Planning*. O *Power Management* será responsável por fazer alimentação dos motores, possibilitando controlar a corrente máxima fornecida para cada grupo. A dependência em relação à funcionalidade de *Motion Planning* está atrelada principalmente com o software *MoveIt!*, que ao receber um *end-effector*,realiza o cálculo de trajetória e envia os comandos de velocidade, esforço e posição para os controladores das juntas, garras e unidades de tração.

3.4.3.3 Premissas Necessárias

Para o correto funcionamento desse módulo, devem ser consideradas as seguintes premissas:

- Os motores devem estar configurados de acordo com o padrão de ID determinado pela equipe, fazendo parte da mesma malha de controle;
- Os controladores das juntas,garras e unidades devem estar configurados de acordo com os comandos que serão recebidos pelo *MoveIt!*;
- Os 3 grupos de motores estarão em malhas de alimentação de 12V individuais.

3.4.3.4 Descrição da Funcionalidade

O *ROS* disponibiliza uma série de drivers para compatibilização dos motores dynamixel, possibilitando a criação de controladores específicos no seu ambiente. Serão criados os controladores referentes as juntas e unidades de tração do robô.Os controladores receberão comandos de *velocity* e *position* do *MoveIt!* junto com os comandos para

movimentar o robô na linha. Após os comandos serem recebidos pelos controladores, eles serão enviados para o *hardware* do robô, de acordo do padrão de comunicação dos motores, por meio de comunicação serial.

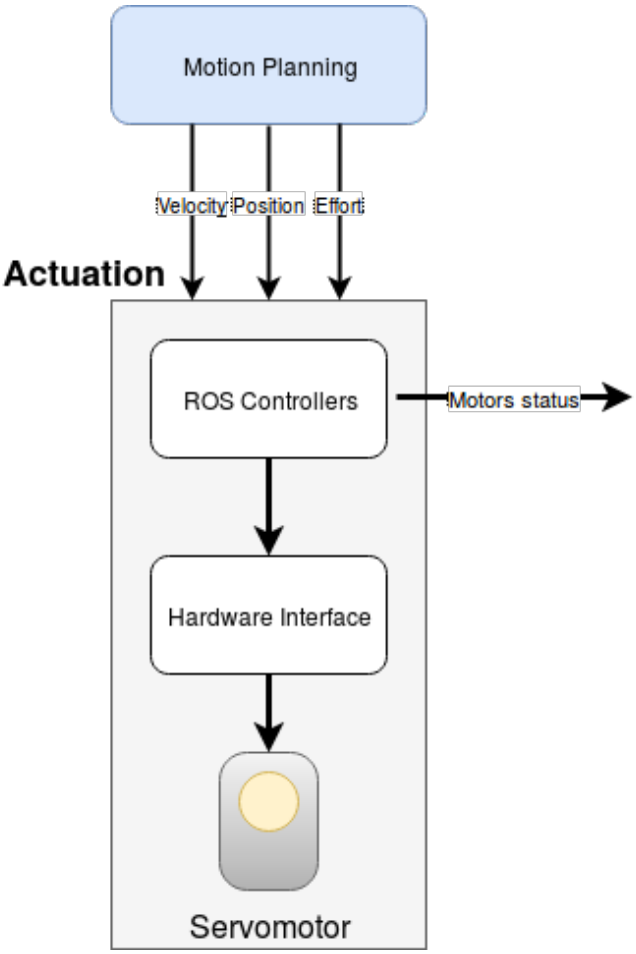

Figura 3.10: Fluxograma da funcionalidade Actuation

Fonte: Própria

3.4.3.5 Saídas

A saída desta funcionalidade é o movimento da estrutura física do robô, que estará de acordo com o planejamento de trajetória do *MoveIt!* e com as instruções para operação na linha

3.4.4 Power Management

3.4.4.1 Definição da funcionalidade

A funcionalidade de *Power Management* é responsável pelo gerenciamento de alimentação elétrica dos componentes elétricos e eletrônicos do robô, através da integração das funcionalidades de seu firmware no ambiente *ROS*.

3.4.4.2 Dependências

Essa funcionalidade depende da comunicação serial por meio da biblioteca `rosserial` para compatibilização e integração das funcionalidades de firmware no ambiente *ROS*. Operacionalização e customização do firmware embarcado no hardware de acordo com as necessidades do projeto e da alimentação fornecida pela placa multiplexadora, por meio de baterias Li-Ion NH2054 14.4 volts.

3.4.4.3 Premissas Necessárias

Para o correto funcionamento desse módulo de *Power Management*, devem ser consideradas as seguintes premissas:

- A placa multiplexadora estará conectada diretamente ao módulo de *Power Management*
- Todos os dispositivos estarão conectados nas suas respectivas entradas
- A placa deverá ser alimentada por 2 baterias de 14.4 Volts e 3 Amperes, totalizando um fornecimento de até 6 Amperes
- A placa estará conectada diretamente na NUC, por meio de uma USB

3.4.4.4 Descrição da Funcionalidade

A funcionalidade *Power Management* é responsável fornece diversos recursos em sua totalidade. O hardware utilizado (placa Zord) possui um sensor de corrente e tensão para cada porta de saída, permitindo o monitoramento individual de cada uma das portas.

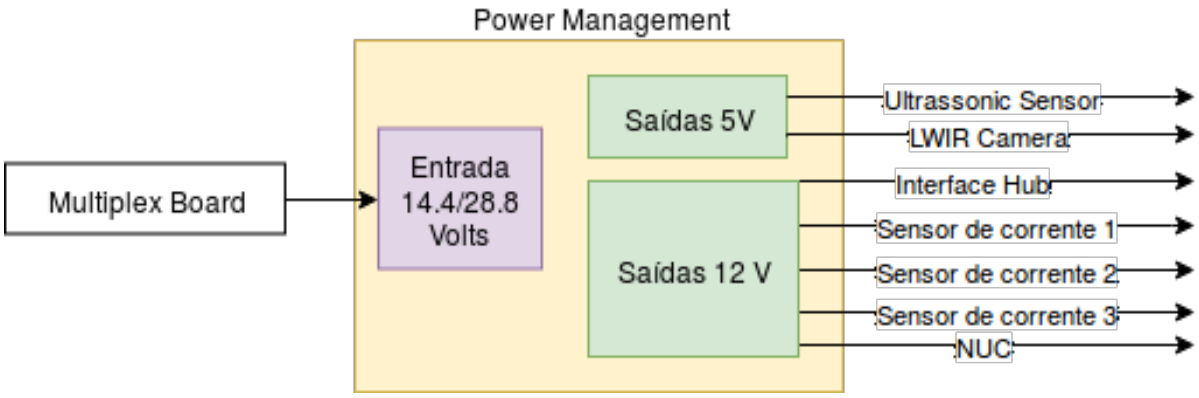

Figura 3.11: Fluxograma de funcionamento da funcionalidade de Power Management

Fonte: Própria

O microcontrolador utilizado Atmega32U4 possui um firmware embarcado onde toda a compatibilização com o ambiente *ROS* é realizada, o que torna essencial o uso do pacote rosserial para o seu funcionamento. O firmware é responsável pela ativação dos relés digitais em caso de surtos de corrente para proteção dos dispositivos elétricos. Os limites nos valores de corrente funcionam justamente para que o hardware interrompa a alimentação em um possível caso de surto de corrente. Todos os aspectos importantes para o funcionamento do sistema de gerenciamento de energia pode ser configurado tanto via *ROS*, por meio das configurações dos serviços, ou por meio do firmware, modificando os parâmetros do tempo de duração dos picos de corrente. Os principais serviços e tópicos criados pela funcionalidade Power Management no *ROS* são:

- *Tópicos*
  - *PowerOutput* Este tópico disponibiliza os valores de tensão e corrente de todas as portas da placa em tempo real.
  - *TakeStatus* Disponibiliza o estado de cada porta da placa, informando os eventos ocorridos e a porcentagem de corrente demandada durante a ocorrência do evento.
- *Serviços*
  - *GetCurrentLimitCommand* Este comando retorna o valor de corrente máxima de saída configurado para a porta escolhida
  - *SetCurrentLimitCommand* Este comando realiza a configuração do valor máximo de corrente de saída em uma determinada porta
  - *PowerOnOffCommand* Este comando realiza a ação de ativação ou desligamento de uma determinada porta.

A placa de Gerenciamento de energia irá receber a carga das baterias pela placa multiplexadora e irá realiza o controle de alimentação dos seguintes componentes:

- Grupos de servo motores
- Grupo de sensores de corrente
- NUC
- Interface HUB
- Câmera LWIR
- Sensor ultrassônico
- Phidgets
- STM Nucleo
- Módulo GPS

3.4.4.5 Saídas

A funcionalidade irá disponibilizar a energia para o robô e as seguintes estruturas no ambiente *ROS*:

- Tópicos com informações de tensão e corrente nas portas
- Tópico para aviso de sobre-corrente
- Tópico para informar disponibilidade da placa
- Serviços para ler e configurar limite de corrente das portas
- Serviço para ligar ou desligar energia em uma porta

3.4.5 System Integrity Check

3.4.5.1 Definição da funcionalidade

É a funcionalidade responsável por checar a integridade do sistema antes do início da missão, verificando os subsistemas e suas variáveis.

3.4.5.2 Dependências

A funcionalidade receberá informações dos seguintes componentes

- Sensor de Temperatura
- Servomotores
- Câmera IR
- Câmera Stéreo
- IMU
- Sensor de Proximidade
- Placa de Power Management
- Sonar
- Baterias

Todas as informações serão enviadas por meio do ambiente *ROS*, na forma de *Services* ou *Publishers*.

3.4.5.3 Premissas Necessárias

As premissas necessárias para o funcionamento dessa funcionalidade são:

- Os subsistemas do robô irão disponibilizar o seu status no ambiente *ROS* por meio de tópicos ou serviços
- A checagem fará parte do planejamento de missão

3.4.5.4 Descrição da Funcionalidade

A checagem da integridade do sistema é uma funcionalidade essencial para garantir o sucesso da missão e preservar a integridade do robô. O *ROS* facilita essa comunicação entre os subsistemas, possibilitando que seja criada uma rotina de checagem antes de cada missão.

Será disponibilizado no sistema uma rotina para iniciar a missão. Ao receber o comando para início de missão, os sistemas serão checados sequencialmente, utilizando estrutura de *Services* e *Publishers* do *ROS*. Caso algum sistema apresente falha, a missão não se iniciará e o erro será mostrado no *terminal* e registrado no arquivo de `log`. Se todos os sistemas estiverem em funcionamento, se iniciará a missão. O fluxograma da funcionalidade está ilustrado na figura 3.12.

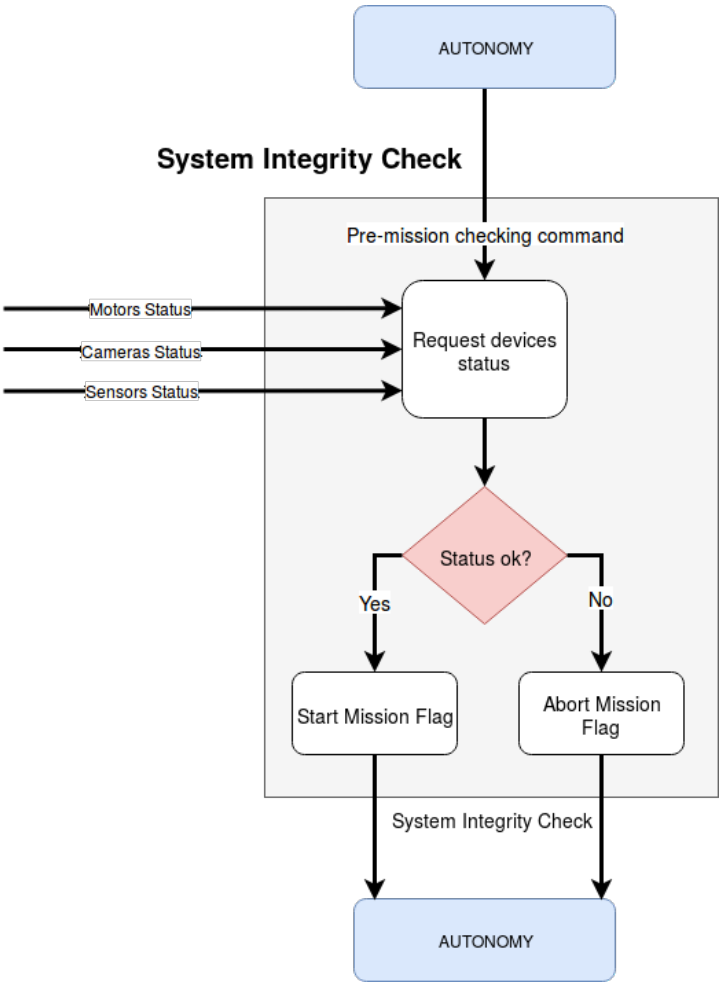

Figura 3.12: Fluxograma da rotina para checagem do sistema

Fonte: Própria

3.4.5.5 Saídas

No início da rotina de inspeção, a funcionalidade será responsável por enviar o sinal inicia a missão. Caso todos os sistemas checados estejam funcionando, a inspeção ocorrerá normalmente, se algum sistema apresentar defeitos, o defeito será mostrado no *terminal*, registrado em log e a missão será abortada.

3.5 Simulação do sistema

A simulação de sistemas robóticos consistem em um dos pilares para o desenvolvimento de projetos. Com a simulação é possível testar aplicações sem a necessidade de adquirir componentes, os membros da equipe de projeto conseguem trabalhar de forma simultânea no robô enquanto o protótipo físico fica reservado para testes específicos.

O *ROS* oferece ferramentas de visualização já integradas no seu sistema, o RVIz, que possibilita o usuário visualizar os modelos do robô e também administrar plugins, como de mapeamento e planejamento de movimento, que é o caso do *MoveIt!*.

Para a simulação do robô no ambiente aberto, é utilizado o software *Gazebo*. A integração entre *ROS* e *Gazebo* consegue fazer com que o modelo *URDF*, por mais que não seja o nativo do *Gazebo*, seja aceito na simulação. Parâmetros do mundo podem ser ajustados e a integração de plugins como câmeras e sensores faz com que a simulação consiga ser utilizada em diferentes estudos. Algoritmos de imagem podem ser testados com os plugins de câmera já implementados, proporcionando um auxílio para demonstrar conceitos e teorias de funcionamentos

A simulação fornecida possui os controladores de juntas já implementados, fazendo com que testes de códigos de movimentação e testes de controles já pudessem ser previamente testados, poupando riscos de dano ao protótipo e possibilitando trabalho simultâneo.

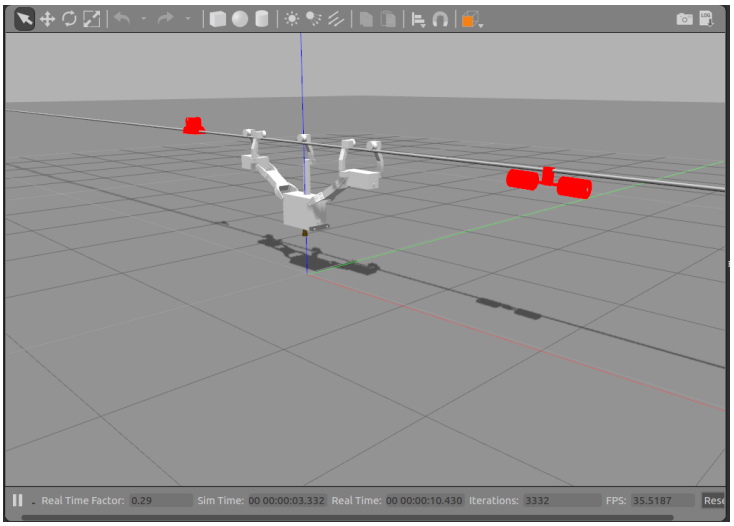

Figura 3.13: Simulação do *ELIR* no *Gazebo*.

Capítulo Quatro

Resultados

A elaboração de cronogramas, e a organização das atividades em pacotes com metas de curto médio e longo prazo contribuiu para que o andamento do projeto ocorresse de forma satisfatória. Os prazos foram mudados durante o projeto e os atrasos gerados por problemas inesperados conseguiram ser compensados com uma mudança na forma de gerir os integrantes e a inserção de tarefas em paralelo, sendo elas individuais e em subgrupos.

4.1 Testes unitários

Consistem nos testes individuais das ferramentas, tem suma importância para atestar se as ferramentas pré determinadas funcionam de acordo com o esperado.

4.1.1 Configuração dos servomotores

Os primeiros testes realizados no início do desenvolvimento do robô foram a configuração dos servomotores, etapa essencial para a definição de como os motores deveriam trabalhar em conjunto. A configuração se deu utilizando diversos softwares, e após as configurações realizadas, os motores eram analisados para validar se a configuração fora realizada de maneira correta. O teste foi realizado no período de 22 de junho a 7 de julho pelos componentes: Carlos, Cleber, Davi e Ícaro.

4.1.1.1 Utilização do software Mixcell

O software Mixcell disponível para utilização gratuita na Ros Wiki, possui a funcionalidade de realizar a configuração de parâmetros em motores Dynamixel de diversos modelos, tais como ID dos motores, modos de operação, baudrate para comunicação serial. O software foi utilizado em toda as etapas de testes dos motores, sendo fundamental para determinar o funcionamento de cada um dos motores e para o desenvolvimento e montagem do robô.

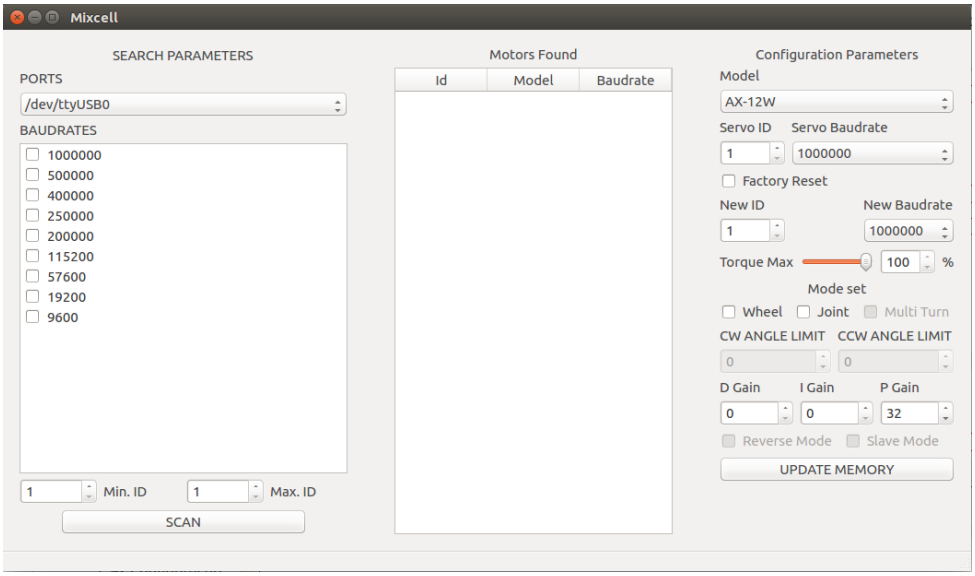

Figura 4.1: Programa para configuração dos servomotores

Fonte: Própria

4.1.1.2 Modos Mestre/Escravo

Durante os testes em modo Mestre/Escravo dos motores Dynamixel, foi observado que mesmo quando dois motores são configurados para trabalharem como mestre/escravo via software, é necessário que se haja a utilização do cabo de sincronização entre os dois motores. Foi constatado durante os testes que o cabo de sincronização transmite a informação de carga para que haja a compensação entre os motores quando trabalham em conjunto.

4.1.1.3 ID, Baud rate e Modos de operação

Durante toda a fase de testes foram observados parâmetros extremamente importantes para definição e funcionamento dos motores Dynamixel. Os IDs dos motores definem basicamente a sua identificação para as configurações de juntas, associando assim os motores as suas respectivas juntas. O baud rate é a taxa de transmissão de dados em bits por segundo entre o computador e os motores. Percebeu-se que durante os testes, para um número grande de motores trabalhando em conjunto, era necessário um valor de baudrate mais elevado para que todos os motores fossem encontrados. Os modos de operação dos motores são parâmetros que definem como os mesmos vão se comportar durante a operação, sendo eles os modos junta e roda. No primeiro modo é necessário estabelecer um limite máximo e mínimo para o giro dos motores, além da definição de qual motor será o mestre ou o escravo durante a operação (caso haja mais de um motor na junta), enquanto que no modo roda o motor irá girar livremente, mudando somente o sentido de

giro.

4.1.2 Controle dos servomotores utilizando a biblioteca dynamixel driver

A biblioteca dynamixel driver foi a utilizada nesse projeto e tal, para fazer a configurações , criar os controladores, possibilitar a integração do motor no sistema ros. Essa biblioteca foi descontinuada na versão utilizada no projeto, o que fez com se mostrassem necessários testes extras. Os testes foram realizado entre 7 de julho até 21 de julho por Carlos, Cleber, Davi e Ícaro, durante a fase inicial do projeto.

4.1.2.1 Controladores de posição

Foi realizado o teste dos motores de posição para verificar o comportamento dos mesmos. Nas juntas dos braços do robô foi utilizado o modo de controlador de posição, isto é: o motor recebe um comando de posição, que no caso é de 0 a 2PI radianos e assim é possível controlar para onde o braço irá se mover.

O controlador recebe um arquivo de configuração das juntas, especificando os ids e assim pode-se controlar cada um individualmente. Foi realizado o teste em todas as juntas de posição do ELIR.

4.1.2.2 Controladores de velocidade

Os controladores individuais de velocidade tem a função de controlar a velocidade do motor, sem controlar a posição. O motor irá rodar livremente com uma velocidade determinada até que receba um comando para parar. Este controlador foi usado nas rodas das garras que possuem a finalidade de fazer o robô andar sobre a linha. Cada roda fora testada individualmente.

4.1.2.3 Controlador de trajetoria

O controlador de trajetória controller realiza o controle do posicionamento da junta como um todo. Neste controlador é possível controlar diversas juntas diferentes do robô por meio de uma única mensagem no *ROS*, contendo o nome das juntas que deverão ser

controladas, as posições desejadas, tempos de execução, velocidades e esforço. Esse controlador se mostrou extremamente importante e eficiente pela sua capacidade de agrupar diversas juntas em um único comando de movimento.

4.1.2.4 Compatibilidade com diferentes protocolos de comunicação

Há uma limitação na biblioteca dynamixel driver, esta funciona somente para motores com firmware versão 1.0 devido a sua descontinuação. Foi testado para todas as versões contidas nos motores que possuíam e não houveram falhas em encontrar nenhum motor na biblioteca. Entretanto, caso houvesse necessidade de atualizar para versão v2.0, no qual possui uma taxa de baud rate maior e protocolo de comunicação melhorado, haveria a necessidade de parar com o uso a dynamixel driver e utilizar a biblioteca oficial da fabricante, a dynamixel workbench.

4.1.3 Controle dos servomotores utilizando a biblioteca dynamixel workbench

A biblioteca do *ROS* dynamixel workbench é outro driver para comunicação com os servomotores, recebe suporte e atualizações diretamente da ROBOTIS, sendo o driver de controle padrão do *ROS* para versões superiores a Kinetc. Considerou se utilizar essa ferramenta devido aos problemas de comunicação apresentados pelos motores. Ela oferece suporte para o protocolo de comunicação na versão 2.0, que suporta uma maior baudrate e muda a estrutura de comunicação.

4.1.3.1 Controladores de posição

A biblioteca oferece a opção de criar controladores de posição individuais para cada motor, sem especificar o ID. Não possibilitando uma criação controlador mestre-escravo para as juntas do robô que são atuadas por 2 motores e também a opção de selecionar quais motores da rede seriam transformados em controladores de posição, o arquivo que instancia os controladores só recebe um alcance de IDs que deve buscar na rede, transformando todos os motores nesse alcance em controladores de posição individuais.

4.1.3.2 Controladores de velocidade

Os controladores de velocidade só recebem o ID de dois motores, assim não apresentando um uso trivial, já que o robô necessita de 5 controladores de velocidade diferentes. Foi encontrado um arquivo customizado no repositório da biblioteca, mas mesmo assim não pode ser utilizado em conjunto com controladores de posição.

4.1.3.3 Controladores de trajetória

Os controladores de trajetória esperados para compatibilização com o planejamento de movimento são os padrões oferecidos pela biblioteca embutida do *ROS*, o `ros_control`. Porém a biblioteca não oferece um driver que instancie esses controladores de trajetória, assim não possibilitando a integração dela com o planejamento de movimento.

4.1.3.4 Comunicação em rede

Em comparação com a biblioteca `dynamixel_drivers`, a comunicação em rede dos servomotores utilizando a biblioteca `dynamixel_workbench` é levemente superior. Quando a comunicação apresentava problemas, essa biblioteca se mostrou mais eficiente para encontrar os motores conectados na rede porém não possibilita o funcionamento completo, já que o *MoveIt!* controla o *ROS* por meio dos controladores de trajetória, assim não se mostrando uma opção para resolução dos problemas de comunicação com os motores. Essa biblioteca oferece suporte para motores com protocolo 1.0 e 2.0 ,possibilitando criar uma rede com motores dos dois tipos, porém apresenta erros caso os motores tenham o mesmo ID, mas não foram feitos testes de controle para motores com dois protocolos diferentes.

4.1.4 Placa de Power Management

A placa de power management é responsável pelo gerenciamento de energia do robô, ela distribui de forma separada a potência para cada parte elétrica ou eletrônica, e pela sua capacidade de monitoramento, pode identificar e atuar sob algum erro na alimentação. Diversos testes foram feitos para garantir que a mesma estivesse trabalhando de maneira correta.

4.1.4.1 Gravação e validação do firmware

A gravação do firmware na placa de Power Management só foi realizada devido a utilização de um botão para conectar dois pinos específicos da placa (-VIN e S2) para que o microcontrolador fosse alimentado, e a conexão da placa em uma porta USB de um computador.

Para que fosse possível estabelecer a comunicação entre o computador e a placa, foi necessário configurar as permissões necessárias no Ubuntu. Durante a gravação do firmware fora utilizada uma extensão do VSCode, o PlatformIO, específica para programação com microcontroladores, para realização de debug e gravação do firmware na placa.

Durante o processo de gravação foi necessário manter o botão pressionado. Após a gravação do firmware na placa, o seu funcionamento fora validado através do uso dos serviços e mensagens gerados no ambiente *ROS*, uma vez que as informações de níveis de corrente e tensão das portas da placa eram publicados nos tópicos específicos. O teste foi coordenado por Ícaro Nascimento no dia 1 de agosto de 2018.

4.1.5 Compatibilidade do Robô com o MoveIt!

Testes realizados no período de 24/05 até 26/06, pelos integrantes Cleber Couto e Davi Oliveira. o teste consiste na criação do pacote *MoveIT!*. O *MoveIt!* fornece uma ferramenta de configuração denominada *MoveIt! Setup Assistant*, que gera a maior parte dos arquivos necessários para a compatibilização.

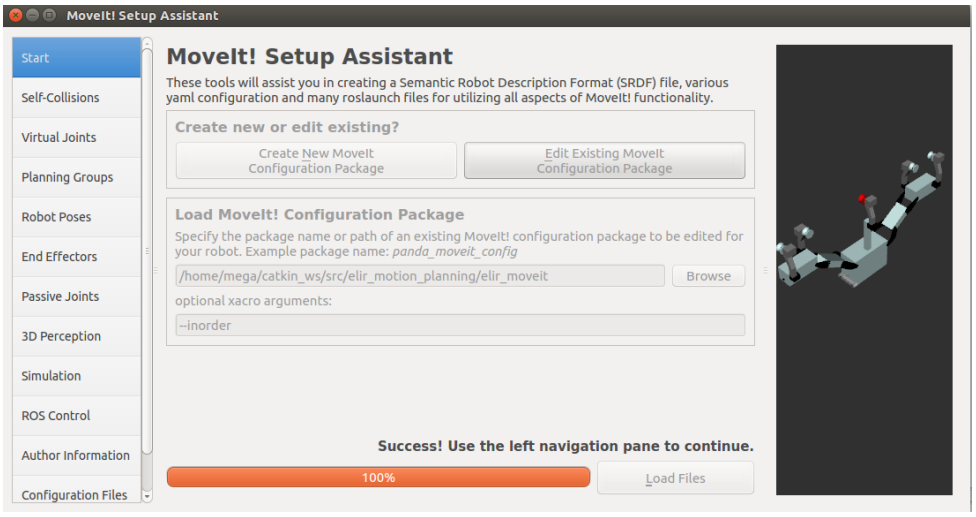

Figura 4.2: Ferramenta para geração de pacote fornecida pelo *MoveIt!*

Fonte: Própria

4.1.5.1 Estado de Colisão

Durante a configuração dos Setup Assistant , o robô apresentava um erro informando que o seu modelo estava em estado de colisão, o que significa que as partes do seu modelo estavam posicionadas erradas e estariam sempre colidindo.

Foi feito um teste removendo todas as partes do robô do arquivo de configuração e checando se a indicação de erro sumia. Com as tentativas se concluiu o problema era o modelo 3D utilizado nas rodas do robô, que gerava a colisão quando se utilizava ele mais de uma vez, a resolução foi a criação de um novo modelo 3D para as rodas, após a substituição o erro sumiu.

4.1.5.2 Definição da corrente-cinemática e end-effector

No estudo de cinemática inversa na robótica, corrente cinemática é o nome dado para o conjunto de links e juntas que se vai calcular o movimento, onde o começo dessa corrente é a referência para o cálculo da cinemática e o final da corrente é o end-effector. No caso do *ELIR*, é necessário se definir duas correntes cinemáticas, já que são dois braços. Com a corrente cinemática definida corretamente ,o *MoveIt!* consegue enviar o comando para movimentar as juntas da corrente simultâneamente.

Ao definir o grupo de controle no Setup Assistant, o usuário escolhe as juntas que irão fazer parte desse grupo, e a corrente cinemática é gerada automaticamente. Foram feitos teste considerando como parte da corrente somente as duas juntas do braço, porém ao analisar outros robôs que já haviam sido configurados, notou-se que sempre se levava em consideração a junta da base do robô para definição do grupo, onde a junta da base é a junta que prende o robô no mundo.

4.1.6 Robô de testes Victory

Como o Setup-Assistant oferece diversas opções de configuração, optou-se por desenvolver um modelo de robô mais simples que o *ELIR*, de forma que se pudessem testar os conceitos, e compreender melhor a relação entre as configurações do modelo e os resultados esperados para movimento. Esse teste foi executado pelos membros Cleber Couto e Davi Oliveira, no período de 6 a 23 de agosto onde esse modelo foi inteiramente escrito pela equipe, tomando como base boas práticas observadas em robôs já compatibilizados com o *MoveIt!*. O *Victory* foi construído como mostra a imagem 4.3:

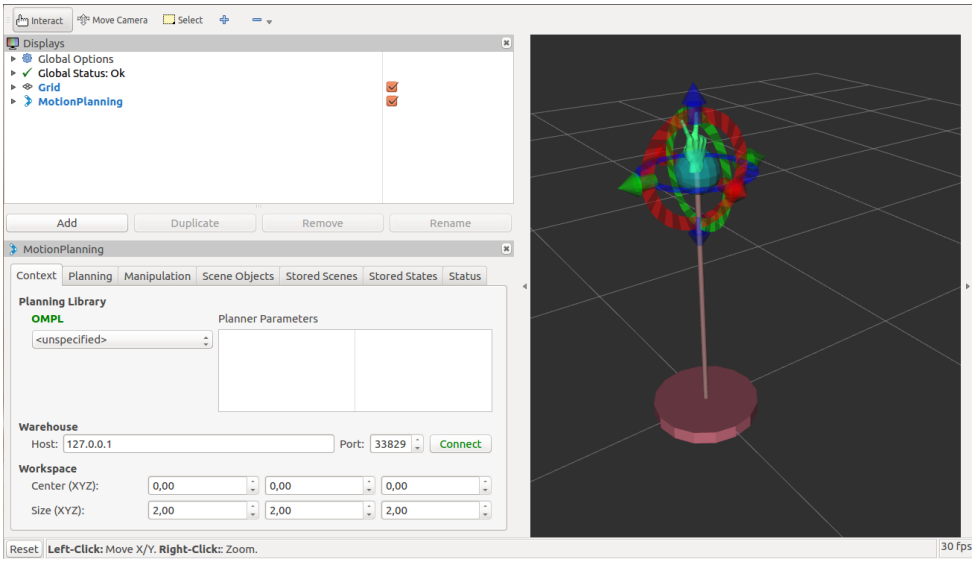

Figura 4.3: Victory Robot na ferramenta de visualização *Rviz*

Fonte: Própria

Antes de realizar qualquer modificação na modelagem do robô *ELIR* primeiramente essa modificação foi feita Davictory, assim foram prevenidos quaisquer mudanças desnecessárias e otimizado o tempo devido que o robô customizado ser mais simples. Todos os testes com o planejamento de movimento foram validados também nele.

4.1.6.1 Teste com diferentes plugins de cinemática

Para configurar o pacote do *MoveIT!* de qualquer modelo de robô, é necessário definir qual solucionador de cinemática será utilizado. Como existem variados tipos de robôs com finalidades e conjuntos de movimentos diferentes essa escolha deve ser feita de forma precisa para que o robô em questão realize o movimento de forma eficiente.

Durante a escolha para a cinemática do *ELIR* foi percebido que nenhuma solução se encaixava nas descrições devido ao número reduzido de graus de liberdades (o *ELIR* possui 2 graus de liberdades e os plugins necessitam de no mínimo 3).

A partir de plugins e o *URDF* é possível criar a solução específica para o robô. A fim de achar uma solução que atendesse ao funcionamento do *ELIR*, foram testados no Davictory todos os plugins de cinemática contidos no site oficial do *MoveIT!*. Nenhum plugin obteve sucesso em resolver a cinemática inversa.

4.2 Testes integrados

Após a realização dos testes unitários, validando os conceitos e funcionalidades individualmente, novos testes foram realizados para validação das funcionalidades em conjunto, na etapa de testes unitários.

4.2.1 Teste dos limites de giro das juntas

Este teste realizado por Carlos Alberto e Ícaro Nascimento entre os dias 2 a 24 de Agosto de 2018, teve como objetivo compatibilizar os limites de giro das juntas nos controladores da biblioteca `dynamixel_drivers` com os limites físicos do robô, a fim de evitar colisões. Os limites a princípio vieram do arquivo *URDF* utilizado no *MoveIt!*. Porém, algumas juntas ainda estavam colidindo, então os limites tiveram que ser reajustados manualmente.

4.2.2 Integração do MoveIt! com os controladores do ROS

Este teste foi feito por Cleber no período de 10 de agosto de 2018 até 11 de setembro de 2018 . Para a compatibilização do MoveIt! com os controladores do *ROS* é necessário que o programa conheça o estado atual do robô e os controladores de trajetória disponíveis, foi necessário desenvolver uma solução que transformasse as informações provenientes dos motores em uma estrutura específica que descreve o estado do robô, provida pelo *ROS*.

O mesmo se deu pelo tópico de informação dos motores, porém a informação de posição de cada motor não dá o valor de ângulo de giro, mas sim o valor escrito no registrador do Dynamixel e também não leva em consideração a referência em que cada motor foi configurado, assim sendo necessário uma compensação de referência em código e uma conversão para radianos, assim como a inserção na estrutura padrão do *ROS* para estado atual do robô.

4.2.3 Serviço para levantar a garra

Para realizar a transposição de objetos na linha o robô precisa realizar movimentos específicos. Um desses movimentos se dá em levantar as garras para que as mesmas fiquem livres para fazer a abertura. Esse movimento de levantar foi feito em forma de serviço no *ROS*, no qual basta somente chamar e solicitar a levantamento e abaixamento das

garras. O teste foi feito primeiramente no Gazebo e quando ocorreu de forma esperada, foi iniciado o teste no robô físico. A simulação e atuação do teste foi feito pelo integrante Davi Oliveira na data de 7 de novembro de 2018 até 9 de novembro de 2018.

4.2.4 Serviço de abrir e fechar as garras

Foi testado o serviço desenvolvido para realizar a abertura e fechamento das garras, o movimento de abrir é realizado para que o obstáculo passe livremente pelo centro e ao finalizar as garras são fechadas. Observou se que o movimento devia levar em consideração o choque entre as garras, admitindo assim somente um sentido. A resolução se deu por trocar o sinal das coordenadas e observar o o comportamento, assim encontrando a melhor opção. O teste foi feito por Cleber Couto na data de 12 de outubro de 2018.

4.2.5 Serviço para levantar haste central na linha

Para a ultrapassagem de um tipo de obstáculo, a melhor forma encontrada foi de aumentar a distância entre as duas unidades de tração, fazendo com que a haste central se levante, e o obstáculo seja atravessado, foi desenvolvido um serviço para que o robô executasse esse movimento. O teste do serviço fora realizado para funcionamento via simulação em Gazebo. Foi necessário realizar a verificação da posição das juntas para que o robô estivesse na posição esticada e a após este procedimento um ajuste foi realizado. A simulação foi realizada pelo integrante Ícaro Nascimento no dia 16 de novembro de 2018.

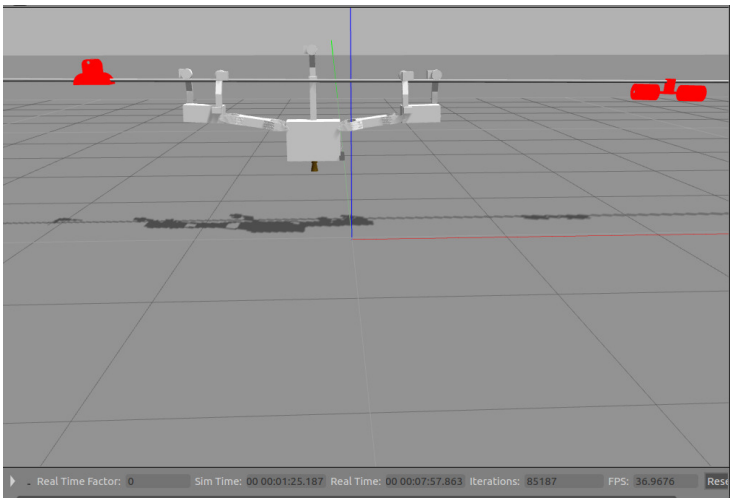

Figura 4.4: *ELIR* realizando o movimento na simulação

Fonte: Própria

4.2.6 Robô na linha sem alimentação

Quando toda a estrutura mecânica do robô estava completamente montada, o mesmo foi colocado sobre uma linha de testes localizada no estacionamento do Senai Cimatec. O teste foi coordenado por Cléber, Marco e Ícaro no dia 10 de setembro de 2018, e consistiu em posicionar o robô na linha de transmissão para verificar o alinhamento das garras e a distribuição do peso. Foi observado que havia um desalinhamento entre as rodas das garras dos motores. Após a observação deste erro as garras foram reposicionadas durante uma nova montagem do robô, resolvendo o problema do desalinhamento.

4.2.7 Robô na linha utilizando alimentação alternativa

No primeiro teste na linha de transmissão, que ocorreu no dia 19 de outubro de 2018, foi montado um setup com uma bateria automotiva 12V - 70Ah para alimentar todos os componentes do robô. A NUC foi alimentada com sua fonte conectada a um inversor que por sua vez estava ligado à bateria. Nesse teste não houve sucesso na comunicação com os motores, a bateria nesse momento fornecia 12,17V e era a única modificação em relação aos testes no laboratório. Com a fonte de bancada, nenhum problema de comunicação aconteceu. O teste foi conduzido por Carlos, Cleber, Davi e Ícaro.

4.2.8 Alimentação do sistema utilizando Nobreak

Para contornar o problema encontrado no primeiro teste na linha de transmissão, a alimentação seria fornecida pela mesma fonte de bancada dos testes unitários e esta seria ligada a um nobreak. Com o nobreak ligado na tomada a fonte foi ajustada em 14V, no momento em que a tomada era desconectada, a fonte passava a fornecer 10V variando por volta de 200mV. Teste realizado no dia 22 de outubro por Carlos.

4.2.9 Teleoperação para movimentação na linha

Para conseguir mover o robô na linha e observar o seu comportamento, era necessário desenvolver uma forma de o operador controlar a velocidade das rodas de forma intuitiva. O teste foi conduzido por Cleber e Carlos, no período de: 12 a 24 de outubro.O *ROS* fornece um driver de teleoperação via teclado, foi criado um nó que faz a interface desse pacote com os controladores do robô, possibilitando assim testar o deslocamento horizontal na linha. Foi observado que os motores não param de girar assim que a tecla para de ser

pressionada, devido a velocidade que o nó do teclado atualiza os comandos.

4.2.10 Alimentação de todos os motores pela placa de Power Management

Após a validação do funcionamento da placa de gerenciamento de energia, foi necessário verificar se o seu fornecimento de energia seria suficiente para alimentar toda a rede de motores para que pudessem operar em conjunto. O teste conduzido por Davi e Ícaro no dia 31 de outubro, consistiu em conectar todos os três hubs dos motores nas portas da Power Management, monitorando o seu funcionamento. Nos testes iniciais fora percebido que os picos de corrente dos motores faziam com que a alimentação nas portas fossem interrompidas. Tendo em vista a ocorrência desse problema foi necessário ajustar os limites de corrente via serviço em *ROS* da placa para que a alimentação não fosse interrompida durante a operação.

4.2.11 Comunicação dos motores em rede

Os dynamixel se comunicam serialmente por meio do padrão físico RS485, que possui uma boa imunidade a ruídos, distância de comunicação e número de receptores maiores do que foi usado. O teste constituiu colocar em paralelo todos os motores por meio dos hubs construídos e assim verificar se o *ROS* o encontram. Foi feita a instalação dos motores na estrutura do robô, realizando as conexões com os cabos de alimentação e comunicação padrão junto com os HUBs desenvolvidos para cada grupo de motores. O driver para os motores fornece a função de busca em rede, o que foi suficiente nesse teste para garantir que os motores estavam se comunicando.

Durante o teste dos motores em rede foi encontrado o problema de achar apenas alguns motores de forma aleatória, dos 18 conectados. Foi feita uma série de testes em relação ao firmware para verificar se o problema era na comunicação. A troca, upgrade, downgrade e restauração foram feitos utilizando o software *ROBOPLUS* da própria fabricante dos motores. Os testes foram feitos por Davi Oliveira e Carlos Alberto nos dias 29 de setembro de 2018 até 5 de outubro de 2018.

4.2.12 Restauração para firmware padrão de fábrica

O firmware de cada motor foi restaurado para o padrão de fábrica e realizando a busca na rede individual. Os motores estavam funcionando porém ao serem conectados todos ao mesmo tempo, o erro de comunicação persistia.

4.2.13 Compatibilização das versões de firmware

A partir da busca de motores que sofriam problemas parecidos, foi encontrado fórum oficial da *Robotis*, que a versão do mais estável era a versão v1.0 -36 para os motores que foram utilizados. Foi verificado que a maioria dos motores estavam na versão v1.0 - 36 e somentes alguns estavam na v1.0 - 37. Foi feita a troca do que constam v1.0 -37 para v1.0 -36 e ainda assim o problema se mostrou consistente.

4.2.14 Atualização para versão de firmware v2.0

Foi testado a troca de firmware para versão v2.0 no qual possui evolução no protocolo de comunicação assim como uma maior baudrate, houve melhora em relação à quantidade de motores encontrados mas ainda assim o problema se mostrou ativo.

4.2.15 Troca dos motores danificados

Como os testes anteriores não apresentaram solução, foi realizado um teste onde os motores eram inseridos na rede de um a um, onde a comunicação era verificada. Quando a comunicação começava a apresentar falhas, esse motor era retirado da rede, assim foram encontrados 2 motores que inseriam esse erro de comunicação na linha. Os motores funcionavam indivualmente e quando conectados com uma pequena quantidade de motores. O problema da falta de comunicação foi sanado após a troca dos motores defeituosos.

4.3 Avaliação da prontidão tecnológica

asdfadsfsdfs

4.4 Trabalhos futuros

A implementação de um robô para inspeção de linha, faz possível que sejam implementadas diferentes tipos de tecnologias, já que a sua estrutura pode ser projetada para receber uma série de implementações. Um drone por exemplo, consegue realizar uma manutenção visual efetiva, sem muito gasto energético, mas seu tamanho reduzido faz com que algumas tecnologias não consigam ser implantadas.

Os resultados dos testes possibilitam o início de novos estudos e mostra comportamentos específicos de cada componente que tem que ser levados em consideração. Os servomotores apresentaram um melhor desempenho quando alimentados com 14.4V, conseguindo realizar movimentos que necessitavam de mais torque de forma efetiva, enquanto com 12V o movimento não acontecia de forma consistente. Hoje a ROBOTIS desenvolve modelos de servomotores que possuem um torque máximo muito elevado, assim se torna viável a troca dos motores nas juntas que realizam mais esforço, assim como o estudo para aumento da tensão de alimentação ou um estudo dirigido para encontrar o melhor valor da curva torque e tensão de alimentação.

O *MoveIt*! possibilita a integração com ferramentas de visualização, assim tornando viável a futura implementação de um sistema de controle baseado em percepção 3D e odometria visual. O uso de algoritmos de detecção de obstáculos precisos, torna possível a ultrapassagem automática de obstáculos, sendo possível inserir o obstáculo no ambiente do robô e realizar o desvio automaticamente.

Capítulo Cinco

Conclusão

O resultado do projeto alcançou as expectativas. Os problemas que aconteceram conseguiram ser contornados e o tempo gasto para sua solução conseguiu se adequar ao esperado pelo cronograma das tarefas. O material produzido atende às demandas do cliente e os pacotes produzidos foram organizados buscando facilitar o uso por terceiros.

5.1 Considerações finais

A gestão do projeto do robô como um todo, fez com que o resultado produzido alcançasse as expectativas. O nível de desenvolvimento aumentou progressivamente de forma que o projeto foi conduzido, adicionando paulatinamente diversos conhecimentos específicos que não seriam vistos normalmentes durante a graduação, mas também fortalecendo conhecimentos já formados.

O que foi produzido para o projeto estará disponível para futuras consultas, o que impulsiona o desenvolvimento de projetos semelhantes. Novos estudos podem ser iniciados como trabalhos de graduação, ou pós-graduação, realizando provas de conceito e abrindo oportunidades para novas tecnologias.

O início do projeto se deu de forma lenta, por apresentar uma área do conhecimento nova para maior parte da equipe. A robótica necessita da integração de diversas áreas diferentes, e para a engenharia elétrica, o conhecimento de diversas camadas de abstração. Com a experiência e o desenvolver das atividades, os conhecimentos adquiridos possibilitaram atividades em paralelo e o aumento da versatilidade dos integrantes.

A experiência como um todo foi muito enriquecedora, adicionando conhecimentos que serão necessários no futuro e proporcionando um crescimento para todos os participantes.

Apêndice A

QFD

Apêndice B

Diagramas mecânicos

Apêndice C

Diagramas eletro-eletrônicos

Apêndice D

Wireframes

Apêndice E

Logbook