

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA E DE
AUTOMÓVEL**

VILSON WENIS DOS SANTOS BELLE

**DESENVOLVIMENTO DE UM AMBIENTE VIRTUAL DE SIMULAÇÃO PARA
VEÍCULOS TERRESTRES**

**RIO DE JANEIRO
2022**

CIP - Catalogação na Publicação

Dos Santos Belle, Vilson Wenis
Desenvolvimento de um Ambiente Virtual de
Simulação para Veículos Terrestres / Vilson Wenis
Dos Santos Belle. – Rio de Janeiro, 2022.
110 f.

Orientador: Diogo Lopes Fernandes.

Projeto Final de Curso (graduação) – Instituto
Militar de Engenharia, Bacharel em Engenharia
Mecânica e de Automóveis, 2022.

1. ROS. 2. Veículos Autônomos. 3. Base Móvel. 4.
Gazebo. I. Fernandes, Diogo Lopes, orient. II.
Título

VILSON WENIS DOS SANTOS BELLE

DESENVOLVIMENTO DE UM AMBIENTE VIRTUAL DE SIMULAÇÃO
PARA VEÍCULOS TERRESTRES

Projeto Final de Curso realizado no Curso de Graduação em Engenharia Mecânica e de Automóvel do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia Mecânica e de Automóvel.

Orientador(es): Diogo Lopes Fernandes, M.Sc.

Rio de Janeiro

2022

©2022

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmар ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

Belle, Vilson Wenis dos Santos.

DESENVOLVIMENTO DE UM AMBIENTE VIRTUAL DE SIMULAÇÃO
PARA VEÍCULOS TERRESTRES / Vilson Wenis dos Santos Belle. – Rio de
Janeiro, 2022.

109 f.

Orientador(es): Diogo Lopes Fernandes.

Projeto Final de Curso (graduação) – Instituto Militar de Engenharia, Engenharia
Mecânica e de Automóvel, 2022.

1. ROS. 2. Rôbos autônomos. 3. Base Móvel. 4. Gazebo. i. Fernandes,
Diogo Lopes (orient.) ii. Título

VILSON WENIS DOS SANTOS BELLE

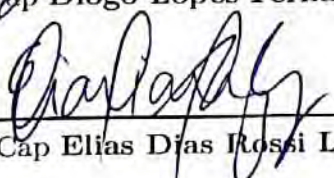
**DESENVOLVIMENTO DE UM AMBIENTE VIRTUAL
DE SIMULAÇÃO PARA VEÍCULOS TERRESTRES**

Projeto Final de Curso realizado no Curso de Graduação em Engenharia Mecânica e de Automóvel do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia Mecânica e de Automóvel.

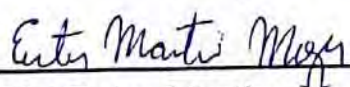
Relatório aprovado em Rio de Janeiro, 10 de outubro de 2022.



Cap Diogo Lopes Fernandes - M.Sc. do IME



Cap Elias Dias Rossi Lopes - D.Sc do IME



Maj Euter Martins Mozer - M.Sc do IME

Rio de Janeiro

2022

Sapere aude.

RESUMO

A robotização do meios de combate vem se tornando imprescindível nos campos de batalha, e por mais que veículos aéreos não tripulados já sejam uma realidade, o mesmo não pode ser dito dos veículos terrestres, tanto no que diz respeito a operação à distância quanto à navegação autônoma. Nesse contexto, o presente trabalho tem como finalidade a utilização de um ambiente de simulação baseado em ROS que permita o teste e o desenvolvimento de algoritmos que podem ser úteis em aplicações militares. Para tanto, esse estudo realiza a modelagem de um veículo de pequenas proporções no qual é montado um manipulador de quatro graus de liberdade, com o objetivo de auxiliar na detonação de explosivos, implementando tanto funcionalidades de controle do manipulador quanto de deslocamento da base móvel. Além disso, é desenvolvido um sistema de controle robusto para as grandezas de velocidade linear e angular do veículo. Esse sistema é posteriormente discretizado e testado em uma malha conjunta de um software comercial com o Hardware que se deseja utilizar na implementação. Posteriormente, o veículo é construído com componentes de baixo custo e os sistemas desenvolvidos na simulação são testados no veículo real.

Palavras-chave: ROS. Rôbos autônomos. Base Móvel. Gazebo.

ABSTRACT

The robotization of the means of combat has become essential on the battlefields, and while unmanned aerial vehicles are already a reality, the same cannot be said of land vehicles, both in terms of remote operation and autonomous navigation. In this context, the present work aims to use a simulation environment based on ROS that allows the testing and development of algorithms that can be useful in military applications. Therefore, this study performs the modeling of a small vehicle in which a four-degree-of-freedom manipulator is mounted, in order to assist in the detonation of explosives, implementing both manipulator control and mobile base displacement functionalities. In addition, a robust control system is developed for the linear and angular velocity variables of the vehicle. This system is later discretized and tested in a joint mesh of commercial software with the Hardware that is intended to be used in the implementation. Subsequently, the vehicle is built with low-cost components and the systems developed in the simulation are tested in the real vehicle.

Keywords: ROS. autonomous robot. mobile base. Gazebo.

LISTA DE ILUSTRAÇÕES

Figura 1 – Estrutura do sistema ROS (1)	20
Figura 2 – Exemplo do funcionamento do sistema ROS (2)	21
Figura 3 – Organização do pacote ROS (1)	21
Figura 4 – Visualizador Rviz	22
Figura 5 – Algoritmo geral para filtros de Bayes (3)	26
Figura 6 – Modelo de deslocamento via odômetro	27
Figura 7 – Distribuição de probabilidade da leitura do sensor	28
Figura 8 – Algoritmo geral para filtro de Kalman (3)	30
Figura 9 – Algoritmo geral para filtro de Kalman estendido (3)	31
Figura 10 – Rover modelo Lynxmotion 4WD1 (imagem de www.lynxmotion.com).	32
Figura 11 – Base móvel no ambiente Gazebo	33
Figura 12 – Juntas de transmissão de velocidade	33
Figura 13 – Contato representado pelas bolas azuis	34
Figura 14 – Modelo de atrito cônico (4)	34
Figura 15 – Manipulador <i>OpenManipulator-X</i> (5)	35
Figura 16 – Modelo do manipulador em uma software CAD com os eixos de rotação (5)	36
Figura 17 – Base e manipulador acoplados	36
Figura 18 – Modelo de câmera acoplada na parte superior da garra	38
Figura 19 – Modelo de câmera acoplada na base do veículo	39
Figura 20 – Modelo após a integração dos sensores a laser	39
Figura 21 – Relação entre o referencial estático e a base do robô	40
Figura 22 – Relação entre o referencial da base e as rodas	40
Figura 23 – Relação entre o referencial da base e sensores e junta do manipulador	40
Figura 24 – Relações entre os diversos corpos do manipulador e do sensor laser frontal	41
Figura 25 – Relações entre a garra e os corpos definidos a partir dela	41
Figura 26 – Exemplo do mapa gerado pelo método <i>Gmapping</i> (6)	42
Figura 27 – Funcionamento do nó <i>move base</i> (7)	43
Figura 28 – Mapa de custo criado pelo nó <i>move_base</i>	43
Figura 29 – Exemplo de planejamento global de trajetória (8)	44
Figura 30 – Ambiente interno no qual são realizadas as simulações	45
Figura 31 – Ambiente externo no qual são realizadas as simulações	45
Figura 32 – Estado inicial de simulação em ambiente interno	46
Figura 33 – Nós ativos e tópicos de comunicação entre eles	47
Figura 34 – Resposta a uma entrada degrau de velocidade longitudinal	47
Figura 35 – Resposta a uma entrada degrau de velocidade longitudinal	47

Figura 36 – Resposta a uma entrada degrau de velocidade longitudinal com o modo turbo acionado	48
Figura 37 – Resposta a uma entrada degrau de velocidade longitudinal com o modo turbo acionado	48
Figura 38 – Resposta a uma entrada degrau de velocidade angular	49
Figura 39 – Resposta a uma entrada degrau de velocidade angular	49
Figura 40 – Janela inicial para o controle do manipulador	50
Figura 41 – Posição inicial do manipulador	50
Figura 42 – Janela de visualização com o manipulador na posição inicial e a final já determinada	50
Figura 43 – Posição final do manipulador	51
Figura 44 – Janela de visualização na posição final	51
Figura 45 – Interface de comunicação com o banco de dados do MoveIt	52
Figura 46 – Posição salva para pegar o explosivo	52
Figura 47 – Posição salva para soltar o explosivo	53
Figura 48 – Conjunto de nós e tópicos para o controle junta a junta	53
Figura 49 – Ambiente externo para o teste da câmera	53
Figura 50 – Posicionamento do robô	54
Figura 51 – Imagem da câmera montada na base do veículo	54
Figura 52 – Imagem da câmera montada na parte superior da garra	54
Figura 53 – Ambiente interno a ser mapeado	55
Figura 54 – Resultado do mapa em três dimensões em um ambiente interno	55
Figura 55 – Resultado do mapa em duas dimensões em um ambiente interno	56
Figura 56 – Resultado do mapa em três dimensões em um ambiente externo	57
Figura 57 – Resultado do mapa em duas dimensões em um ambiente externo	57
Figura 58 – Posição inicial e posição final do objetivo	57
Figura 59 – QR code referente ao deslocamento do veículo em ambiente interno	58
Figura 60 – Posição inicial do e posição objetivo	58
Figura 61 – QR code referente ao deslocamento do veículo em ambiente externo	59
Figura 62 – Esquema geral dos componentes do veículo	60
Figura 63 – Câmera específica para ser utilizada no raspberry	62
Figura 64 – Montagem da câmera no microcomputador	63
Figura 65 – Modelo CAD da proteção da câmera	63
Figura 66 – Parte da frente da proteção montada na câmera	64
Figura 67 – Parte da frente da proteção montada na câmera	64
Figura 68 – Tópicos referentes ao nó da câmera	64
Figura 69 – Parâmetros que podem ser configurados dinamicamente	65
Figura 70 – QR Code referente ao vídeo de teste da câmera	65
Figura 71 – Pinagem do arduino Mega	66

Figura 72 – Conexão entre o arduino e o raspberry pi	67
Figura 73 – Eixos de medida do acelerômetro em relação ao sensor	67
Figura 74 – Eixos de medida do giroscópio em relação ao sensor	68
Figura 75 – Sensor MPU-605	68
Figura 76 – Conexão entre o arduino mega e o MPU6050	69
Figura 77 – Leitura da aceleração linear no eixo x do sensor antes da calibração	69
Figura 78 – Leitura da aceleração linear no eixo x do sensor após da calibração	69
Figura 79 – Sabertooth 2x12 (9)	70
Figura 80 – Saídas PWM (9)	70
Figura 81 – Modelo de atrito de coulomb	72
Figura 82 – Modelo de atrito de coulomb linearizado	72
Figura 83 – Representação d veículo no plano	73
Figura 84 – Forças dos pneus durante o movimento do veículo	73
Figura 85 – Representação de sistema em diagrama de blocos e na forma padrão para a síntese do controlador	75
Figura 86 – Planta representada na forma padrão de controle com a representação das incertezas sendo uma LFT superior	77
Figura 87 – Sistema em malha fechada para a síntese do controle	78
Figura 88 – Forma padrão de controle para o sistema do veículo	78
Figura 89 – Ganho das saídas ponderadas do sistema em todo o domínio da frequência	79
Figura 90 – Velocidade longitudinal do sistema em malha fechada quando submetido a um degrau de velocidade linear	80
Figura 91 – Velocidade angular do sistema em malha fechada quando submetido a um degrau de velocidade angular	80
Figura 92 – Velocidade longitudinal do sistema em malha fechada quando submetido a um degrau de velocidade angular	81
Figura 93 – Velocidade angular do sistema em malha fechada quando submetido a um degrau de velocidade longitudinal	81
Figura 94 – Comportamento de um sistema discretizado por meio do método <i>zero order hold</i>	82
Figura 95 – Comparação entre as respostas de velocidade linear das malhas contínua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s	83
Figura 96 – Comparação entre as respostas de velocidade angular das malhas conti- nua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s	84
Figura 97 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidade de 0.3 m/s	84

Figura 98 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s	85
Figura 99 – Comparação entre as respostas de velocidade angular das malhas contínuos e discretas devido a uma entrada em degrau de velocidade de 0.3 m/s	85
Figura 100 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidade de 30 °/s	86
Figura 101 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas	87
Figura 102 – Comparação entre as respostas de velocidade angular das malhas contínuos e discretas devido a uma entrada em degrau de velocidades combinadas	87
Figura 103 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidades combinadas	88
Figura 104 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas	89
Figura 105 – Comparação entre as respostas de velocidade angular das malhas contínuos e discretas devido a uma entrada em degrau de velocidades combinadas	89
Figura 106 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas	90
Figura 107 – Comparação entre as respostas de velocidade angular das malhas contínuos e discretas devido a uma entrada em degrau de velocidades combinadas	90
Figura 108 – Comparação entre as respostas de velocidade linear do controlador no Matlab e no Arduino devido a uma entrada em degrau de velocidades combinadas	90
Figura 109 – Comparação entre as respostas de velocidade angular do controlador no Matlab e no Arduino devido a uma entrada em degrau de velocidades combinadas	91
Figura 110 – Recorte das respostas dos sistemas para a saída de velocidade longitudinal	91
Figura 111 – Recorte das respostas dos sistemas para a saída de velocidade angular	91
Figura 112 – Visão frontal da montagem do veículo	92
Figura 113 – Visão lateral da montagem do veículo	92
Figura 114 – <i>QR code</i> relativo ao vídeo do teste do veículo controlado a distância . .	93
Figura 115 – <i>QR code</i> complementar relativo ao vídeo do teste do veículo controlado a distância	94
Figura 116 – <i>QR code</i> relativo à visão do operador do veículo	94

Figura 117– <i>QR code</i> complementar relativo ao vídeo do teste do modo de aumento da velocidade do veículo	94
Figura 118– <i>QR code</i> relativo à visão do operador do veículo no teste de aumento de velocidade	94

SUMÁRIO

1	INTRODUÇÃO	15
1.1	JUSTIFICATIVA DO TRABALHO	15
1.2	OBJETIVOS DO TRABALHO	16
1.3	REVISÃO BIBLIOGRÁFICA	16
1.4	POSICIONAMENTO DO TRABALHO	18
2	INTRODUÇÃO AO ROS	19
2.1	JUSTIFICATIVA DA ESCOLHA DO ROS	19
2.2	FUNCIONAMENTO DO ROS	19
2.3	ESTRUTURA DOS PACOTES	21
2.4	FERRAMENTA DE VISUALIZAÇÃO: RVIZ	22
2.5	AMBIENTE DE SIMULAÇÃO	22
2.6	REQUISITOS DO SISTEMA	23
3	ROBÓTICA PROBABILÍSTICA	24
3.1	DISTRIBUIÇÕES DE CONHECIMENTO	24
3.2	FILTROS DE BAYES	25
3.3	MODELO DE MOVIMENTO	26
3.4	MODELOS DE SENSORES	28
3.5	FILTROS DE KALMAN	29
3.6	FILTROS DE KALMAN ESTENDIDOS	30
4	CONSTRUÇÃO DO MODELO	32
4.1	BASE MÓVEL	32
4.2	MANIPULADOR	35
4.3	SISTEMA DE CONTROLE DO MANIPULADOR	36
4.3.1	CINEMÁTICA	37
4.3.2	PLANEJAMENTO DE MOVIMENTO	37
4.3.3	PLANNING SCENE MONITOR	37
4.3.4	PLANEJAMENTO DE TRAJETÓRIA	37
4.4	SISTEMAS AUXILIARES	38
4.5	ÁRVORE DE FUNÇÕES DE TRANSFERÊNCIAS	39
4.6	MAPEAMENTO	41
4.7	DESLOCAMENTO AUTÔNOMO	42
5	SIMULAÇÕES	45
5.1	DESLOCAMENTO TELEOPERADO EM AMBIENTE INTERNO	46

5.2	TESTES DOS SISTEMAS DE CONTROLE DO MANIPULADOR	49
5.2.1	MANIPULAÇÃO POR MEIO DA INTERFACE GRÁFICA	49
5.2.2	BANCO DE DADOS	51
5.2.3	CONTROLE POR MEIO DE JOYSTICK	52
5.3	COMPONENTES AUXILIARES	53
5.4	MAPEAMENTO	54
5.4.1	AMBIENTE INTERNO	55
5.4.2	AMBIENTE EXTERNO	56
5.5	DESLOCAMENTO AUTÔNOMO	56
5.5.1	AMBIENTE INTERNO	56
5.5.2	AMBIENTE EXTERNO	58
6	MONTAGEM DOS SISTEMAS DO VEÍCULO	60
6.1	COMPONENTES DO VEÍCULO	60
6.1.1	ROS MASTER	60
6.1.2	RASPBERRY PI	61
6.1.3	CAMERA	62
6.1.4	ARDUINO	63
6.1.5	SENSOR MPU6050	66
6.1.6	PLACA CONTROLADORA DO MOTOR	69
7	SISTEMA DE CONTROLE	71
7.1	DINÂMICA VEICULAR	71
7.2	SÍNTESE DO CONTROLADOR	74
7.2.1	REPRESENTAÇÃO NA FORMA PADRÃO	74
7.2.2	REPRESENTAÇÃO DAS INCERTEZAS	77
7.3	DISCRETIZAÇÃO DO CONTROLE	81
7.3.1	DEGRAU DE VELOCIDADE LINEAR DE 0.3 M/S	83
7.3.2	DEGRAU DE VELOCIDADE ANGULAR DE 30 °/S	85
7.3.3	DEGRAU DE VELOCIDADE LINEAR E ANGULAR COMBINADOS	86
7.4	VALIDAÇÃO DO CONTROLE NO HARDWARE (<i>HARDWARE IN THE LOOP SIMULATION</i>)	88
8	TESTES NO VEÍCULO	92
9	CONCLUSÕES	95
	REFERÊNCIAS	97
	ANEXO A – PARÂMETROS DE DESLOCAMENTO AUTÔNOMO	100

ANEXO B – PROGRAMA DO ARDUINO PARA O TESTE DE HARDWARE	102
ANEXO C – PROGRAMA DO ARDUINO PARA O TESTE DO SISTEMA COM A LEITURA DOS SENSORES . . .	104
ANEXO D – PROGRAMA DO ARDUINO PARA O TESTE DO SISTEMA SEM A LEITURA DOS SENSORES	108

1 INTRODUÇÃO

1.1 Justificativa do Trabalho

A robótica vem se tornando cada vez mais essencial no ambiente militar, deixando de ser algo futurístico para ser realmente uma realidade e, mais do que isso, uma necessidade nos conflitos armados. Nos recentes conflitos entre Armênia e Azerbaijão e entre Rússia e Ucrânia deixam evidentes que os exércitos precisam de meios robotizados para avançar no terreno, pois os sistemas robóticos permitem minimizar baixas, já que diminuem a exposição dos militares ao fogo inimigo.

Sendo o cenário de drones uma realidade, começa-se a corrida pelo desenvolvimento de tecnologias semelhantes também para os veículos terrestres, aumentando-se a necessidade de sistemas que podem ser controlados à distância e que possuam algum tipo de autonomia. Esses sistemas demandam um conjunto de sensores, algoritmos de visão e de otimização, entre outros, que devem estar trabalhando paralelamente. Portanto, é fundamental que um ambiente de simulação possa ser utilizado para diminuir o tempo de testes, otimizar previamente parâmetros, adiantar-se a eventuais problemas e evitar acidentes que podem danificar o sistema.

Dentre as diversas plataformas para simulação de robôs, o ambiente Gazebo é o escolhido para o presente trabalho. Isso porque é um ambiente baseado em ROS (*Robot Operating System*) de plataforma aberta, estando de acordo com o plano de migração para software livre do Exército Brasileiro (Portaria 046-DCT de 12 de dezembro de 2005). Além disso, possui uma série de ferramentas que podem acelerar o desenvolvimento de diversas funcionalidades.

É importante ressaltar que o ROS é muito mais do que apenas um ambiente de simulação e pode ser utilizado para o controle de robôs, não apenas na simulação. Entretanto, por mais que isso seja extremamente realizado na indústria em geral, esse sistema possui uma série de debilidades que poderiam ser exploradas em um ambiente de guerra eletrônica. Portanto, não é interessante utilizar essa arquitetura diretamente nesse contexto. Mas, isso não impede que se utilizem os métodos já existentes e disponibilizados na comunidade de desenvolvedores ROS, por exemplo, sistemas de navegação autônoma, e reescrevê-los em outra arquitetura mais eficaz para o projeto.

1.2 Objetivos do Trabalho

O presente trabalho tem como objetivo criar um ambiente virtual baseado em ROS que possibilite a simulação de veículo terrestres tanto autônomos quanto teleoperados, mas também forneça uma base sólida para que o ambiente de trabalho possa ser facilmente expandido para outras aplicações, como braços robóticos e drones.

O ambiente virtual permite os testes de sistemas de controle, de visão, navegação e autonomia que se desejem aplicar ao veículo, sendo uma ferramenta poderosa para facilitar o desenvolvimento e otimização desses sistemas. O presente estudo possui um foco inicial no desenvolvimento de um veículo terrestre com um manipulador acoplado, para servir em operações auxiliando grupos de detonação de explosivos. Portanto, uma série de aplicações devem ser desenvolvidas a fim de possibilitar a fácil operação do sistema. Dentre elas, o método de SLAM (*simultaneous localization and mapping*) a ser utilizado é uma das tarefas mais complexas e de maior importância, pois permite ao sistema saber onde ele está no ambiente, dando suporte a um grau de autonomia que o veículo pode ter, além de possibilitar mapas detalhados que são úteis ao operador.

Além da parte de simulação, também é realizada a aplicação desse modelo em um veículo real, inserindo no rover o sistema ROS. Devido a limitações materiais nem todos os sistemas desenvolvidos na simulação puderam ser empregados, com destaque maior ao método de SLAM, que depende de um sensor *LIDAR*. Entretanto, o veículo pode ser controlado a distância e também são inseridos sensores inerciais, giroscópio e acelerômetro, bem como uma câmera de forma a receber as respostas do sistema aos comandos enviados.

Um sistema de controle é desenvolvido para o veículo com o objetivo de controlar as velocidades longitudinal e angular do veículo. Por conta da aplicação do veículo, que é de cunho militar, é necessário que o sistema de controle seja robusto o suficiente para resistir a variações no terreno e de carregamento do veículo, por conta disso é desenvolvido um sistema de controle robusto para o veículo.

Buscando a aplicação desse controlador em sistemas digitais esse controle é discretizado. Um teste de aplicação é realiza utilizando uma malha no Simulink e o Hardware no qual se deseja aplicar, ambas trabalhando em paralelo.

1.3 Revisão Bibliográfica

O primeiro passo é a construção do veículo no ambiente gazebo. Para tanto utiliza-se o trabalho de (10), no qual se realiza a modelagem e testes de movimento de um robô sobre rodas. Esse trabalho fornece explicações acerca da forma de declaração e organização dos arquivos *URDF*, que contém informações visuais, de impacto, atrito e entre outros, essenciais na declaração do robô. Mais do que isso, encontra-se em (10)

as relações dinâmicas, bem como aproximações, utilizadas pelo Gazebo para simular o ambiente, como a dinâmica de atrito, da atmosfera e etc.

Após o desenvolvimento da base é necessário criar o manipulador que é acoplado nela. Para tanto, é utilizado o mesmo processo presente em (11), no qual o braço tem seu *URDF* modelado e integrado ao ROS por meio do pacote *MoveIt*, que agrupa uma série de controladores de posição, deslocamento e de força, que permitem um controle simples e prático do manipulador. Isso mostra uma das potencialidades desse sistema operacional, que possui grande modularidade. O mesmo pacote pode ser utilizado por robôs diferentes, e pode ser igualmente empregado em simulações, como no presente trabalho, e manipuladores reais, como é o caso de (11).

Em (12) é utilizada uma estratégia de controle do manipulador por meio de uma teleoperação, enviando os comandos de deslocamento diretamente para as juntas do robô, uma maneira diferente e com menos possibilidades da apresentada por (11), mas que pode funcionar como um sistema paralelo.

Os algoritmos de SLAM (*simultaneous localization and mapping*) são baseados na teoria de robótica probabilística presente em (3). Nesse livro estão presente desde os modelos de movimento do veículo, de leitura de sensores e os algoritmos referentes aos Filtros de Kalman, uma parte essencial para o funcionamento dos métodos de SLAM.

Sistemas de mapeamento como o utilizado por (13) permitem que se tenha uma análise situacional do ambiente, ampliando em muito as capacidades do operador, além de permitir a implementação de sistemas de navegação autônoma. Nele é utilizado uma combinação de um sensor LIDAR 2D e uma câmera RGB-D para a criação de um mapa em três dimensões do ambiente.

Existem diversos servidores diferentes de mapas que podem ser criados. Encontra-se em (14) uma comparação entre os três mais utilizados, *Gmapping*, *Cartographer* e *Hector Slam*, trabalho no qual se obtém como resultado que os pacotes *Gmapping* e *Cartographer* apresentam melhor precisão, pois recebem como entrada dados de odometria do robô (tais como a velocidade angular das rodas).

Já nos trabalhos de (15) e (6) são testados deslocamentos autônomos para ambientes externos e internos, respectivamente, utilizando o método de mapeamento *Gmapping*. O sistema utilizado é muito útil por ser de simples implementação, já que não depende de algoritmos complexos de inteligência artificial, apenas um sistema de mapeamento e o pacote *move base*.

Apesar deste método de deslocamento autônomo ser relativamente simples de empregar, ele possui uma lista grande de variáveis que precisam ser otimizadas para que o sistema funcione da melhor forma possível. Nesse contexto, o trabalho desenvolvido por (8), no qual há uma descrição detalhada de todos os parâmetros que podem ser alterados

bem como de boas práticas a seguir, pode servir como um norteador para a otimização dessas variáveis. Aliado a isso, o trabalho de (16) possui um foco muito mais voltado em otimizar as velocidades do rôbo, o intervalo de atualização do mapa e o filtro de partículas, gerando resultados muito mais detalhados acerca desses parâmetros.

1.4 Posicionamento do Trabalho

O presente trabalho se enquadra no estudo de desenvolvimento de veículos robóticos. Nele se destacam os seguintes tópicos:

- Desenvolvimento de um veículo terrestre em ROS.
- SLAM
- Testes de sistemas de navegação e deslocamento autônomo.
- Sistemas de controle de um manipulador com 4 graus de liberdade.

Portanto, o presente estudo pode contribuir promovendo as bases para futuros trabalhos em ROS e Gazebo, que podem ser explorados de formas extremamente variados em trabalhos futuros.

2 INTRODUÇÃO AO ROS

2.1 Justificativa da Escolha do ROS

ROS pode ser resumido como um conjunto de bibliotecas e ferramentas de código aberto, com aplicações para robôs. Possui uma grande variedade de aplicações, de drivers para motores até algoritmos de exames de drones. Além disso, possui uma extensa comunidade mundial que permite a rápida troca de informações e facilita a aplicação e desenvolvimento dessas tecnologias. Dentre as suas principais vantagens, (1) pontua como principais:

- **Recursos de ponta:** pode-se instalar pacotes de alta complexidade muito facilmente. Por exemplo, métodos de SLAM (*Simultaneous Localization and Mapping*) e AMCL (*Adaptive Monte Carlo Localization*) para o desenvolvimento de navegação autônoma em bases móveis. Caso seja necessário controlar um braço robótico existe o pacote Moveit. E portanto, não é necessário reescrever códigos semelhantes, sendo que já existem exemplos funcionais e altamente customizáveis.

- **Grande número de ferramentas:** o sistema pode ser configurado com inúmeras ferramentas de visualização de dados, alteração de parâmetros, simulação e *debugging*.

- **Suporta um grande número de sensores:** muitas soluções já foram desenvolvidas para uma grande gama de sensores comerciais, tais como *Kinect*, Laser scanners, GPS, câmeras, entre outros.

- **Plataforma permite a operação de diversas linguagens:** o seu sistema baseado em nós e mensagens permite que programas escritos em diferentes linguagens de programação (*Python*, C++, Java, etc) se comuniquem.

- **Comunidade ativa:** essa é uma ferramenta *open source* com uma imensa comunidade de desenvolvedores ativos. Isso acelera o desenvolvimento, pois não é necessário construir ferramentas que já existem, e permite a retirada de dúvidas.

2.2 Funcionamento do ROS

O sistema operacional ROS é organizado em de uma maneira particular. Seu processo computacional utiliza um processo de cálculo em rede denominado nós. Os principais conceitos desse método é mostrado na figura 1.

De acordo com (1), pode-se explicar cada um dos parâmetros da figura 1 como:

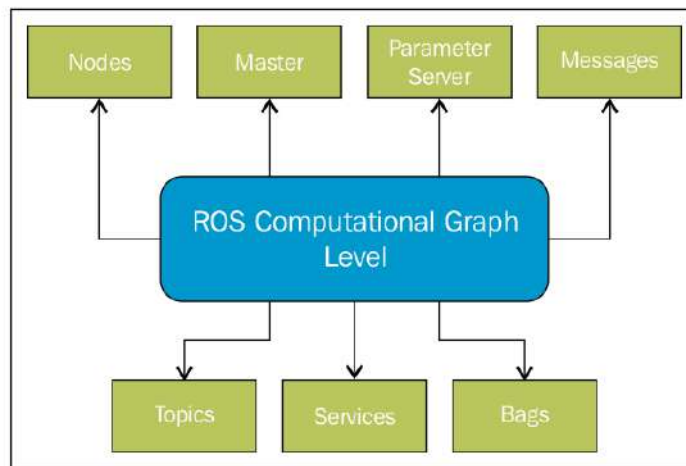


Figura 1 – Estrutura do sistema ROS (1)

- **Nós (*nodes*):** são os processos que realizam os cálculos computacionais, ou seja, que executam tarefas, como receber e filtrar dados de sensores, calcular distância de obstáculos com base em sensores a laser e etc. Cada nó pode ser escrito usando as bibliotecas *roscpp* e *rospy*, baseadas respectivamente em C++ e *Python*. Essa estrutura permite que se *debug* códigos e sistemas complexos com mais facilidade;

- **ROS Master:** fornece um registro e uma supervisão de todos os nós. A partir dele que os nós encontrarão uns aos outros para trocar mensagens. Ele inclui o *Parameter Server*, que nada mais é do que um parâmetro que permite o armazenamento de informações em uma central local;

- **Mensagens:** os nós se comunicam entre si utilizando mensagens. São uma simples estrutura de contendo o tipo da mensagem e sua informação;

- **Tópicos:** canais que transportam as mensagens. Quando um nó envia uma mensagem é necessário que ele publique ela em um tópico, sendo denominado de *publisher*, já quando ele a recebe é denominado de *subscriber*;

- **Serviços:** é usado para interações de demanda/resposta dos sistemas. Usando esse sistema é possível criar um nó cliente, que faz o pedido de uma ação, e o servidor que emitirá a resposta; e

- **Bags:** são uma forma de salvar e armazenar dados.

Para exemplificar esses conceitos é possível utilizar a Figura 2, na qual há dois nós, um *talker*, que publica a mensagem, e um *listener*, que recebe a mensagem por meio do tópico *talker*. É possível notar que o ROS Master está ligado à ambos os nós e a mensagem é uma string (*std_msgs/String*) cujo conteúdo é "Hello World". Exemplificando assim os principais conceitos de comunicação do ROS.

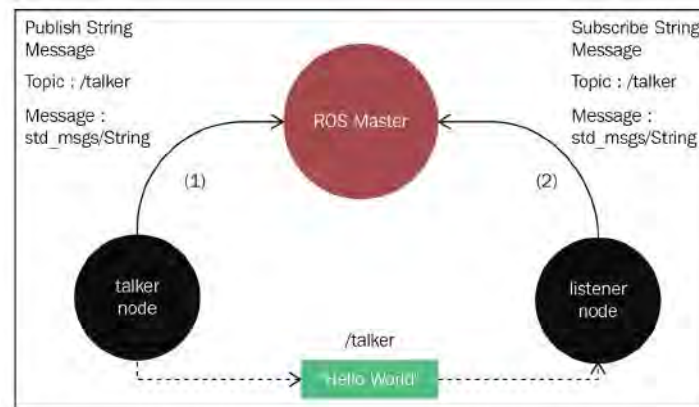


Figura 2 – Exemplo do funcionamento do sistema ROS (2)

2.3 Estrutura dos Pacotes

Além do processo computacional é necessário entender como são organizados os pacotes ROS. Os pacotes são as estruturas mais básicas do sistema, contém nós, bibliotecas, arquivos de configuração, serviços, entre outros. Um típico pacote ROS está indicado na figura 3, contendo 8 pastas e dois arquivos de texto.

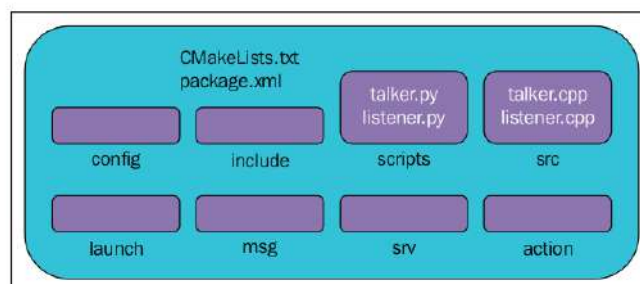


Figura 3 – Organização do pacote ROS (1)

Pode-se discutir o uso de cada pasta, de acordo com (1), da seguinte forma:

- *Config*: representa os parâmetros que podem ser alterados em um nó, como por exemplo os valores das constantes de um controlador PID, variáveis de um algoritmos de deslocamento autônomo, entre outros;
- *Include*: consiste em definição de variáveis e bibliotecas que são usadas nas pastas *script* e *src*;
- *Scripts*: armazena os executáveis em python;
- *Src*: armazena os executáveis em c++;
- *Launch*: utilizado para que se possa lançar um ou mais nós ao mesmo tempo;
- *Msg*: contém as definições de mensagens customizadas;

- *Srv*: contém as definições de serviços;
- *Action*: contém as definições de serviços de longa duração;
- *package.xml*: contém as dependências do pacote, bem como informações a certa do autor, data e estado de desenvolvimento; e
- *CMakeLists.txt*: arquivo utilizado quando se constrói o pacote. É a partir dele que bibliotecas podem ser exportadas para outros nós, bem como se todo o sistema do pacote está funcionando com todas as dependências e sem nenhum erro.

2.4 Ferramenta de Visualização: Rviz

O Rviz é uma ferramenta de visualização 3D extremamente poderosa, ela permite a análise e acompanhamento de robôs, sensores, ferramentas e algoritmos. Além disso, possibilita ao programador ou operador da ter uma visão da percepção do robô a cerca do ambiente, seja ele real ou simulado (1). A aparência dessa ferramenta é mostrada na Figura 4

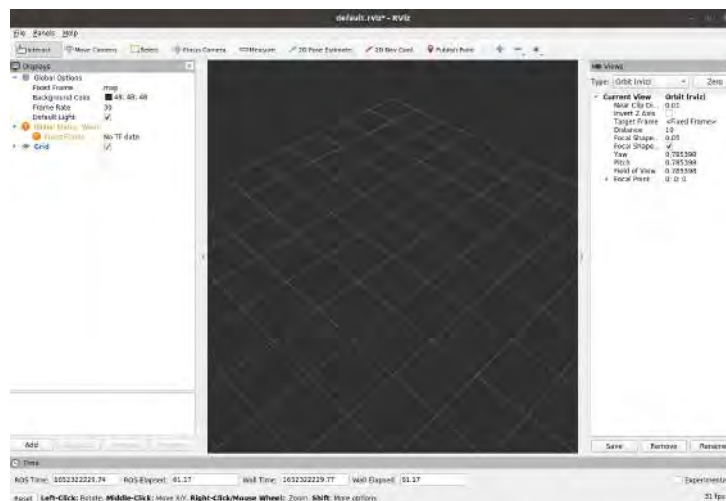


Figura 4 – Visualizador Rviz

2.5 Ambiente de Simulação

Gazebo é um simulador para robótica tanto para ambientes internos quanto externos. Pode representar uma grande gama de robôs, de sensores e objetos 3D, além de possuir uma grande base de dados aberta de programas já prontos. Além disso, a integração do Gazebo com o ROS é muito facilitada pela arquitetura do software (1). As relações matemáticas acerca dos parâmetros físicos desse sistema de simulação podem ser encontrados em (17).

2.6 Requisitos do Sistema

Primeiramente é necessário possuir um sistema Ubuntu para que se possa instalar o ROS. Apesar de ser possível utilizar uma máquina virtual que simule o esse sistema Linux isso não é recomendável pois diminui a performance do ambiente.

Além disso, é possível notar que existem diversas distribuições tanto de ROS quanto do Ubuntu, sendo fundamental que se utilize uma distribuição ROS compatível com o sistema operacional. Devido ao fato deste ser um sistema em contínuo desenvolvimento e aprimoramento por parte da comunidade, utiliza-se no presente trabalho o ROS Noetic, a mais recente versão, e como consequência direta é necessário utilizar o Ubuntu 20.04. Existem vantagens e desvantagens de se utilizar a versão mais recente do ROS. Como ponto positivo pode-se destacar a maior confiabilidade dos pacotes e estabilidade do sistema, pois em versões mais recentes são corrigidos problemas de versões anteriores. Entretanto, como ele ainda está em desenvolvimento existe um grande número de aplicações que ainda não foram integradas completamente com esse novo distribuidor, podendo levar a incoerência entre pacotes.

No que tange ao ambiente de simulação o *Gazebo 11* deve ser instalado, para garantir a compatibilidade com o ROS *Noetic*. Neste caso, é importante instalar o sistema direto da fonte (*source install*), a instalação por comando *sudo* não é recomendável.

3 ROBÓTICA PROBABILÍSTICA

Sistemas robóticos inteligentes precisam interagir com o ambiente por meio de sistemas mecânicos e eletrônicos. Esta interação é sempre acompanhada por incertezas no sistema, tais como: limitações e ruídos dos sensores, ambientes com alto grau de imprevisibilidade, atuadores que não são totalmente precisos, os modelos dinâmicos utilizados podem não representar de forma perfeita o sistema além de limitações computacionais de sistemas trabalhando em tempo real.

Para lidar com a incerteza nos sistemas robóticos foi desenvolvida a robótica probabilística, cujo objetivo é representar de forma explícita o cálculo da incerteza, dadas as medidas dos sensores e o sinal de controle do robô. Matematicamente esse objetivo está representado na Eq. 3.1, na qual x é o estado do veículo, z é o vetor da leitura dos sensores e u é o vetor dos sinais de controle.

$$p(x|z, u) \quad (3.1)$$

Para o presente trabalho, no qual um dos objetivos é a obtenção de um mapa do ambiente, o vetor x representa a posição e a orientação do veículo no plano, enquanto z é a leitura dos sensores a *Lidar* e u são os comandos de velocidade para o veículo.

3.1 Distribuições de Conhecimento

O conhecimento (*belief*) é um dos termos mais importantes em robótica probabilística. Ela reflete o conhecimento interno do robô acerca do ambiente. Ela é uma distribuição de probabilidade condicionada aos dados disponíveis dos sensores e dos comandos fornecidos ao robô, como mostra a Eq. 3.2.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \quad (3.2)$$

Futuramente, na análise dos filtros de Bayes, será útil realizar a análise do *belief* antes de se realizar a medição em z_t , chegando-se portando em:

$$\overline{bel}(x_t) = p(x_t|z_{1:t}, u_{1:t}) \quad (3.3)$$

Essa distribuição de probabilidade é comumente referenciada como a predição do modelo. Isso se deve ao fato de que $\overline{bel}(x_t)$ prediz o estado do tempo t baseado no estado anterior, antes de incorporar a medição no tempo t .

3.2 Filtros de Bayes

Este é um algoritmo geral para o cálculo do *belief* do modelo. Partindo-se da distribuição da Eq. 3.2, pode-se aplicar o teorema de Bayes para obter a Eq. 3.4, no qual o parâmetro η é termo um normalizador da distribuição. A explicação mais detalhada dos conceitos matemáticos constando nessa transformação pode ser encontrada em (3)

$$bel(x_t) = \eta p(z_t|x_t, z_{1:t-1}, u_{1:t}) p(x_t|z_{1:t-1}, u_{1:t}) \quad (3.4)$$

Além disso, pode-se aplicar a consideração de Markov sobre a primeira distribuição da Eq. 3.4, ou seja, como o estado do ambiente é conhecido a um tempo t é possível ignorar todas as medidas e comandos realizados antes do tempo t . Isso leva a Eq. 3.5.

$$bel(x_t) = \eta p(z_t|x_t) p(x_t|z_{1:t-1}, u_{1:t}) \quad (3.5)$$

Já a segunda distribuição está relacionada com a estimação do estado atual dado todas as medições e comando realizados até esse momento. Aplicando a lei da probabilidade total a essa distribuição a equação resultante será:

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} \quad (3.6)$$

Aplicando mais uma vez a consideração de Markov à distribuição de probabilidade de x_t é possível eliminar todos os comandos e as medidas do sistema até o tempo $t - 1$, pois é conhecido o estado x_{t-1} . A partir disso chega-se na Eq. 3.7

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}u_t) p(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} \quad (3.7)$$

Além disso, pode-se utilizar mais uma vez a consideração de Markov, mas agora para o termo futuro, pois segundo essas considerações ele não fornece informações sobre o instante anterior ao sistema e, portanto, chega-se na Eq. 3.8. Isso claramente é uma suposição, já que em diversos casos saber os sinais de controle futuros podem melhorar o conhecimento do estado atual, porém isso é algo negligenciado neste caso.

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}u_t) p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} \quad (3.8)$$

Aplicando a Eq. 3.2 na Eq. 3.8 se tem como resultado a Eq. 3.9, que relaciona o *belief* de um instante de tempo com o anterior.

$$bel(x_t) = \eta p(z_t|x_t) \int_{x_{t-1}} p(x_t|x_{t-1}u_t) bel(x_{t-1}) dx_{t-1} \quad (3.9)$$

Ou seja, é possível saber a distribuição do estado atual do robô com base em seu estado passado, no seu sinal de controle e na leitura dos sensores. Na Figura 5 está representado o algoritmo do que foi tratado matematicamente. Nele, o cálculo é dividido em dois passos um denominado passo de predição, $\overline{bel}[x_t]$, e outro de correção, $bel(x_t)$. Ou seja, primeiramente o sistema prediz o seu estado futuro, com base no estado inicial e no comando, e após isso realiza correções com base na medida dos sensores.

```

1:   Algorithm Bayes filter( $bel(x_{t-1}), u_t, z_t$ ):
2:     for all  $x_t$  do
3:        $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$ 
4:        $bel(x_t) = \eta p(z_t | x_t) \overline{bel}(x_t)$ 
5:     endfor
6:     return  $bel(x_t)$ 

```

Figura 5 – Algoritmo geral para filtros de Bayes (3)

O filtro de Bayes é apenas uma estrutura de trabalho, portanto pode possuir diferentes realizações. A escolha da realização correta vai depender das características do sistema, por exemplo se o modelo for linear ou não, se as distribuições são todas gaussianas, entre outras. Dois modelos de realização são os mais comuns, a família dos filtros de Kalman e a família de filtro de partículas. Enquanto os de Kalman são indicados para distribuições gaussianas e modelos lineares ou linearizados, o filtro de partícula já é mais geral servindo para modelos arbitrários. No presente trabalho, será dada uma importância maior para os filtros de Kalman, pois esse é o método utilizado pelo *Gmapping* do ROS.

Mas, além da estrutura de trabalho, é necessário estudar como se dão as distribuições de probabilidade da leitura do sensor e de movimento do modelo, para calcular o método recursivo.

3.3 Modelo de movimento

Essa é a distribuição de probabilidade relativa à predição do estado do robô dado o seu estado anterior e os comandos. Esse modelo possui uma incerteza inerente, seja por dinâmicas que não foram modeladas corretamente, situações abruptas que não estão previstas, entre outras.

Para deixar mais claro como o modelo é intrinsecamente incerto, pode-se imaginar um veículo que faz um mapa 2d do ambiente e que utiliza os dados de odometria das rodas como o modelo de movimento, ou seja, por meio da rotação delas ele pode prever a próxima localização do veículo no ambiente. Entretanto, uma corrente muito forte de

ar pode estar atuando, ou pode haver um deslizamento de uma das rodas no solo, o que acarreta uma diferença entre o esperado e o real.

Em robótica, em geral, utilizam-se dois modos: *Odometry-based* e *Velocity-based*. No primeiro admite-se que existem sensores nas rodas do veículo e que fornecem a rotação delas, é o mais indicado em aplicações de veículos robóticos terrestres. Enquanto isso, o segundo modelo admite que são enviados comandos de velocidade ao veículo e que esses comandos são efetivamente as velocidades desse veículo, é o mais indicado para o caso de veículos aéreos.

Sendo coerente com a aplicação do veículo o *Odometry-based* será apresentado mais profundamente. Neste modelo o veículo se desloca de $(\bar{x}, \bar{y}, \bar{\theta})$ para $(\bar{x}', \bar{y}', \bar{\theta}')$, como indicado pela Figura 6, no qual o estado final é obtido pelas informações de odometria.

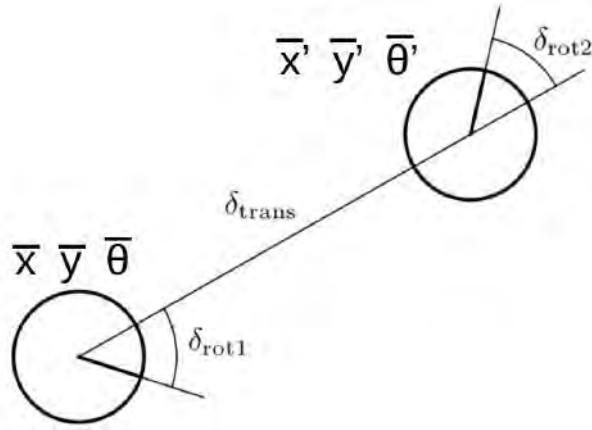


Figura 6 – Modelo de deslocamento via odômetro

Matematicamente, tem-se que:

$$\delta_{rot1} = atan2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \quad (3.10)$$

$$\delta_{trans} = \sqrt{(\bar{y}' - \bar{y})^2 + (\bar{x}' - \bar{x})^2} \quad (3.11)$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1} \quad (3.12)$$

Para o modelo é assumido que o valor real de rotação e translação é obtido por meio do calculado subtraído de um ruído independente ϵ_b , de média zero e covariância b . Portanto, tem-se:

$$\delta'_{rot1} = \delta_{rot1} - \epsilon_{\alpha_1} |\delta_{rot1}| + \alpha_2 |\delta_{trans}| \quad (3.13)$$

$$\delta'_{trans} = \delta_{trans} - \epsilon_{\alpha_3} |\delta_{trans}| + \alpha_4 |\delta_{rot1}| + |\delta_{rot2}| \quad (3.14)$$

$$\delta'_{rot2} = \delta_{rot2} - \epsilon_{\alpha_1} |\delta_{rot2}| + \alpha_2 |\delta_{trans}| \quad (3.15)$$

Onde os parâmetros de α_1 até α_4 representam erros específicos do robô para esse movimento. Conseqüentemente é possível obter a posição predita por meio da posição anterior, como fica claro na Eq.

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \delta'_{trans} \cos(\theta + \delta'_{rot1}) \\ \delta'_{trans} \sin(\theta + \delta'_{rot1}) \\ \theta + \delta'_{rot1} + \delta'_{rot2} \end{bmatrix} \quad (3.16)$$

3.4 Modelos de Sensores

Além do modelo de movimento do veículo é necessário conhecer o modelo do sensor para utilizar o algoritmo de Bayes, presente na Figura 5. O sensor utilizado para que o veículos se localize no ambiente foi um *Lidar*, que é representado por um modelo de sensor laser.

O método utilizado é o *Ray-Cast Model*, que é a união de quatro modelos de distribuição diferentes e possui o formato mostrado na Figura 7.

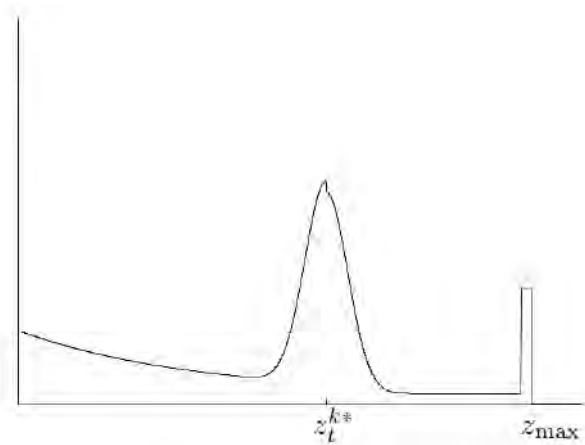


Figura 7 – Distribuição de probabilidade da leitura do sensor

O primeiro componente é uma distribuição gaussiana em volta do obstáculo. Entre o sensor e a gaussiana do obstáculo existe um decaimento exponencial, que pode traçar obstáculos variáveis no ambiente. O pico final está relacionado ao máximo físico de distância que o sensor pode medir. E por último existe uma distribuição constante muito pequena que serve para determinar algum efeito aleatório que não pode ser medido ou estimado.

3.5 Filtros de Kalman

Os filtros de Kalman são modelos de filtros de Bayes, sendo ferramentas extremamente poderosas quando se possui informações incertas. É um método muito recomendado quando se trabalha com sistemas que estão em constante mudança, já que são leves de um ponto de vista computacional. Para o uso específico em algoritmos de SLAM, os modelos de Kalman têm como objetivo melhorar a fusão das distribuições de probabilidade preditiva e dos sensores.

Este modelo é apropriado para sistemas gaussianos lineares ou linearizados, possuindo uma resposta ótima para esse tipo de sistema. Já que os modelos obedecem a uma distribuição gaussiana eles podem ser escritos da forma da Eq. 3.17. Na qual a densidade da variável x é caracterizada pela sua média, μ , e covariância, Σ . A média é um vetor que possui a mesma dimensão que o vetor de estados, enquanto a covariância é uma matriz simétrica e positiva.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left(-\frac{(x - \mu)^T \Sigma^{-1} (x - \mu)}{2}\right) \quad (3.17)$$

O filtro de Kalman assume que o modelo é linearizado, portanto o modelo de movimento e de observação devem ser funções lineares acrescidas de um ruído gaussiano. Na Eq. 3.18, A_t é uma matriz quadrada $n \times n$ que relaciona o estado anterior ao atual, sendo n a dimensão do vetor de estados e B_t uma matriz $n \times m$ que relaciona o sinal de comando com o estado atual, sendo m a dimensão do vetor de controle e o termo ϵ_t se refere ao ruído gaussiano, cuja variância é R_t .

Na Eq. 3.19, tem-se somente a matriz C_t de dimensão $n \times n$ e o parâmetro δ_t que representa o ruído gaussiano desta equação cuja variância é Q_t .

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \quad (3.18)$$

$$z_t = C_t x_t + \delta_t \quad (3.19)$$

Aplicando a suposição de um ruído Gaussiano à Eq. 3.18 se obtém como resultado a distribuição presente na Eq. 3.20, que representa a distribuição de probabilidade do estado na posição t , dado que se conhece o estado na posição $t - 1$ e o sinal de comando.

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left(-\frac{(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1} (x_t - A_t x_{t-1} - B_t u_t)}{2}\right) \quad (3.20)$$

Procedendo de forma análoga é possível encontrar a distribuição referente a medida do sistema, obtendo-se portanto a Eq. 3.21.

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left(-\frac{(z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t)}{2}\right) \quad (3.21)$$

Ao aplicar a Eq. 3.20 e Eq. 3.21 no algoritmo da Figura 5, a partir de um estado inicial, chega-se no algoritmo padrão dos filtros de Kalman, mostrado na Figura 8, na qual μ_t representa a média no tempo t e Σ a sua respectiva variância.

```

1:   Algorithm Kalman filter( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):
2:      $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$ 
3:      $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$ 
4:      $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$ 
5:      $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$ 
6:      $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$ 
7:     return  $\mu_t, \Sigma_t$ 

```

Figura 8 – Algoritmo geral para filtro de Kalman (3)

3.6 Filtros de Kalman Estendidos

Apesar de apresentar ótimos resultados o filtro de Kalman se mostra limitado em aplicação mais realísticas, já que a maior parte dos sistemas não é linear.

Por exemplo, se ao invés das matrizes das Eq. 3.18 e Eq. 3.19, os modelos fossem descritos por funções não lineares descritas pela Eq. 3.22 e Eq. 3.23 o filtro de Kalman perderia a sua validade.

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \quad (3.22)$$

$$z_t = h(x_t) + \delta_t \quad (3.23)$$

Para consertar essa limitação o filtro estendido realiza uma linearização local, ou seja, em torno do ponto de análise. Linearizando as funções tem-se como resultado:

$$g(u_t, x_{t-1}) = g(u_t, \mu_{t-1}) + \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} (x_{t-1} - \mu_{t-1}) \quad (3.24)$$

$$h(x_t) = h(\bar{x}_t) + \frac{\partial h(\bar{x}_t)}{\partial x_t} (x_t - \bar{\mu}_{t-1}) \quad (3.25)$$

Para simplificar a notação as matrizes jacobianas G_t e H_t são introduzidas, de forma que:

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} \quad (3.26)$$

$$H_t = \frac{\partial h(\bar{u}_t)}{\partial x_t} \quad (3.27)$$

Aplicando exatamente os mesmos passos realizados para o filtro de Kalman convencional, a partir das linearizações, chega-se na Fig. 9 que representa o algoritmo de filtro de Kalman estendido.

```

1:   Algorithm Extended Kalman filter( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$ 
3:      $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ 
4:      $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$ 
5:      $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$ 
6:      $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$ 
7:     return  $\mu_t, \Sigma_t$ 

```

Figura 9 – Algoritmo geral para filtro de Kalman estendido (3)

4 CONSTRUÇÃO DO MODELO

O modelo é composto por uma base móvel sobre rodas e um manipulador com quatro grau de liberdade, para realizar a simulação é necessário modelar ambos no formato *URDF*.

4.1 Base Móvel

A base é baseada no Rover Lynxmotion, como mostra a Figura 10, cujos parâmetros geométricos e de massa estão na Tabela 4.1.



Figura 10 – Rover modelo Lynxmotion 4WD1 (imagem de www.lynxmotion.com).

O chassi pode ser modelado como um paralelepípedo retangular, conservando os valores de massa e momento de inércia. Enquanto que cada uma das rodas é modelada por como um cilindro de altura e raio de 6 cm ambos. A utilização de geometrias simples

Tabela 4.1. Parâmetros geométricos e de massa do veículo.

Comprimento do Chassi	18.7 cm
Largura do Chassi	20.3 cm
Diâmetro da Roda	12 cm
Momento de inércia do Chassi	0.00338 kg.m ²
Massa do Veículo	3.044 kg
Momento de inércia da Roda	0.0000331 kg.m ²
Largura da Roda	6 cm

é sempre preferível, desde que sejam sempre próximas das reais, pois diminuem o custo computacional da simulação. Após a montagem do sistema, chega-se no resultado da Figura 11

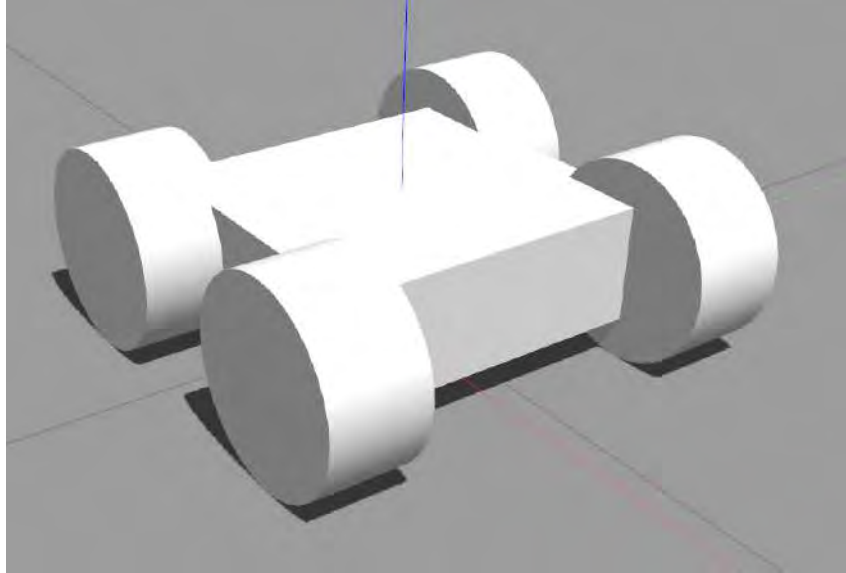


Figura 11 – Base móvel no ambiente Gazebo

A junção entre o chassi e cada uma das rodas é feita por uma transmissão de rotação, que emula o efeito de um motor elétrico com o seu controlador, transmitindo comandos de velocidade para a respectiva roda. Na Figura 12 são mostradas as 4 juntas de rotação que permitem a integração da base com as rodas.

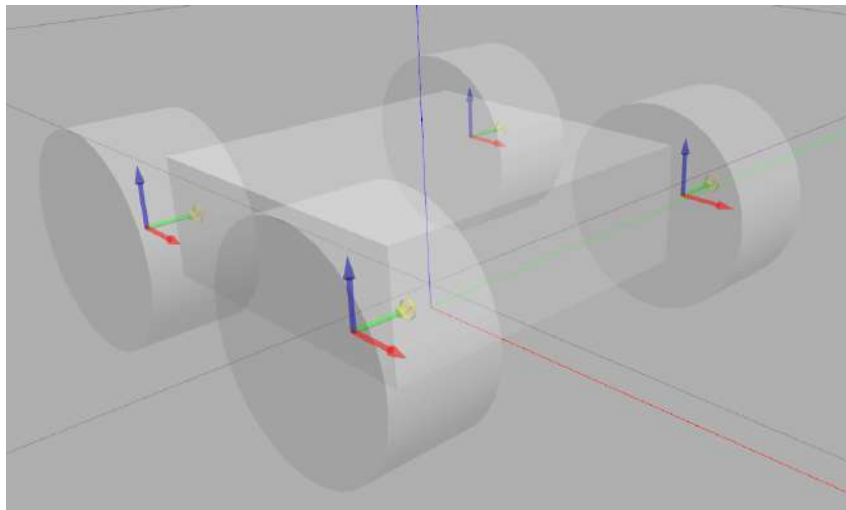


Figura 12 – Juntas de transmissão de velocidade

Para que a simulação seja coerente com a realizada é necessário ainda parametrizar de forma correta o modelo e os parâmetros de atrito das rodas com o solo. Sendo que o atrito é calculado em relação aos pontos de contato entre a roda e o solo, como mostrado

na figura 13. É importante ressaltar que é possível estabelecer um número máximo de pontos de contato a serem considerados quando há impactos entre dois corpos, como o da roda com o solo, sendo dois o máximo escolhido para a interação entre o pneu e o solo. Isso se deve ao fato de que muitos pontos podem elevar consideravelmente o custo computacional da simulação, tornando-a excessivamente lenta (4).

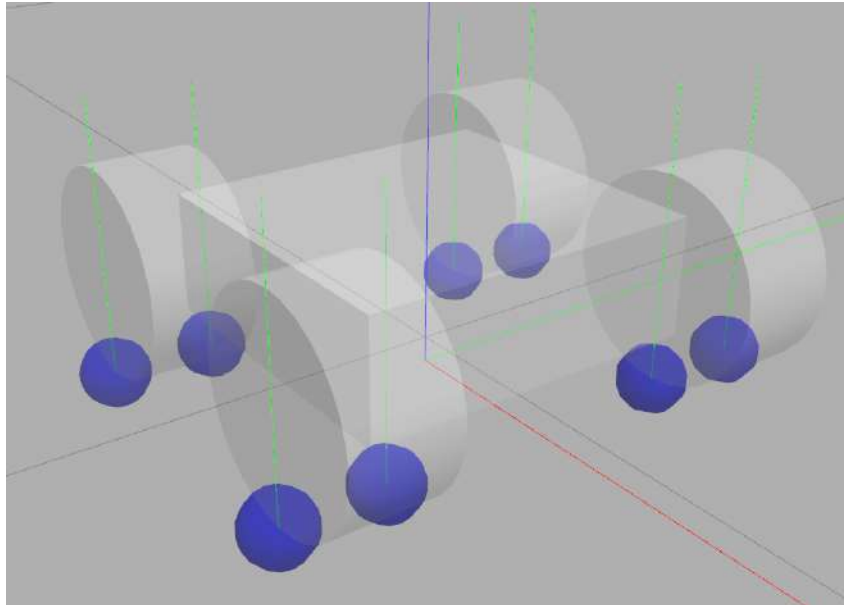


Figura 13 – Contato representado pelas bolas azuis

A partir dos pontos de contato é necessário calcular a dinâmica relacionada a essa interação, para tanto o método utilizado foi o cônico. Segundo (4), ele utiliza dois coeficientes de atrito (μ_1 e μ_2), um na mesma direção da velocidade e um na direção perpendicular a ela, como mostrado na Figura 14, para calcular os esforços do contato.

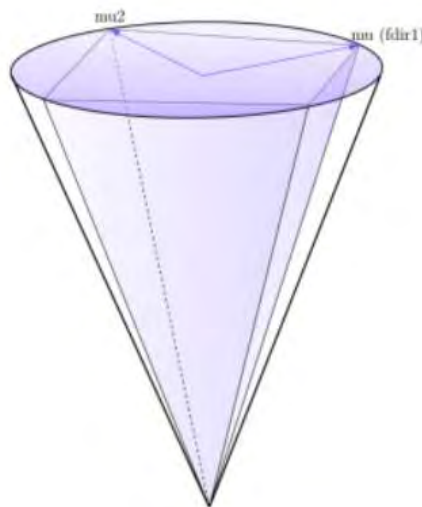


Figura 14 – Modelo de atrito cônico (4)

Além disso, é necessário definir uma forma de comunicação entre o operador e o veículo, para tanto construiu-se um código em $C++$ que recebe como entrada a velocidade linear e angular requerida pelo veículo e tem como saída a velocidade angular das rodas para que se alcance tais velocidades. As relações matemáticas estão explicitadas nas equação 4.1 e 4.2, respectivamente as velocidades angulares das rodas da direita e da esquerda do veículo, e se baseiam na cinemática de um veículo diferencial desenvolvida em (18).

$$\omega(e) = \frac{v_x - v_w(\frac{b}{2})}{r} \quad (4.1)$$

$$\omega(d) = \frac{v_x + v_w(\frac{b}{2})}{r} \quad (4.2)$$

No qual b representa a largura do veículo, r o raio da roda, v_x a velocidade longitudinal e v_w a velocidade angular. Após a construção desse *driver* é possível deslocar a base por meio de comandos pelo tópico *cmd_vel*, que é o tópico de entrada do nó, e analisar o seu resultado pelo tópico *joint_states*, que controla o estado das juntas do veículo.

4.2 Manipulador

Assim como foi feito para a base móvel é necessário criar os arquivos *URDF* também para o braço. Como base do desenvolvimento foi utilizado o manipulador *OpenManipulator-X*, disponível em (5) e apresentado na Figura 15, que apresenta 4 juntas de rotação e uma garra, cujos eixos de rotação estão indicados na Figura 16.



Figura 15 – Manipulador *OpenManipulator-X* (5)

Além da modelagem do braço é necessário definir qual é o tipo de controle que deve ser utilizado. Dentre as opções apresentadas em (19), é utilizado o controle de

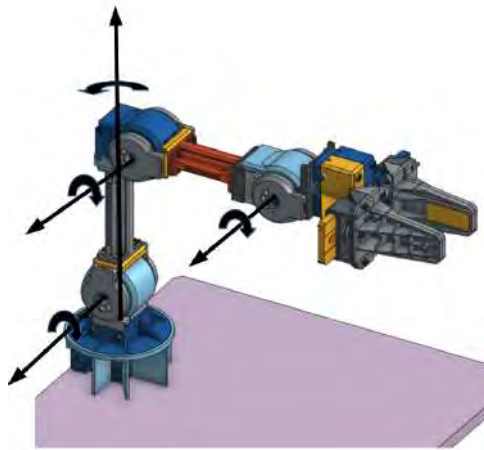


Figura 16 – Modelo do manipulador em uma software CAD com os eixos de rotação (5)

posição (*position_controllers*), que comandam posição desejada para a interface de Hardware do sistema, já que as aplicações de manipulação do sistema não demandam um controle mais fino como o que pode ser realizado por meio dos controladores de trajetória (*joint_trajectory_controllers*). Os parâmetros dos controladores PID utilizados estão presentes na Tabela 4.2.

Tabela 4.2. Parâmetros de controle de posição das juntas.

	P	I	D
Junta do braço	300	0.1	1.0
Garra	300	0.1	1.0

Após essas configurações é necessário acoplar o manipulador a base, sendo possível visualizar o resultado na Figura 17.

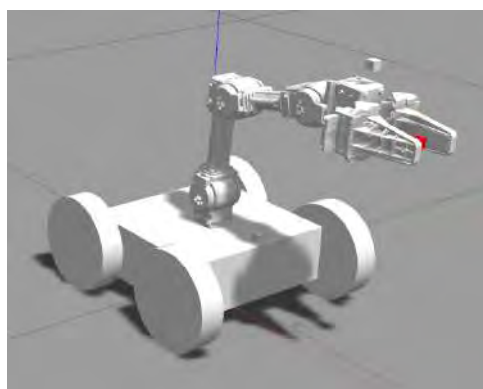


Figura 17 – Base e manipulador acoplados

4.3 Sistema de Controle do Manipulador

Para o controle do manipulador é utilizado o pacote ROS *MoveIt*. Uma aplicação que incorpora algoritmos avançados de planejamento, manipulação, percepção 3D,

cinemática, controle e navegação (20). Pode-se separar nas partes mais importantes dessa aplicação com base em: cinemática, planejamento de movimento, *planning scene monitor* e processamento de trajetória. A seguir são discutidos cada um desses parâmetros, com base na documentação disponibilizada em (20), e sua configuração na simulação.

4.3.1 Cinemática

É o responsável pela solução das cinemáticas direta e inversa. Além disso, a própria arquitetura desse *plugin* permite que o usuário escreva e aplique os seus próprios algoritmos. O padrão utilizado pelo *MoveIt* é o KDL *numerical jacobian-based solver* e este foi mantido na configuração do manipulador.

Esse sistema de cinemática também apresenta a possibilidade de checar possíveis colisões entre partes do mesmo robô (*Allowed Collision Matrix*), entretanto por necessitarem de uma maior capacidade operacional decidiu-se por não utilizar essas funcionalidades, já que não teriam grande impacto para em um manipulador com 4 graus de liberdade.

4.3.2 Planejamento de Movimento

Essa etapa pode ser dividida em 3 partes. Primeiramente, há o pedido de movimento, ou seja, esse subsistema recebe como entrada o que se deseja realizar com o braço, por exemplo mudar ele de uma posição e orientação para outra. Após isso, o nó *move_group* é acionado e calcula a trajetória que deve ser realizada. Por fim é gerado um plano de resposta.

4.3.3 Planning Scene Monitor

É um objeto utilizado para armazenar as informações do mundo em volta do robô, seja por meio de informações de sensores a laser ou imagens de câmeras. É extremamente útil quando se trabalha com atividades de *pick and place*. No manipulador do trabalho são utilizadas as configurações padrões, como se pode verificar em (21).

4.3.4 Planejamento de Trajetória

Esse pacote age nas informações de trajetórias que são fornecidas pelo planejamento de movimento e aplica algum algoritmo de otimização, otimização por menor deslocamento angular de junta, menor tempo, entre outros (tais algoritmos podem ser encontrados em (22)), ou aplica as restrições de movimento, tais como: velocidades, acelerações, ângulos e esforços máximos.

4.4 Sistemas Auxiliares

Uma das grandes vantagens do ROS é a sua modularidade de sistemas, em outras palavras, é relativamente simples acoplar e desacoplar sistemas. Como exemplo disso, são adicionados ao sistema um par de câmeras e de sensores a laser, buscando auxiliar o conhecimento observacional do terreno em que está sendo empregado o veículo, além de permitir a aplicação de métodos de mapeamento e deslocamento autônomo.

Ambas as câmeras são representadas por um cubo de um centímetro de aresta, está é uma representação visual simplificada já que a geometria de uma câmera de pequenas dimensões não impacta no desempenho do projeto. Uma das câmeras é adicionada na parte superior da garra, de forma a facilitar a observação do objeto que está sendo manipulado, como mostrado na Figura 18. E uma outra na parte frontal do veículo, como é mostrado na Figura 19, essa câmera é utilizando tanto para a visualização da imagem propriamente dita mas também para a futura implementação de um mapa em três dimensões. E é por conta disso que as câmeras adicionadas são do tipo *Kinect* e portanto simulam um hardware com capacidade 3D, utilizando as suas capacidades de cores e de profundidade.

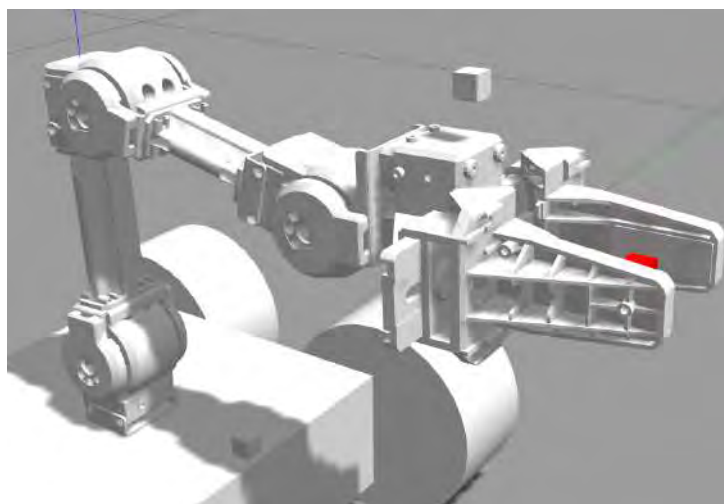


Figura 18 – Modelo de câmera acoplada na parte superior da garra

Além das câmeras é implementado também um sensor a laser no robô. Devido a presença do manipulador não é possível utilizar apenas um sensor que apresente um ângulo de atuação de 360°, portanto são utilizados 2 sensores hokuyo, pois é um sensor comercial facilmente encontrado, com cada um possuindo um ângulo de atuação de 190°. Na Figura 20 a parte em azul indica o raio do laser e se percebe que existe apenas uma pequena região que não é englobada, mas ela é relativamente próxima ao veículo e por conta disso não apresenta problemas.

Apesar da implementação de dois sensores aumentar a faixa de operação dos sensores ele cria a necessidade de um pacote adicional, que possa realizar o *matching* entre os dados dos sensores, se completando e também interpretando de forma correta quando há

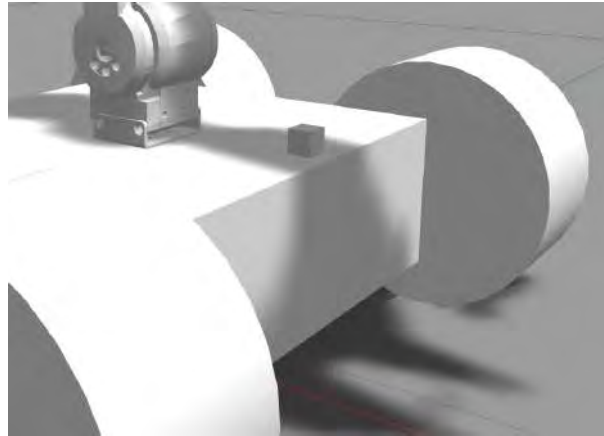


Figura 19 – Modelo de câmera acoplada na base do veículo

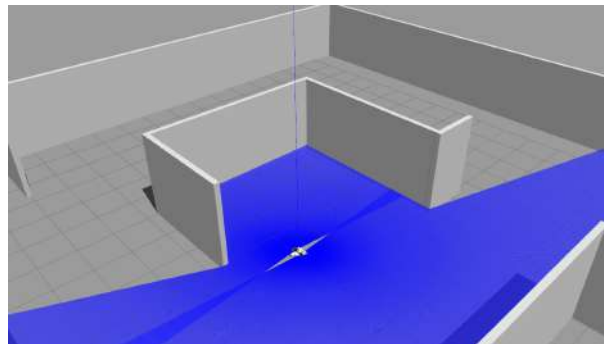


Figura 20 – Modelo após a integração dos sensores a laser

sobreposição de dados. Para tal se utiliza o pacote *ira_laser_tools*, mais especificamente o nó *multi_merger* para tratar os dados dos dois sensores e fornecer como saída os dados em apenas um tópico, tal como é explicado em (23).

4.5 Árvore de Funções de Transferências

O ROS utiliza o pacote *transfer_function* para relacionar as coordenadas de cada corpo com o passar do tempo. Esse pacote mantém uma estrutura em árvore para relacionar cada elemento entre si e também com o referencial global da simulação, além de indicar qual é o tópico que publica essas informações para cada um dos corpos. Da Figura 21 até a Figura 25 é mostrada a estrutura da simulação.

Na figura 21 são mostradas as relações entre o *odom*, que é a referência entre a base e o referencial fixo, e um corpo auxiliar, sem propriedades de massa e de inércia, denominado *base_footprint*, esse corpo representa a projeção da base no solo (isso é uma boa prática que pode ser útil em diversas aplicações). O chassi do robô foi definido com base nesse corpo fictício e a relação entre ambos é realizada por meio do tópico *robot_state_publisher*, que nada mais é do que o canal de comunicação entre juntas definidas nos arquivos URDF.

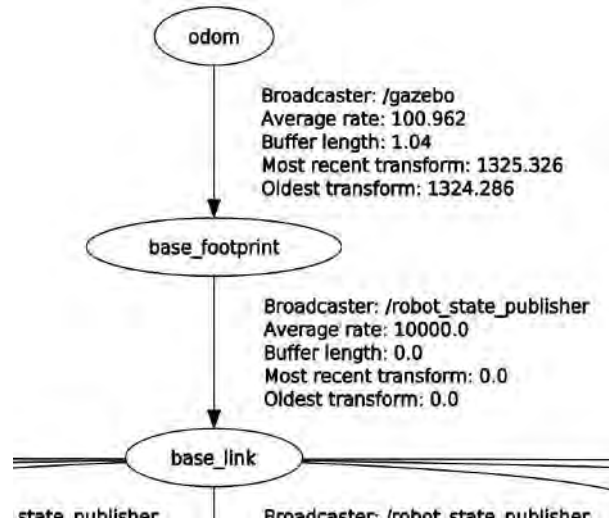


Figura 21 – Relação entre o referencial estático e a base do robô

Devido ao fato de existirem muitos corpos definidos a partir da base do robô essa análise é definida em duas imagens separadas, Figura 22 e 23. Na Figura 22 se tem a relação entre a base e as rodas do veículo. Enquanto na Figura 23 é mostrado que ambos os sensores são definidos a partir da da base (*front_hokuyo* e *rear_hokuyo*), assim como a primeira junta de rotação do manipulador.

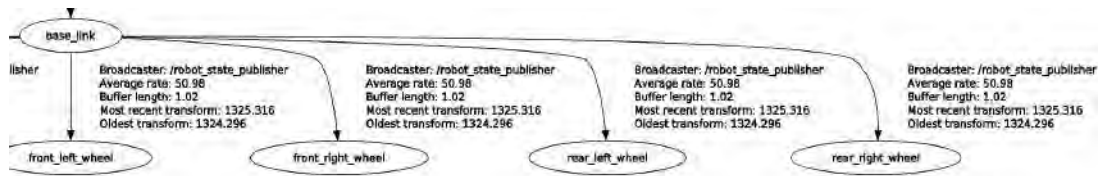


Figura 22 – Relação entre o referencial da base e as rodas

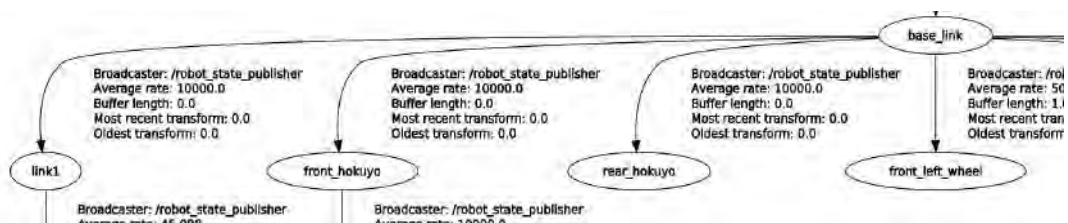


Figura 23 – Relação entre o referencial da base e sensores e junta do manipulador

A figura 24 estabelece que a câmera da base do robô, aquela que é efetivamente é usada na geração do mapeamento, além de um outro corpo nomeado como *camera_depth_link* necessário para a geração de imagens em 3 dimensões. Além disso, nessa figura estão relacionadas as juntas do manipulador, sendo a garra o *link5*.

Na figura 25, são mostrados as últimas correlações geométricas desse modelo. Os corpos *end_effector_link*, *gripper_link* e *gripper_link_sub* são necessários para a

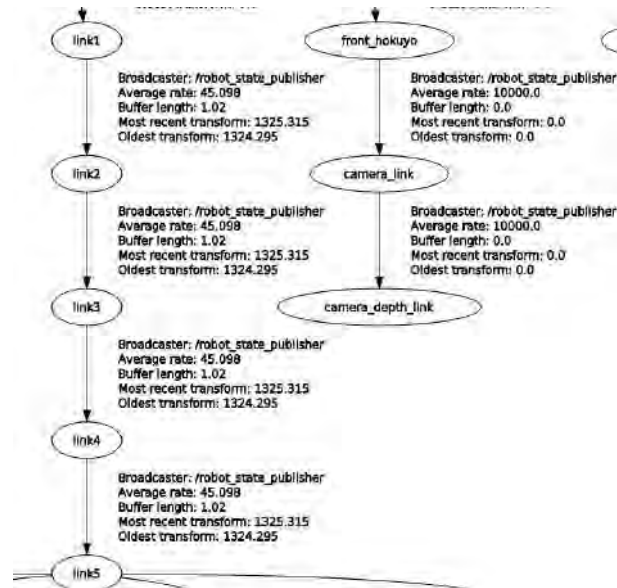


Figura 24 – Relações entre os diversos corpos do manipulador e do sensor laser frontal

construção da garra em formato URDF, já que sua geometria teve de ser simplificada, enquanto o corpo *camera_arm_link* se refere a câmera instalada na parte superior da garra e presente na figura 18.

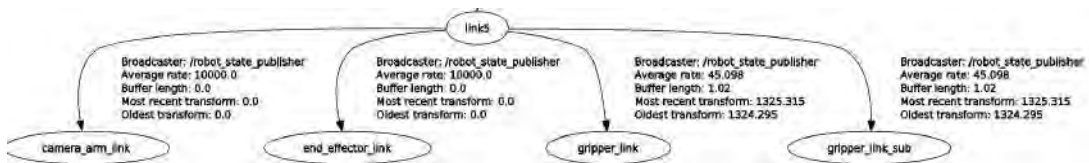


Figura 25 – Relações entre a garra e os corpos definidos a partir dela

4.6 Mapeamento

Dentre os diferentes métodos de mapeamento possíveis é utilizado o *Gmapping*, pois como é discutido nos trabalhos de (14) a utilização de dados de odometria do robô, tais como as velocidades angulares das rodas, permite criar um mapa com um nível de precisão melhor.

O pacote do *Gmapping* precisa ter como entrada duas informações: os dados do sensor a laser, que precisam ser fornecidos em um tópico único independentemente se há vários sensores desse tipo, e as funções de transferência entre o robô e o referencial da origem, informação fornecida pela relação entre os nós *odom* e *base_footprint* mostrado na Figura 21.

As saídas desse nó são: um tópico de entropia do mapa, que nos fornece um indicador da precisão do mapa, e um tópico contendo as informações do mapa propriamente dito.

Caso o sistema funcione corretamente, espera-se obter um resultado semelhante ao da Figura 26.

Existem ainda diversos parâmetros que podem ser adaptados para fornecer melhores configurações do mapa, variando de acordo com especificações técnicas e de processamento da máquina. No presente trabalho utiliza-se os parâmetros definidos como padrão, que podem ser encontrados em (24), já que os resultados obtidos com eles foram satisfatórios.

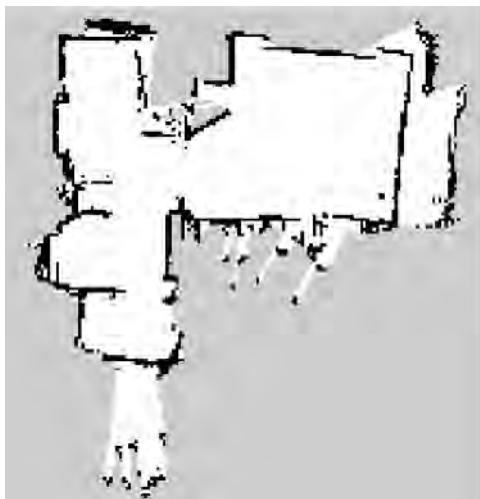


Figura 26 – Exemplo do mapa gerado pelo método *Gmapping* (6)

4.7 Deslocamento Autônomo

É um pacote que fornece uma aplicação de deslocamento autônomo de uma base móvel a partir da entrada de um ponto objetivo (*goal*). O nó central da aplicação é o *move_base* que cria um plano global e um local para atingir o objetivo formados a partir de dois mapas, um local e um global, de inflação, que servem para evitar obstáculos.

Na Figura 27 se tem uma imagem que resume o funcionamento desse nó. O sistema recebe como entradas as informações do mapa, do laser, dados de odometria e as funções de transferência (o *amcl* é um método para melhorar a precisão dessas transformações com o ambiente mas que não é essencial para o funcionamento do nó). Além disso, o operador ou algum recurso interno criado fornece ao nó a pose (posição e orientação) que se deseja atingir.

Na arquitetura interna dos nós ele utiliza as informações do mapa e do scan para criar mapas de custo (*local_costmap* e *global_costmap*), que criam uma inflação nos obstáculos detectados de forma que o veículo planeje seu movimento passando a uma distância de segurança deles. Esses parâmetros podem ser ajustados de forma a aumentar ou diminuir essa distância de segurança, atribuindo um custo ao movimento de movimento em cada região do mapa.

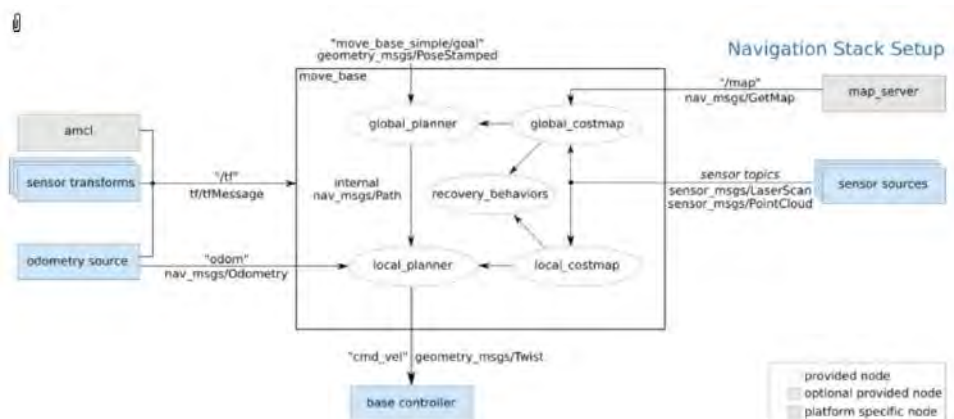


Figura 27 – Funcionamento do nó *move base* (7)

Na Figura 28 se tem um exemplo de mapa de custo, no qual as regiões totalmente escuras não são permitidas para o robô, ou seja, apresentam um valor máximo (252). Já a região mais clara apresenta um custo zero para o deslocamento. Enquanto isso a região intermediária varia de acordo com uma função que varia de acordo com a distância entre as duas regiões.

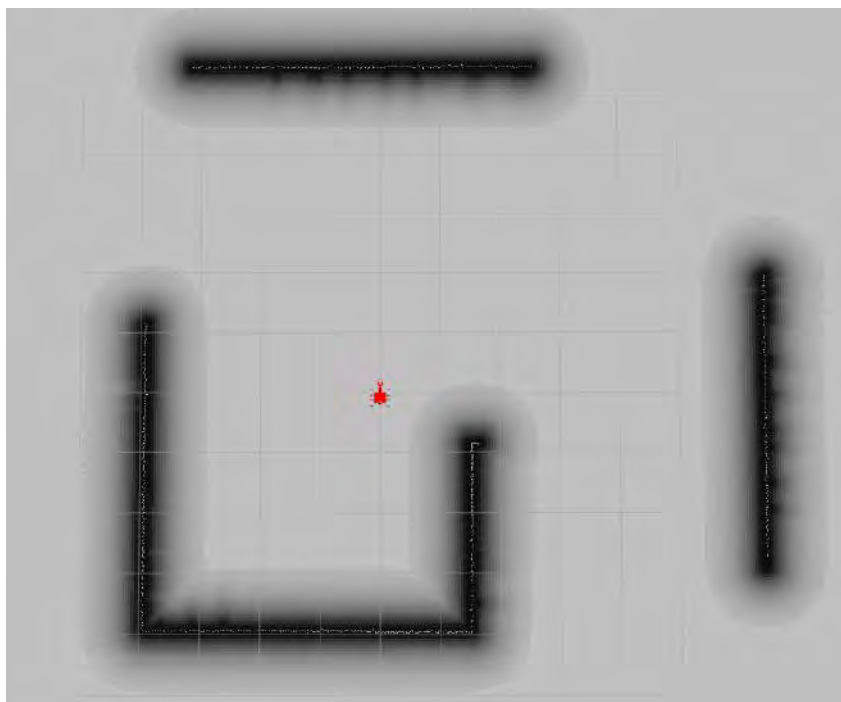


Figura 28 – Mapa de custo criado pelo nó *move_base*

A caracterização do movimento propriamente dito pode ser separada em dois planejadores, o global e o local. O planejador global possui um funcionamento simples, ele calcula uma curva de deslocamento do ponto inicial até o final com o menor custo possível, levando e conta dos parâmetros dos mapas de custo global (8). Um exemplo de utilização do planejamento global é mostrado na Figura 29, na qual é criado um caminho

de referência, representado pela linha amarela, entre o ponto inicial e o final para o robô.

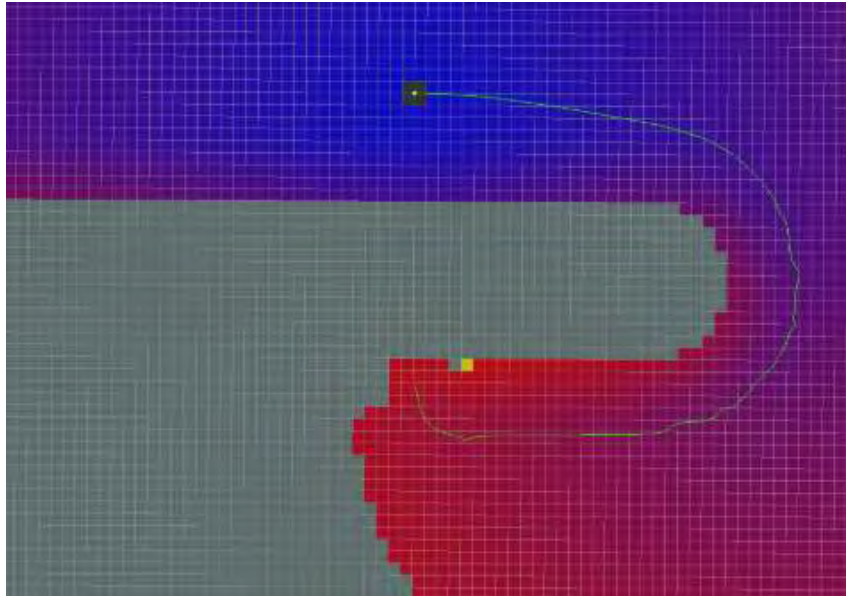


Figura 29 – Exemplo de planejamento global de trajetória (8)

Entretanto, o planejamento global não é responsável por efetivamente mover o robô, essa função é reservada ao planejador local (*local_planner*), que efetivamente age como o controlador de movimento. Ele recebe informações do objetivo, de uma parcela da trajetória global, apenas uma região limitada, e também do mapa de custo local, em cada intervalo de tempo de funcionamento do controlador. A partir dessas informações ele segue o seguinte algoritmo:

- Cria uma série de pares de velocidades, uma angular e uma linear longitudinal, para atingir o objetivo.
- Calcula-se o que acontece com o veículo quando esse comando de velocidade é aplicado por um período de tempo.
- Calcula-se qual é o melhor par de velocidades a partir de uma série de parâmetros, tais como: proximidade com obstáculos, proximidade com o objetivo, proximidade com o planejamento global e velocidade, descartando sempre as trajetórias que colidem com objetos.
- Escolhe-se o melhor conjunto de velocidades.
- Repete o processo para a próxima interação.

Existem uma série de parâmetros desse método que podem ser alterados para maximizar o desempenho do planejador local. Seguindo como referência os resultados e recomendações de (16) e (8) chega-se a um grupo de configurações presente no anexo A do trabalho.

5 SIMULAÇÕES

Partindo do modelo construído é possível realizar uma série de testes a respeito das funcionalidades do sistema em diferentes situações: deslocamento da base controlada a distancia, testes dos sistemas de controle do manipulador, mapeamento e deslocamento autônomo. Sendo que os dois últimos casos são separados em um ambiente interno, presente na Figura 30, e um externo, representado na Figura 31.

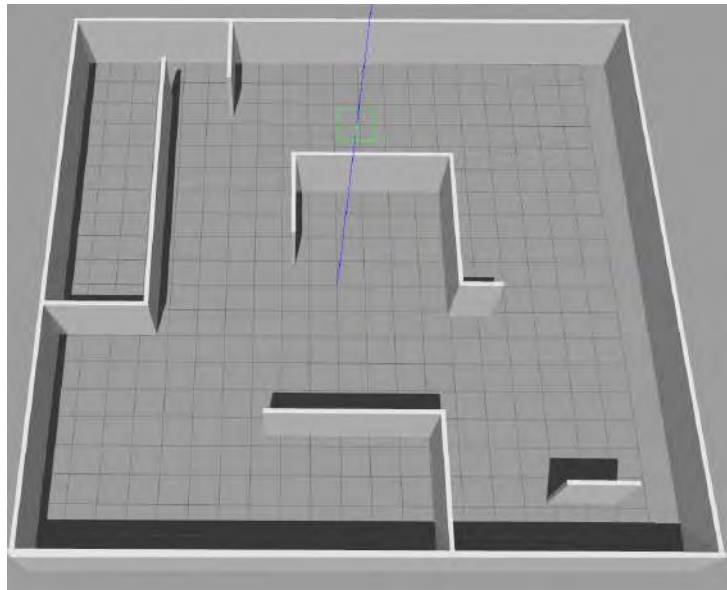


Figura 30 – Ambiente interno no qual são realizadas as simulações

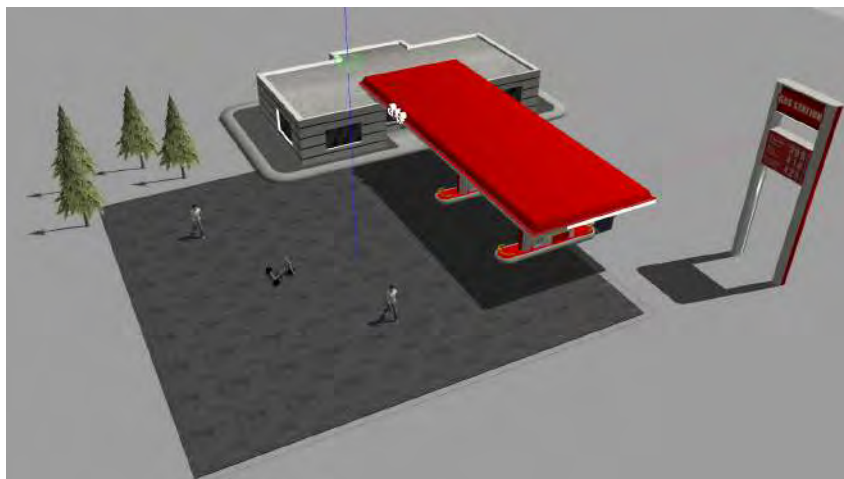


Figura 31 – Ambiente externo no qual são realizadas as simulações

5.1 Deslocamento teleoperado em ambiente interno

Essa é uma simulação que deve ser realizada para testar se as interações do veículo com o solo estão coerentes. Para tanto, desenvolveu-se um nó para que os comandos de velocidade sejam dados por um *joystick* que se comunica com o veículo por meio de uma comunicação *bluetooth*. Essa comunicação não é a ideal, seria mais adequado usar um controle por meio de ondas de rádio, mas isso pode ser empregado apenas na construção e desenvolvimento de um robô real, para a simulação ele pode ser utilizado sem interferências no desempenho.

Além dos comandos de velocidade, um botão de segurança também é utilizado e portanto o veículo só se desloca quando o botão está pressionado. E pensando em situações mais extremas, nas quais seja necessário um aumento substancial de velocidade, é adicionado um botão que dobra o valor de velocidade longitudinal comandado pelo operador.

O estado inicial de simulação é mostrado na Figura 32, na qual o robô é iniciado na origem do sistema global de referência.

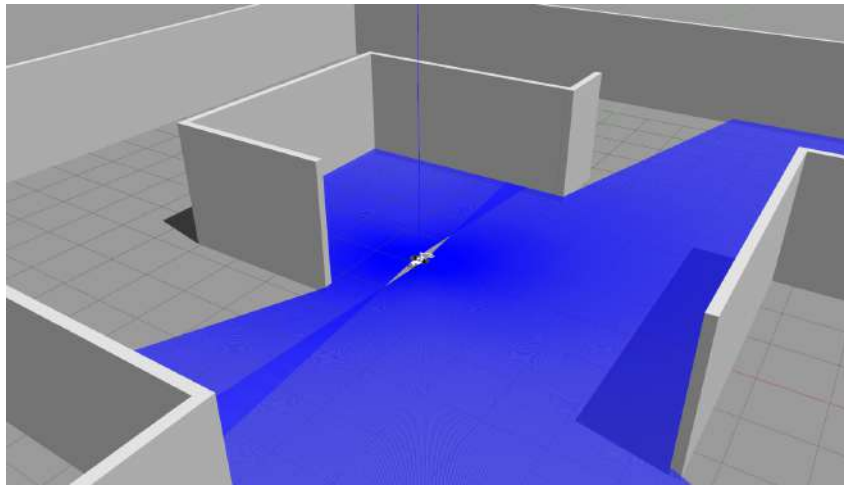


Figura 32 – Estado inicial de simulação em ambiente interno

Existe no ROS uma ferramenta que permite analisar o fluxo de informação chamada *rqt_graph*, ela é essencial para o *debug* e entendimento do funcionamento de sistemas. Na Figura 33 estão indicados os nós ativos do sistema e seus respectivos tópicos de comunicação. O nó *joy_node* envia os comandos do joystick para um interpretador denominado *teleop_twist_joy*. Esse nó, por sua vez, relaciona essa informação, dos comandos do joystick, com um comando de velocidade definido em um arquivo *.config*. Esse comando de velocidade é enviado para o simulador e então publicado no robô, por meio do tópico *joint_states*.

Para validar as configurações de atrito e de *driver* utilizadas são realizados dois testes nos quais o *joystick* envia comandos muito próximos a uma entrada em graus de

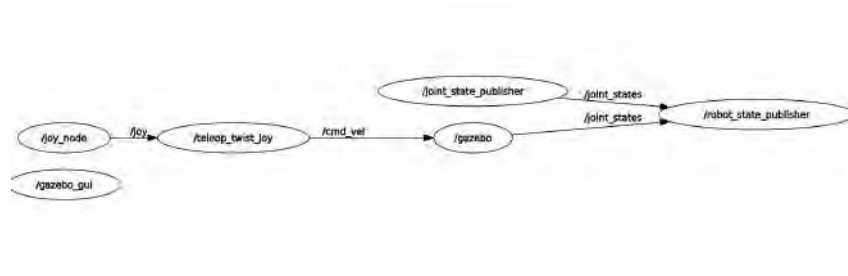


Figura 33 – Nós ativos e tópicos de comunicação entre eles

velocidade longitudinal e angular (o comando não chega a efetivamente ser um degrau porque depende do tempo de acionamento do botão analógico do *joystick*).

Nas Figuras 34 e 35 estão indicadas as respostas de aceleração e frenagem para um degrau de velocidade de 0.5 m/s. Percebe-se que o sistema além de reagir de maneira extremamente veloz não possui *overshooting* e nem deslizamento na frenagem para esse valor de velocidade.

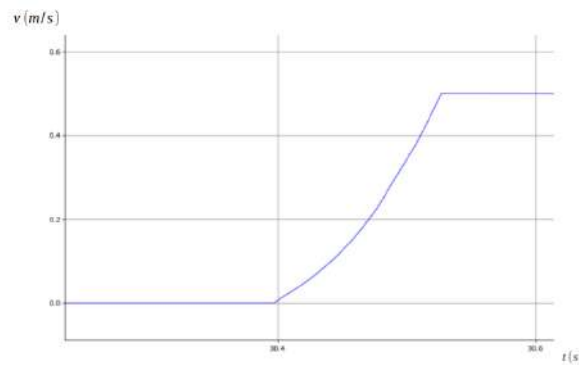


Figura 34 – Resposta a uma entrada degrau de velocidade longitudinal

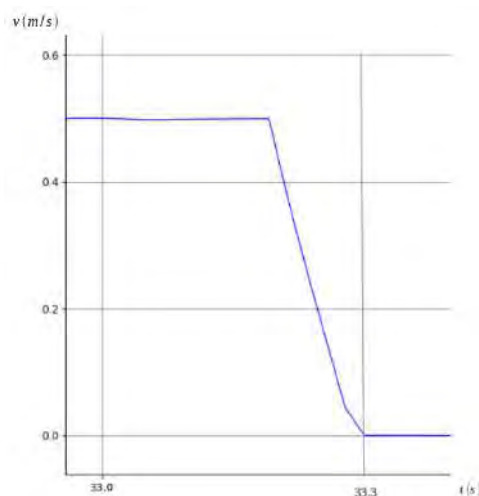


Figura 35 – Resposta a uma entrada degrau de velocidade longitudinal

O sistema também possui uma função de aumento da velocidade, um modo turbo para o sistema, que pode ser dada com o acionamento de de outro botão no controle remoto, situação na qual a velocidade máxima sobe para 1.0 m/s. Apesar das Figuras 36 e 37 mostrarem um resultado satisfatório houve movimentações no manipulador que está acoplado a base devido a alta aceleração o que, dependendo da disposição dele, pode gerar problemas de dirigibilidade no veículo. Pois quando o braço estiver uma posição mais retraída seu momento de inércia não terá tanto efeito sobre o sistema, contudo se ele estiver em uma posição esticada pode causar movimentos de *pitch* no veículo.

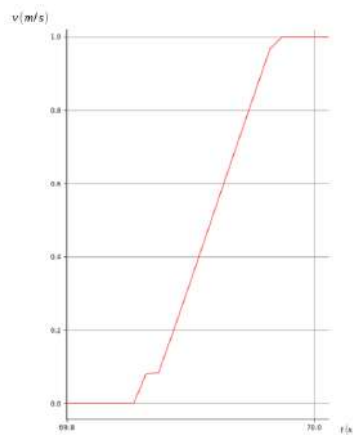


Figura 36 – Resposta a uma entrada degrau de velocidade longitudinal com o modo turbo acionado

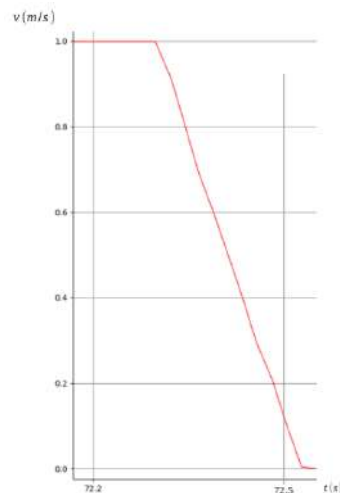


Figura 37 – Resposta a uma entrada degrau de velocidade longitudinal com o modo turbo acionado

Realizando os mesmos testes para a velocidade angular, cujo valor de degrau é 0.2 rad/s, são obtidos os resultados presentes nas Figuras 38 e 39.

Primeiramente, percebe-se que o sistema possui um tempo de resposta mais elevado para atingir o valor desejado na aceleração, mas ainda assim esse quesito da resposta está

adequado aos propósitos do trabalho. Entretanto, é possível perceber que o sistema não atinge a velocidade desejada de 0.2 rad/s e também possui certas variações de velocidade angular, que são pequenas mas mostram que o sistema está oscilando.

Isso se deve a dificuldades no próprio simulador em retratar uma situação em que há esse tipo de deslizamento. Pois é possível perceber que os pontos de contato de uma roda com o solo podem mudar de interação para interação, de acordo com uma rotina interna do simulador, e isso leva a diferentes momentos aplicados na base. Além disso, segundo a discussão realizada em (25), o método de cálculo do atrito realizado pelo simulador pode apresentar esse comportamento oscilatório em situações em que há deslizamento e rolamento, devido a erros numéricos.

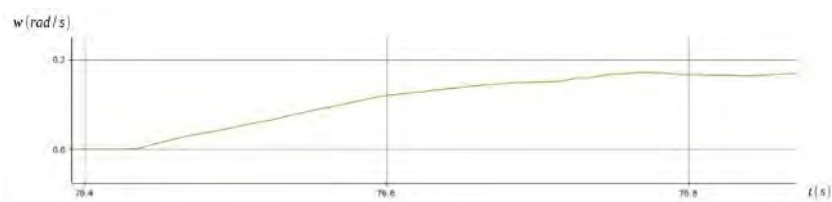


Figura 38 – Resposta a uma entrada degrau de velocidade angular



Figura 39 – Resposta a uma entrada degrau de velocidade angular

Apesar desses problemas apresentados da velocidade angular eles não interferem de forma significativa o estudo do presente trabalho. Portanto, é possível utilizar essa configuração do *driver* da base móvel para as demais simulações.

5.2 Testes dos Sistemas de Controle do Manipulador

Ao lançar os pacotes do ROS *MoveIt*, responsável pelo controle do braço, e o *Rviz* para a visualização se têm o resultado da Figura 40. No projeto foram desenvolvidas 3 maneiras distintas de manipular o braço.

5.2.1 Manipulação por meio da Interface Gráfica

A primeira consistem em alterar a posição da esfera que está no centro da garra, ponto de trabalho, para a posição requerida utilizando o próprio cursor do computador para realizar tal tarefa. Após o local ser escolhido basta executar a trajetória por meio

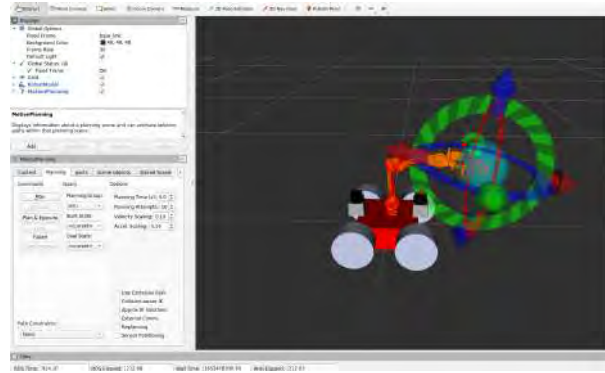


Figura 40 – Janela inicial para o controle do manipulador

dos comandos *Plan and Execute*. A Figura 41 e 42 retratam a simulação e a janela *Rviz*, respectivamente, na situação em que o braço se encontra em uma posição inicial, representado pela cor vermelha na Figura 42, enquanto é escolhida uma posição final para ele, representado pela cor laranja.

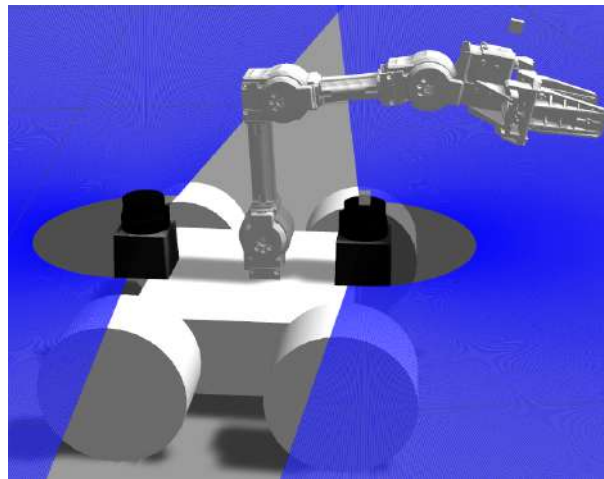


Figura 41 – Posição inicial do manipulador

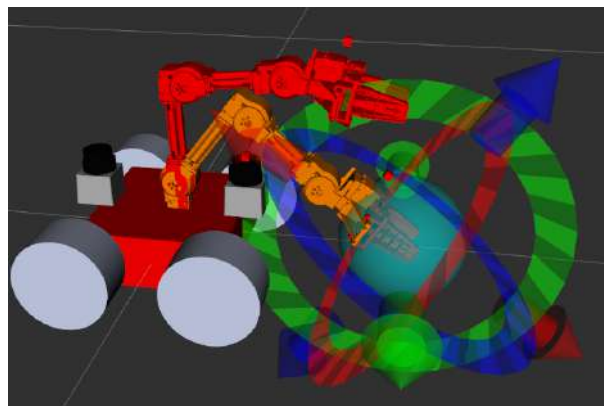


Figura 42 – Janela de visualização com o manipulador na posição inicial e a final já determinada

Após o comando para executar se tem como resultado na simulação as Figuras 43 e 44, respectivamente no Gazebo e no *Rviz*, mostrando que o braço atingiu a sua posição final.

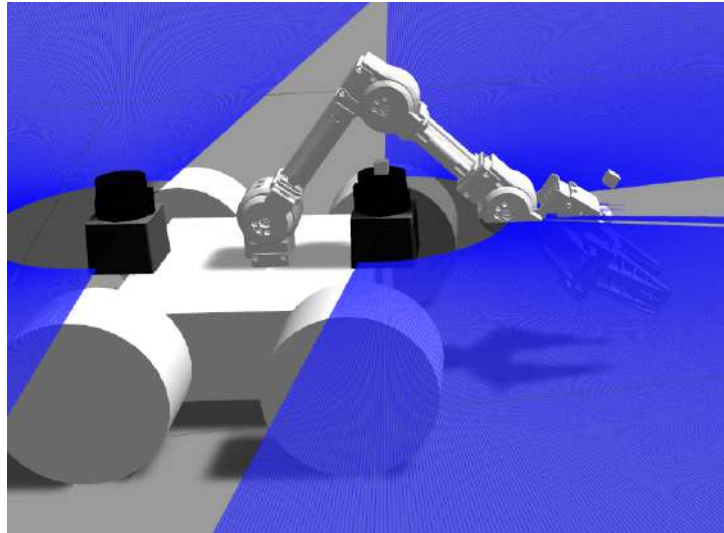


Figura 43 – Posição final do manipulador

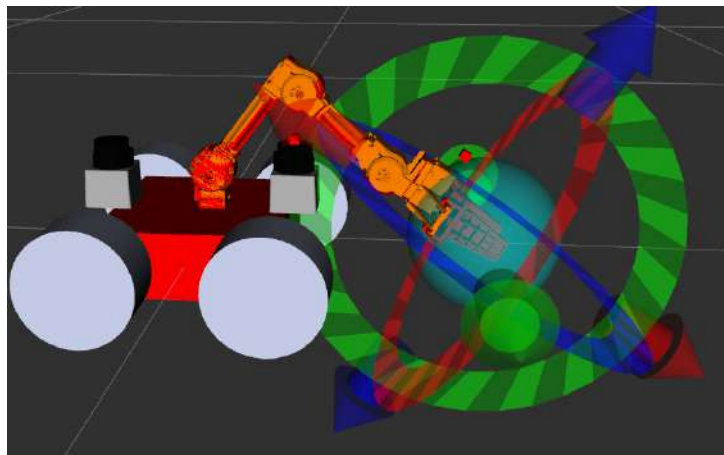


Figura 44 – Janela de visualização na posição final

Essa forma de manipulação do braço tem a grande vantagem de ser simples, podendo ser realizada de maneira gráfica e rápida a uma grande gama de movimentos diferentes. Entretanto, ela possui um problema relacionado a precisão pois realizar pequenos ajustes de movimento nesse interface é uma tarefa complicada.

5.2.2 Banco de Dados

Para contornar a questão da precisão foi desenvolvido uma comunicação com o banco de dados do sistema de controle. Sendo assim é possível estabelecer pré posições de trabalho para o manipulador. O banco de dados utilizado é o *warehouse_mongo* e sua interface é mostrada na Figura 45.

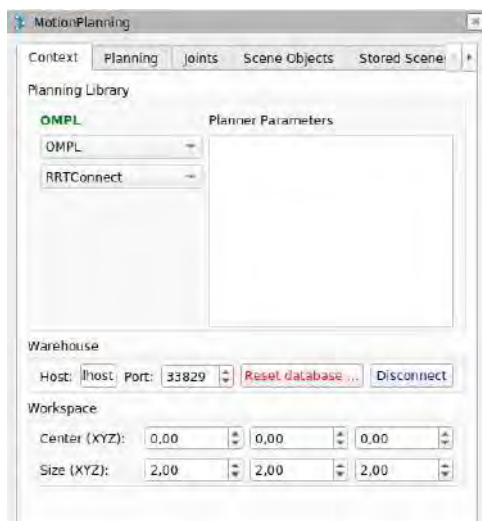


Figura 45 – Interface de comunicação com o banco de dados do MoveIt

A detonação de explosivos seria uma aplicação prática para esse sistema. Pois o veículo leva o explosivo de detonação em uma posição fixa e já conhecida, portanto bastaria salvar essa posição inicial do manipulador, como mostrado na Figura 46. Da mesma forma é possível salvar uma posição final para soltar esse explosivo, ou pelo menos deixá-lo em uma posição próxima para isso, como mostrado na Figura 47. Esse método é muito indicado para situações de pegar e soltar, pois é rápido e não demanda muito do operador. Entretanto, possui uma clara limitação de só poder atingir os objetivos previamente estabelecidos.

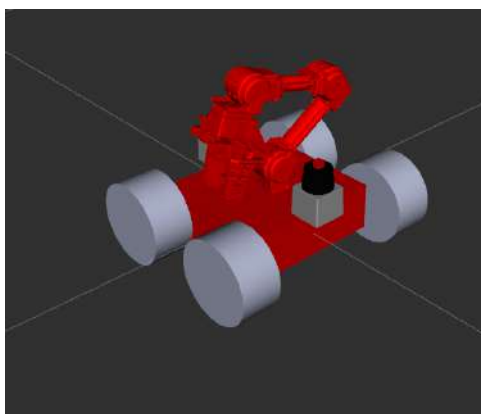


Figura 46 – Posição salva para pegar o explosivo

5.2.3 Controle por meio de Joystick

Esse sistema foi introduzido para permitir que, caso o operador deseje, ele possa controlar o manipulador junta a junta por meio de um controle remoto. Sendo uma ferramenta mais simples de se utilizar do que o controle por meio da interface gráfica, apesar de um pouco mais lenta. Na Figura 48 é mostrado como age esse nó na estrutura de

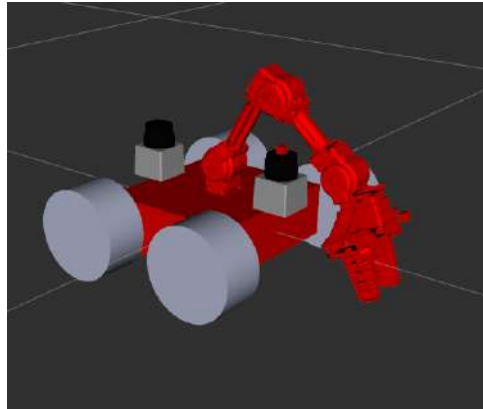


Figura 47 – Posição salva para soltar o explosivo

comunicação do sistema. Enviando por meio do tópico do controlador do braço comandos de movimento.

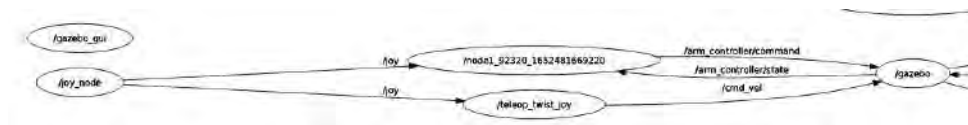


Figura 48 – Conjunto de nós e tópicos para o controle junta a junta

5.3 Componentes Auxiliares

É importante também mostrar as ferramentas de visualização das câmeras, feitas por meio do *Rviz*. Para tanto, em um ambiente externo mostrado na Figura 49, enquanto o posicionamento do robô é mostrado na Figura 50.

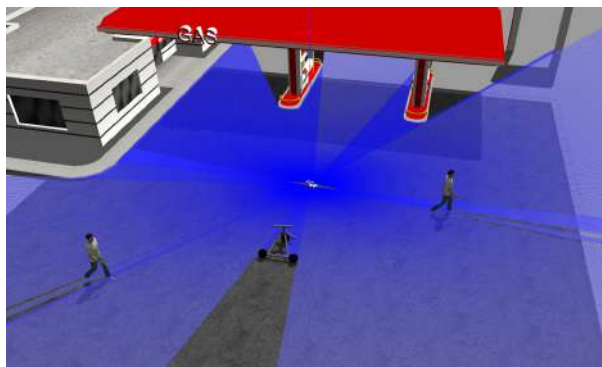


Figura 49 – Ambiente externo para o teste da câmera

A imagem da câmera da base é mostrado na Figura 51, enquanto que aquela montada na parte superior do braço é mostrada na Figura 52. Essas imagens podem auxiliar no controle do robô, tanto no controle da base móvel quanto o do manipulador.

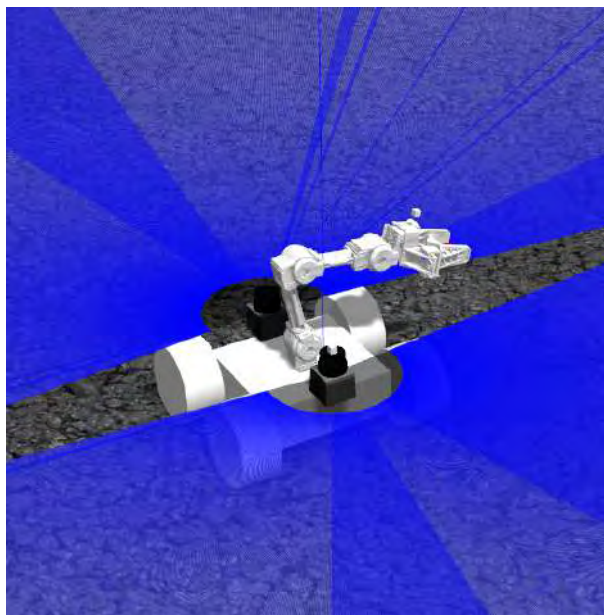


Figura 50 – Posicionamento do robô

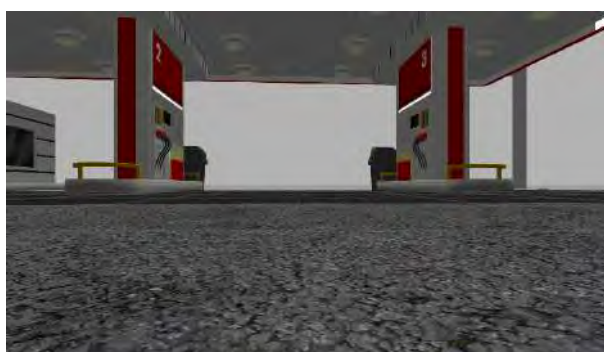


Figura 51 – Imagem da câmera montada na base do veículo



Figura 52 – Imagem da câmera montada na parte superior da garra

5.4 Mapeamento

Essas simulações têm o objetivo de criar um mapa do ambiente, tanto em duas como em três dimensões para que esses dados sejam usados posteriormente para uma análise mais precisa do ambiente por um operador quando para o deslocamento autônomo

do veículo. As simulações são separadas em mapeamento em ambiente interno e externo.

5.4.1 Ambiente Interno

Nesse ambiente há a vantagem de receber as reflexões dos sensores a laser de forma mais completa, pois em geral esses sensores possuem uma distância máxima de aplicação. O ambiente escolhido é o da Figura 53 e resultado do mapeamento em três dimensões é mostrado na Figura 54. Ele é obtido fazendo com que o veículo se movimente pelo ambiente, de forma com que o recurso de profundidade da câmera e o laser captem as paredes.

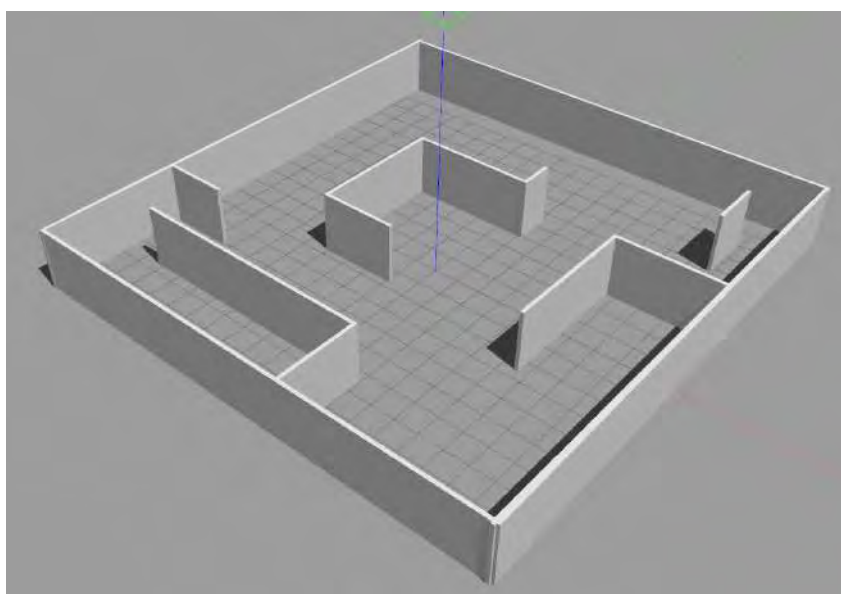


Figura 53 – Ambiente interno a ser mapeado

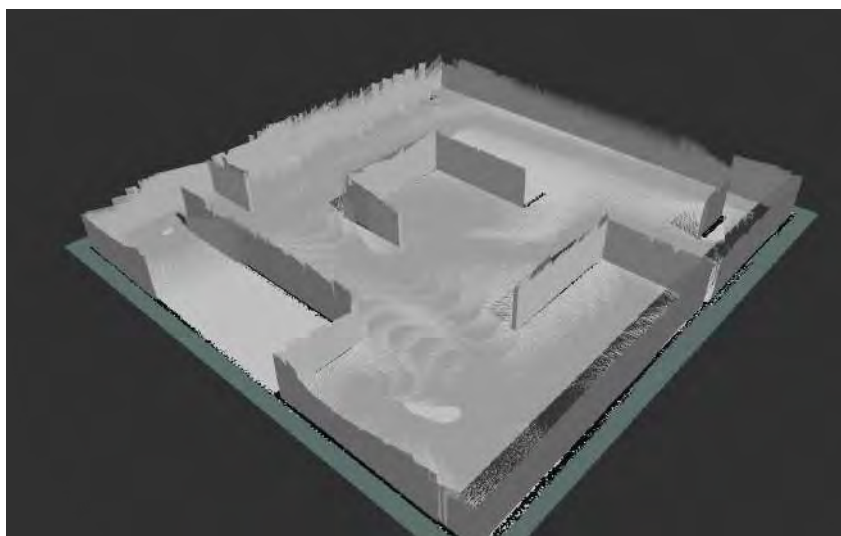


Figura 54 – Resultado do mapa em três dimensões em um ambiente interno

O resultado obtido é satisfatório pois permite a um operador tem uma análise do fidedigna do ambiente. É importante também conseguir uma boa qualidade no mapa em duas dimensões, pois é esse que efetivamente será utilizado nos parâmetros de custo para o deslocamento autônomo. Na Figura 55 se tem o resultado deste mapa não apresenta distorções ou problemas de superposição, como os apresentados em (14).

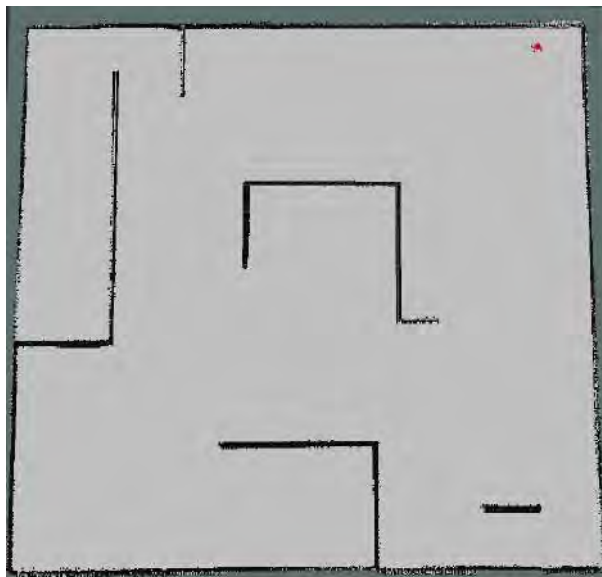


Figura 55 – Resultado do mapa em duas dimensões em um ambiente interno

5.4.2 Ambiente Externo

Procedendo de forma análoga para o ambiente externo se obtém o mapa em três dimensões da Figura 56. Fica evidente que a qualidade do mapa é menor em comparação ao interno (Figura 54). Entretanto, isso se deve as dificuldades de mapeamento em um ambiente externo, no qual o recebimento das reflexões do laser é limitada. E mesmo com a diminuição da qualidade ele ainda fornece informações detalhadas sobre o ambiente em questão.

O mesmo não se pode dizer do mapa em duas dimensões, Figura 57, já que não é possível ter uma noção clara dos obstáculos nesse mapa. Contudo, isso é uma limitação que não está relacionada ao ambiente analisado (se é interno ou externo), mas sim devido aos obstáculos presentes, representados pelos pontos escuros no mapa.

5.5 Deslocamento Autônomo

5.5.1 Ambiente Interno

Na Figura 58 se tem o veículo na sua posição inicial. A seta vermelha indica a posição final que se quer atingir com o veículo. É importante notar que essa posição não

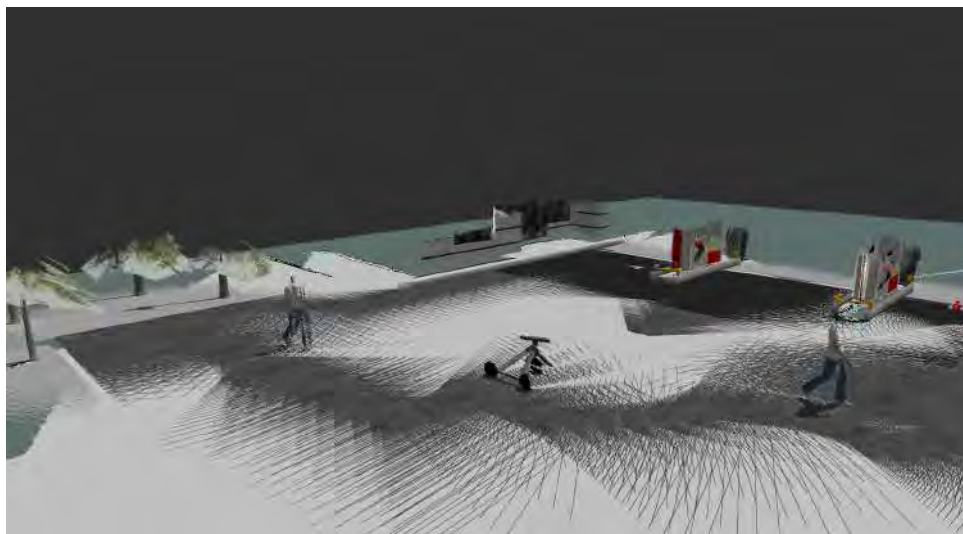


Figura 56 – Resultado do mapa em três dimensões em um ambiente externo



Figura 57 – Resultado do mapa em duas dimensões em um ambiente externo

precisa estar do mapa conhecida, pois ele atualiza tanto os planejadores quanto a próprio mapa com o passar da simulação a uma frequência definida.



Figura 58 – Posição inicial e posição final do objetivo

Na Figura 59 está presente um QR Code que remete a um vídeo referente ao teste

de deslocamento interno da simulação. No vídeo, a linha em amarelo, próximas ao robô, representam o planejador global enquanto a linha preta representa o planejador local. Percebe-se que o veículo evitou ficar próximo da parede e portanto os parâmetros de custo dos mapas foram bem escolhidos e ao final da ação, o veículo atinge a posição desejada com uma pequena tolerância permitida.



Figura 59 – QR code referente ao deslocamento do veículo em ambiente interno

5.5.2 Ambiente Externo

Procedendo de forma análoga ao teste de deslocamento em ambiente interno se tem as posições inicial e final desejada na Figura 60. Mais uma vez a posição requerida foi colocada em uma região a princípio desconhecida do mapa.

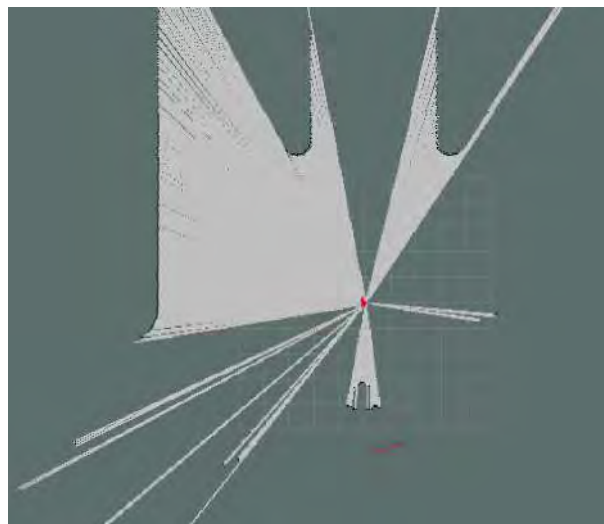


Figura 60 – Posição inicial do e posição objetivo

Mais uma vez é utilizado um QR Code para mostrar os resultados obtidos nesse teste. Com base no vídeo referente a Figura 61, percebe-se que o veículo foi capaz de evitar o obstáculo representado pelo triciclo de forma suave, passando a uma distância segura deste. E além disso, mesmo com informações menos completas, causadas pelo ambiente

ser externo, o pacote de deslocamento autônomo conseguiu performar de forma muito semelhante ao teste realizado em ambiente interno.



Figura 61 – QR code referente ao deslocamento do veículo em ambiente externo

6 MONTAGEM DOS SISTEMAS DO VEÍCULO

O sistema ROS não se trata apenas de uma ferramenta de simulação, tendo na verdade como principal objetivo a aplicação em sistemas que trabalham em tempo real. Para implementar esse sistema ao veículo, o rover Lynxmotion 4WD1, é preciso modificar os sistemas embarcados que o veículo possui de fábrica de forma que ele seja capaz de suportar o sistema operacional em questão.

6.1 Componentes do Veículo

De um panorama geral o veículo tem os componentes presentes na Figura 62, na qual as setas pretas representam fluxo de informação entre as partes.

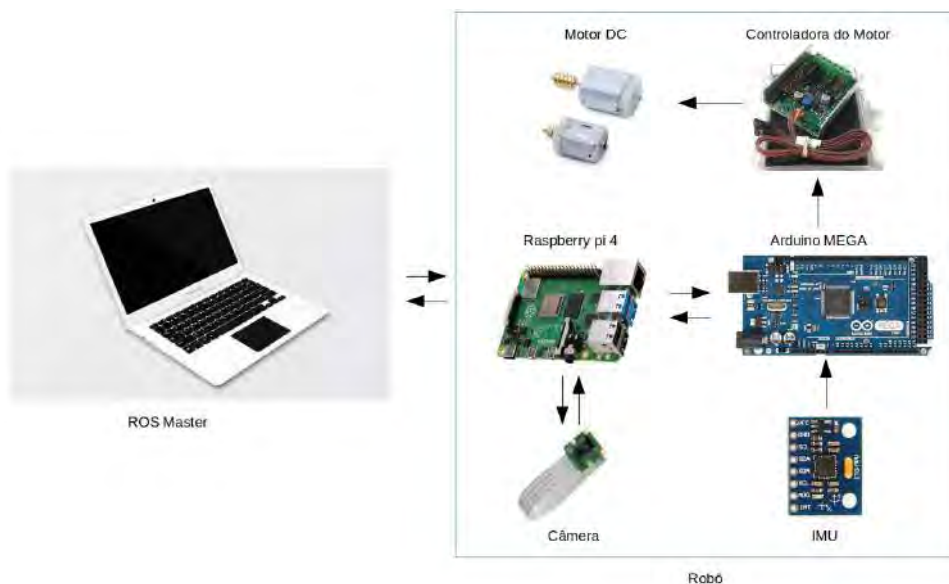


Figura 62 – Esquema geral dos componentes do veículo

O papel de cada um dos componentes, bem como as suas configurações para que as comunicações ocorram corretamente devem ser discutidas separadamente.

6.1.1 ROS Master

Um computador já com o sistema ROS (neste caso a versão Noetic referente ao Ubuntu 20.04) instalado deve ser usado para se comunicar com o robô. Esta comunicação pode ser realizada quando ambos os computadores, o *ROS Master* e o Raspberry (o computador embarcado no robô), estão conectados na mesma rede, um modelo de como proceder com esta conexão pode ser encontrado em (26), entretanto para que ocorra também o compartilhamento de mensagens é preciso realizar a alteração presente em (27).

Isso abre um grande número de possibilidades, pois todos os tópicos de mensagens podem ser acessados e alterados tanto pelo ROS Master quanto pelo computador embarcado. Portanto, os comandos de movimento por exemplo podem ser enviados diretamente para o robô e os dados de sensores, posição e situação do terreno podem ser lidas em tempo real.

Além disso, essa forma de conexão permite que sejam exploradas outras aplicações, pois um Mestre pode gerenciar diversos robôs e estes por suas vez podem também trocar informações entre si. Possibilitando aplicações de enxames de drones e comboios de veículos autônomos.

Há ainda a possibilidade de que o Mestre do sistema se comunique diretamente com o microcontrolador do veículo, neste caso um arduino mega, também por meio de uma rede interna. No entanto, nesse método o veículo não iria possuir nenhum grau de autonomia (que pode ser adicionado em um projeto futuro como mostrado nas simulações do Capítulo 4), além de ficar extremamente dependente da rede de comunicação, que pode falhar dependendo da situação, e também invalidaria situações muito complexas tais como comboios de veículos, devido ao alto fluxo de informações que pode sobrecarregar o Mestre.

6.1.2 Raspberry pi

A central lógica do veículo precisa suportar um sistema operacional compatível com o ROS utilizado, sendo portanto necessário ser capaz de trabalhar com o Ubuntu 20.04. Dentre as opções do mercado, o modelo Raspberry pi 4 de 8 gigabytes de RAM é aquele que apresentam o maior custo benefício quando comparadas às placas Jetson Nano, da Nvidia. Já que alia uma alta capacidade de RAM com um custo bem menor do que a sua concorrente, que se destaca em aplicações de visão e inteligência artificial.

Apesar deste microcomputador aceitar diversos sistemas operacionais, ele é mais comumente utilizado com o sistema operacional Raspian (28). Por conta disso é necessário seguir boas práticas quando se utiliza o utiliza com o Ubuntu para aplicações ROS. Dentre elas podem-se destacar:

- Utilização do Ubuntu server, ou seja, com uma quantidade reduzida de programas e sem desktop.
- Delegar ao mestre funções com custo computacional muito alto e que não precisa de uma rápida atualização, por exemplo o *global planner* em deslocamentos autônomos.
- Priorizar o c++ em relação ao python.
- Usar um sistema de resfriamento. Um dos maiores problemas do raspberry pi, principalmente em versões mais potentes como a de 8 gigabytes, é o seu excessivo aquecimento que pode comprometer seriamente a performance do computador.

Seguindo essas recomendações é muito provável que o Raspberry pi 4 utilizado seja capaz de ser utilizado em uma grande gama de operações sem comprometimento de desempenho.

6.1.3 Camera

A utilização de uma câmera é fundamental o sistema, permitindo ao operador do veículo controlá-lo mesmo sem contato visual com o veículo.

Dentro da comunidade do ROS já existem trabalhos que criam o sistema de comunicação entre o driver da câmera e o ROS. Podendo ser empregados desde câmeras mais complexas, como as da intel que possuem inclusive medição de profundidade (como foi trabalhado nas simulações de mapeamento em 3 dimensões), até simples webcams comerciais.

Inclusive, o próprio raspberry possui uma entrada específica para uma câmera desenvolvida pela mesma empresa. Ela possui inúmeras vantagens, tais como o baixo preço, a boa resolução e a grande possibilidade de alterações nos parâmetros da imagem, sendo possível configurar o driver em diferentes tipos de aplicação.

A câmera e sua montagem no microcomputador são mostrados, respectivamente, nas Figura 63 e Fig 64. Percebe-se que o componente fica extremamente exposto e pode ser danificado facilmente. Para contornar esse problema, modelou-se em CAD uma proteção para esse componente, mostrado na Figura 65, para ser construída via impressão 3D, obtendo-se o resultado mostrado nas Figura 66 e Figura 67



Figura 63 – Câmera específica para ser utilizada no raspberry

Apesar de suas vantagens, o driver deste modelo de câmera existe no sistema operacional nativo do Raspberry, o Raspian, entretanto está se utilizando o ubuntu 20.04. Portanto é necessário instalar o driver que reconheça essa entrada do computador, as instruções para tanto podem ser encontradas em (29).



Figura 64 – Montagem da câmera no microcomputador

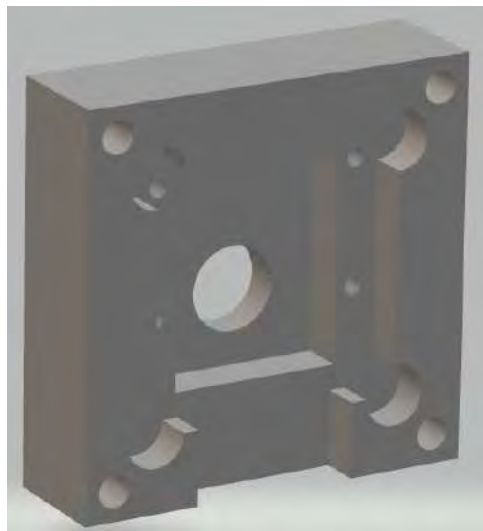


Figura 65 – Modelo CAD da proteção da câmera

Após a configuração completa do sistema é possível lançar o nó da câmera obtendo como resultado os tópicos da Figura 68. O tópico `/raspicamnode/parameterupdates` se refere ao conjunto de parâmetros que podem ser alterados dinamicamente no sistema e que estão representados na Figura 69. Apesar de haver instruções referentes à calibração da câmera, encontradas em (30), ela não pode ser realizado por motivos ainda desconhecidos, entretanto a resolução e a escala da imagem não estão comprometidos apesar disso, como pode ser visto no vídeo referente a Figura 70, que mostra a visualização dos dados da câmera no ROS Master.

6.1.4 Arduino

O arduino é uma placa de prototipagem eletrônica em código aberto. Ela é composta basicamente por um microcontrolador, uma interface de comunicação serial e pinos analógicos e digitais.

Implementar o arduino juntamente com o raspberry, ao invés de apenas utilizar



Figura 66 – Parte da frente da proteção montada na câmera



Figura 67 – Parte da frente da proteção montada na câmera

```
/diagnostics  
/raspicam_node/camera_info  
/raspicam_node/image/compressed  
/raspicam_node/parameter_descriptions  
/raspicam_node/parameter_updates  
/rosout  
/rosout_agg
```

Figura 68 – Tópicos referentes ao nó da câmera



Figura 69 – Parâmetros que podem ser configurados dinamicamente



Figura 70 – QR Code referente ao vídeo de teste da câmera

o microcomputador, é algo muito recomendável em aplicações que necessitem de rápida comunicação em tempo real com hardwares, já que o arduino trabalha a um nível mais baixo. Além disso, as portas GPIO do raspberry apresentam uma tensão máxima de 3.3 volts, enquanto o arduino pode apresentar até 5 volts. Isso é fundamental pois muitos sensores e drivers, como a própria controladora do motor que é usada no trabalho, funciona com entrada de tensão.

O arduino utilizado é do modelo MEGA cuja pinagem é mostrada na Figura 71. A sua tensão de entrada recomendada é de 6 a 12 volts, mas pode suportar até 12 volts sem prejuízos a placa.

O arduino pode ser programado por meio de uma IDE criada especificamente para ele, contendo uma linguagem própria e muito simples para facilitar o aprendizado, já que essa é uma ferramenta de entrada para outros microcontroladores. Contudo, para aplicações mais complexas, como trabalhar conjuntamente com o ROS, é recomendável que se utilize C++ para o desenvolvimento do programa.

A troca de informações entre o arduino e o raspberry é realizada por meio de uma comunicação serial, um processo no qual os dados são enviados bit a bit sequencialmente por meio de um canal de comunicação ou barramento. Para aplicar essa comunicação ao sistema é necessário utilizar a biblioteca *rosserial-arduino* que possui os pacotes ROS

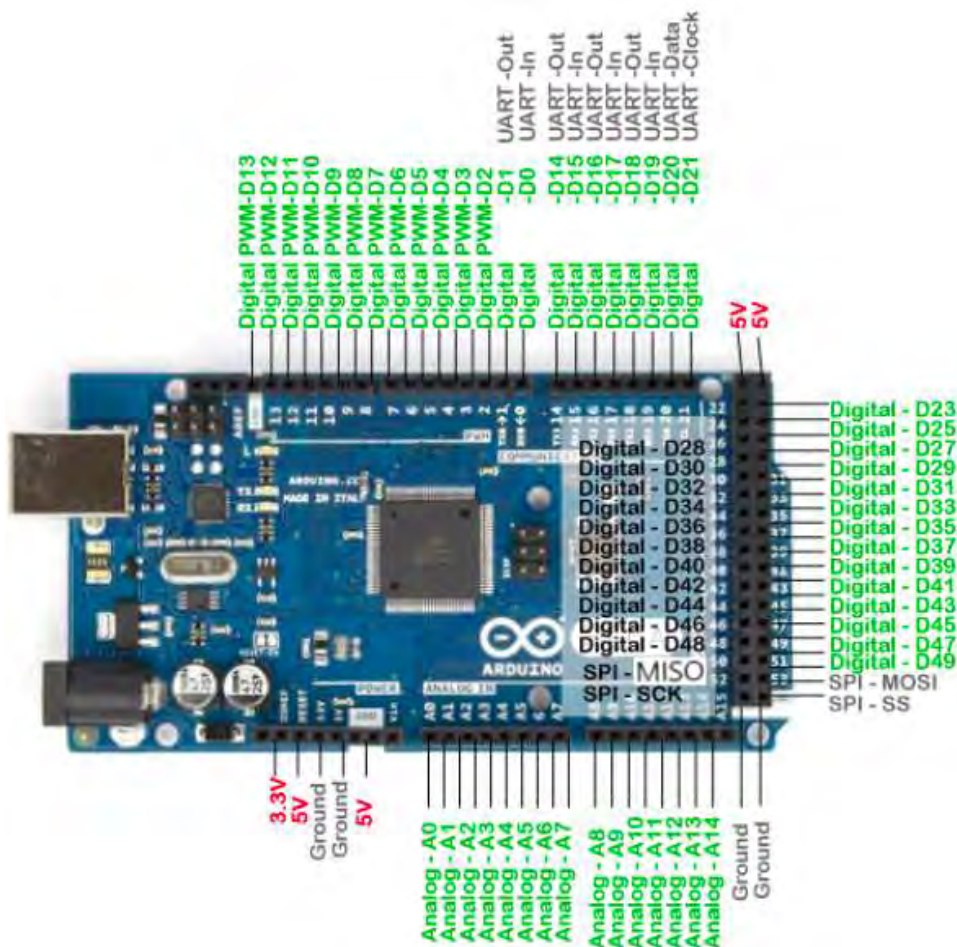


Figura 71 – Pinagem do arduino Mega

necessários para a ação, bem como a biblioteca ROS.h instalada no compilador do arduino.

A montagem do sistema é mostrado na Figura 72, na qual o cabo que liga os dois sistemas é o canal de comunicação serial.

6.1.5 Sensor MPU6050

O sensor MPU6050 é um componente que agrega três sensores, um de temperatura, um giroscópio e um acelerômetro cujos eixos de medida dos sensores são indicados nas Figura 73 e Figura 74.

Os acelerômetros, geralmente, contêm placas capacitivas internamente. Algumas são fixas, enquanto outras estão ligadas a molas minúsculas que se movem internamente à medida que as forças de aceleração atuam sobre o sensor. Quando estas placas se movem em relação uma a outra, a capacitância entre elas muda. A partir dessas mudanças na capacitância, a aceleração pode ser determinada.

Enquanto isso, quando o giroscópio é rotacionado, uma pequena massa de ressonância é deslocada à medida que a velocidade angular muda. Este movimento é convertido

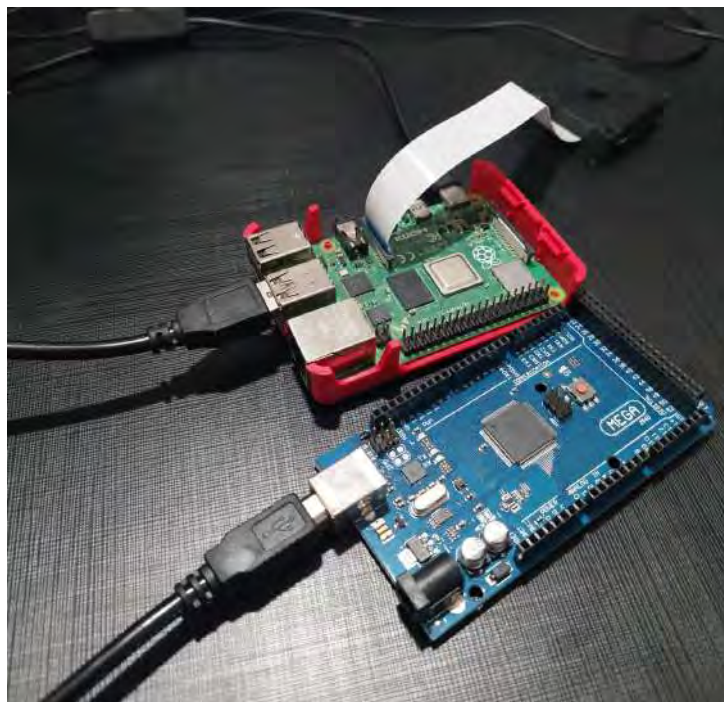


Figura 72 – Conexão entre o arduino e o raspberry pi

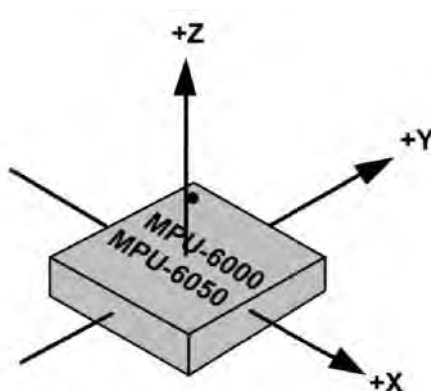


Figura 73 – Eixos de medida do acelerômetro em relação ao sensor

em sinais elétricos de corrente muito baixa que podem ser amplificados e lidos por um microcontrolador hospedeiro.

Todas as informações relevantes sobre o sensor pode ser encontrada em seu *datasheet* (31). Esse é um documento imprescindível para a utilização do sensor, pois especifica o que cada pino significa, parâmetros de segurança, bem como como recuperar a leitura do sensor. Por exemplo, é por meio deste documento que se descobre que a informação do sensor, como cada componente da aceleração, da velocidade angular e a temperatura são armazenados em dois registradores de 8 bits cada e, portanto, deve-se concatenar as informações de forma a obter um registrador único de 16 bits.

Na Figura 75 está representado o sensor bem como a sua pinagem. Destes pinos

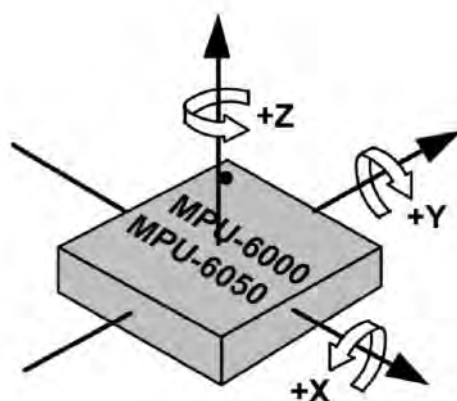


Figura 74 – Eixos de medição do giroscópio em relação ao sensor

são utilizados apenas 4, a entrada de tensão (VCC), o terra (GND) e os dois pinos de comunicação I2C (SCL e SDA). O protocolo de comunicação I2C utiliza dois pinos, o SDA, referente a comunicação serial de dados, e o SCL, referente a comunicação serial de *clock*, ou seja, um pino é para a transferência propriamente dita de dados, a leitura do sensor, enquanto o outro é a sincronização do sistema.

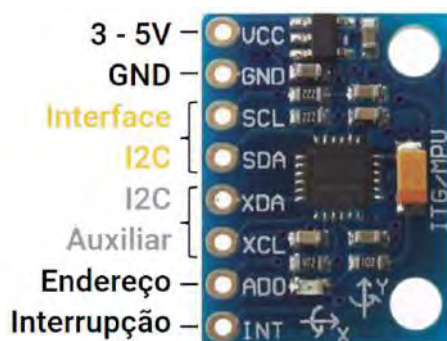


Figura 75 – Sensor MPU-605

A montagem do sistema é mostrada na Figura 76, onde são utilizados pelo arduino MEGA os pinos 20 e 21, respectivamente, os pinos SDA e SCL da placa, como é possível notar na Figura 76.

Após a montagem ainda é necessário calibrar o sistema, já que ele vem de fábrica com uma calibração inicial mas por pequenos defeitos de fabricação e variações de temperatura ela pode não ser mais válida. Percebe-se que apenas a aceleração na direção do eixo x do sensor precisa ser calibrada. O estado inicial bem como o resultado da calibração estão indicados, respectivamente, na Figura 77 e Figura 78.

Para calibrar esse sistema foram adicionadas pequenas parcelas as leituras de forma a obter os valores nulos em uma situação sem acelerações, com exceção da gravitacional, e sem velocidades angulares, além de garantir que o componente esteja paralelo ao solo.

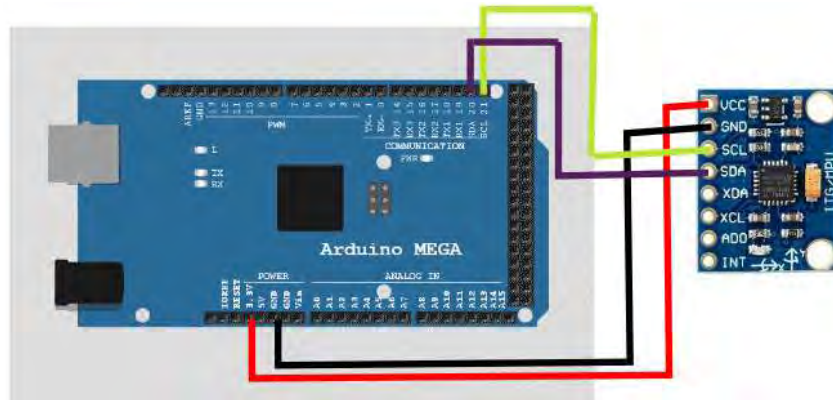


Figura 76 – Conexão entre o arduino mega e o MPU6050

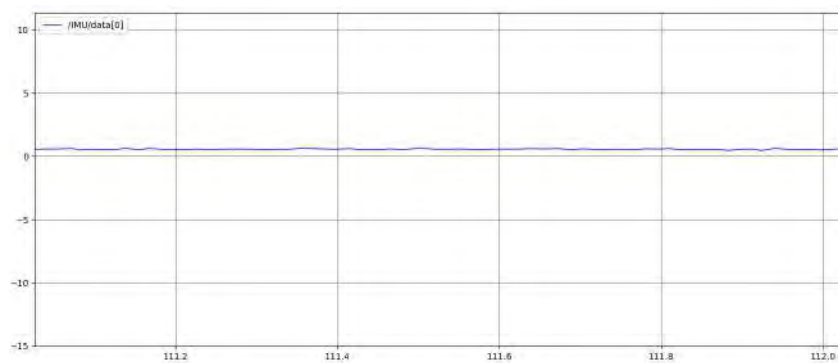


Figura 77 – Leitura da aceleração linear no eixo x do sensor antes da calibração

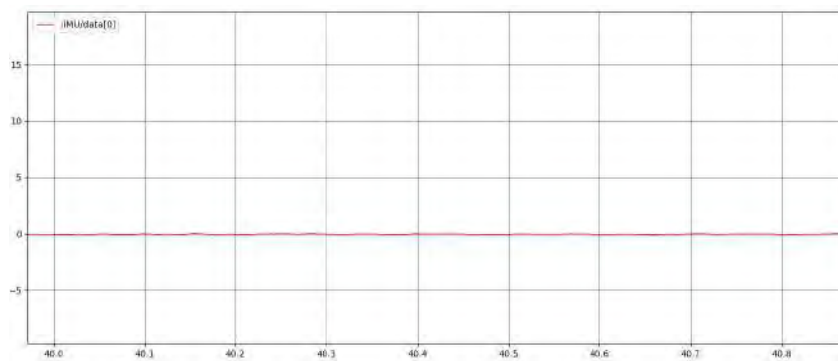


Figura 78 – Leitura da aceleração linear no eixo x do sensor após da calibração

6.1.6 Placa controladora do motor

O arduino possui uma limitação na tensão de saída que pode ser fornecida, chegando a um máximo de 5 volts. Entretanto, o motor do rover opera de 0 a 12 volts e portanto não seria possível utilizar o motor em toda a sua capacidade com apenas o arduino. Para aumentar a tensão fornecida utiliza-se uma placa controladora sabertooth 2 x 12, mostrada na Figura 79.

Além de aumentar a tensão fornecida ao motor, essa placa também disponibiliza



Figura 79 – Sabertooth 2x12 (9)

uma série de configurações que podem alterar o funcionamento do motor, uma descrição mais precisa delas pode ser encontrada em (32). Neste modo um dos canais de comunicação controla o movimento longitudinal do veículo enquanto o outro canal controla a rotação do veículo. O sinal de entrada é de 0 a 5 volts, sinais abaixo de 2.5 volts corresponde ao movimento para frente enquanto acima de 2.5 volts move na outra direção.

O driver do motor recebe uma entrada analógica, entretanto, as saídas digitais do arduino são mais precisas. Portanto, são utilizadas as saídas digitais pwm do arduino, uma estratégia que simula sinais analógicos por meio de sinais digitais a partir do tempo de atuação do sinal digital, como mostrado na Figura 80.

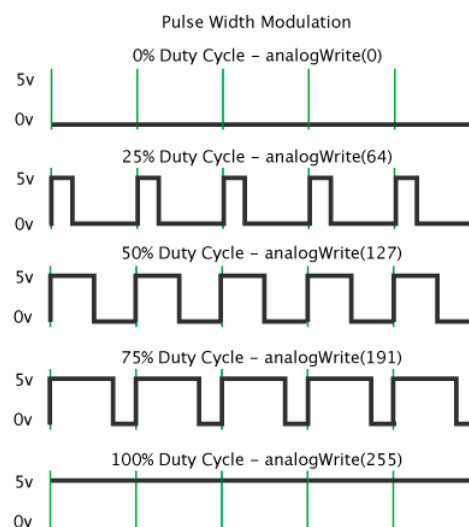


Figura 80 – Saídas PWM (9)

7 SISTEMA DE CONTROLE

Até o presente momento o veículo real funciona em malha aberta, ou seja, um sinal de velocidade por parte do operador é enviado ao raspberry, que transmite ao arduino que por sua vez converte isso em um sinal de tensão para a placa controladora do veículo. Esse sistema funciona de forma efetiva para aplicações em que o veículo é operado a distância, já que o operador pode modular o sinal com base na sua visão e na câmera instalada no veículo.

Entretanto, para aplicações que demandem autonomia do veículo em suas, como mostrado nas simulações das seções 5.5.1 e 5.5.2, é necessário que haja um sistema em malha fechada, que responda a um sinal referência de velocidade, tanto longitudinal quanto angular, sendo portanto um controle de baixo nível. Esse controle é responsável por fornecer o sinal de tensão à placa controladora do motor.

Existem uma grande gama de métodos de controle que podem ser utilizados para esse sistema, tais como um controle PID, ótimo, por lógica *Fuzzy*, entre outros. Entretanto, um dos requisitos de um veículo militar é apresentar um bom desempenho mesmo quando submetido a diferentes situações, logo ele deve ser robusto o bastante para lidar com variações do sistema. Devido a essa necessidade é escolhido um controle robusto para o veículo, capaz de resistir tanto a variações do coeficiente de atrito na interação pneu solo, que representa diferentes terrenos, e também diferentes valores de massa que o veículo pode apresentar, representando diferentes carregamentos.

7.1 Dinâmica Veicular

Para a síntese de um controle robusto é necessário um modelo linearizado do sistema, portanto, é necessário linearizar a dinâmica longitudinal do veículo. Na construção de qualquer modelo longitudinal é fundamental que se faça a consideração do atrito mais adequada para o sistema. Dentre as diversas modelagens possíveis, é utilizado o atrito de Coulomb. Neste caso a força de atrito é proporcional à força de contato normal (F_N) e apresenta uma direção contrária a velocidade relativa entre os corpos (\dot{x}). Essa relação é exposta na Eq. 7.1, na qual μ é o coeficiente de atrito entre as superfícies, no caso do veículo entre o pneu e o solo.

$$F_{atr} = \mu F_N \text{sign}(\dot{x}) \quad (7.1)$$

Essa equação apresenta um problema, pois possui uma descontinuidade na origem, como fica claro na Figura 81. Para contornar esse problema é possível linearizar essa

expressão em torno da origem com base na velocidade de deslizamento do veículo, com base no trabalho de (33), obtendo-se como resultado a Eq. 7.2, que tem como resultado o gráfico da Figura 82.

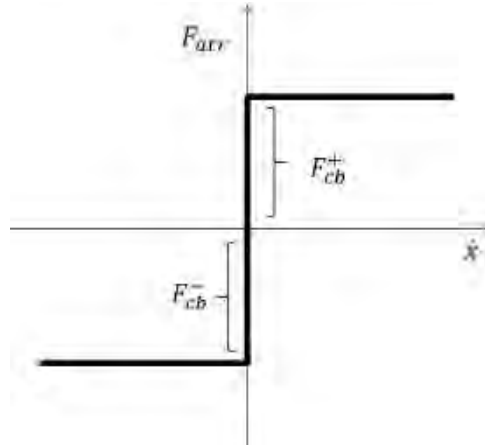


Figura 81 – Modelo de atrito de coulomb

$$F_{atr} = \mu F_N \frac{V_{desl}}{V_{lim}} \tag{7.2}$$

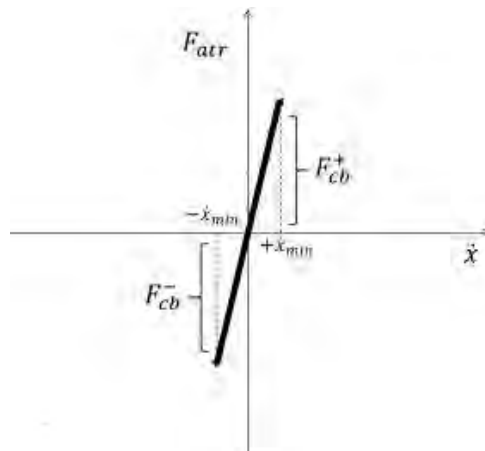


Figura 82 – Modelo de atrito de coulomb linearizado

O veículo é considerado como um chassi rígido que contém quatro rodas e portanto é possível se considerar apenas o movimento plano do veículo, tanto de translação quanto de rotação. Na Figura 83 está representado o veículo, bem como o referencial global, X_1Y_1 , e o referencial local do veículo, X_2Y_2 . É importante relembrar que o veículo é diferencial e portanto apresenta como graus de liberdade apenas de rotação e deslocamento longitudinal no seu referencial local.

O modelo de atrito de Coulomb linearizado necessita da velocidade de deslizamento entre a roda e o solo, pode-se definir a velocidade de deslizamento a partir da Eq. 7.3. No qual V_i representa a velocidade do veículo em relação ao seu próprio referencial.

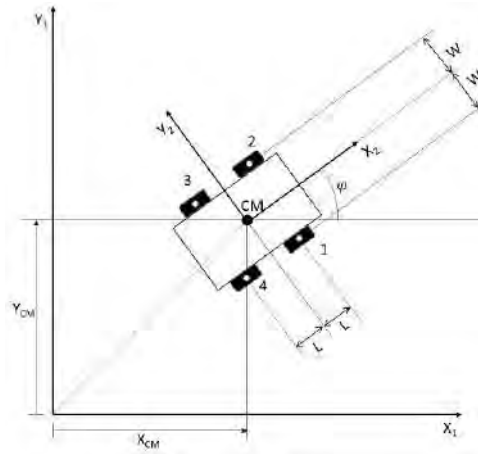


Figura 83 – Representação d veículo no plano

$${}^2V_{Di} = -{}^2V_i + \begin{bmatrix} x' \\ y' \end{bmatrix} \quad (7.3)$$

Já que este se trata de um sistema plano, pode-se considerar que a carga sobre cada uma das rodas é a mesma. Aplicando essa hipótese é possível encontrar o valor da força desenvolvida por cada uma das rodas do veículo, como mostrado na Eq. 7.4. A atuação dessas forças agindo no veículo está representado pela Figura 84

$${}^2F_{atr,i} = \mu \frac{mg}{4} \frac{{}^2V_{Di}}{V_{lim}} \quad (7.4)$$

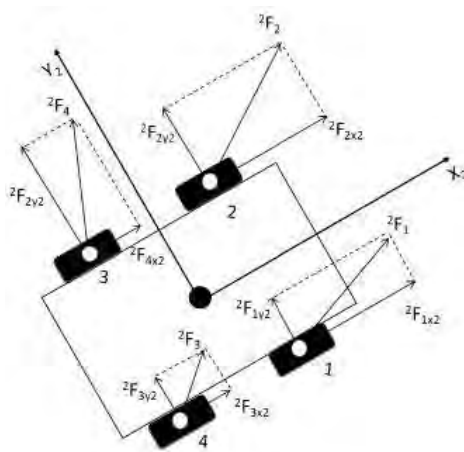


Figura 84 – Forças dos pneus durante o movimento do veículo

Aplicando a segunda lei de Newton para referencias móveis e a lei de Euler ao veículo chega-se nas Eq. 7.5 e Eq. 7.6, que representam o movimento do veículo no plano, sendo r_i a posição da roda i em relação à origem do referencial do chassi.

$$m \frac{d}{dt} v = m(\dot{v} + \ddot{\psi}) = \sum_1^4 F_i \quad (7.5)$$

$$\ddot{\psi} = \sum_1^4 \tilde{r}_i^2 F_i \quad (7.6)$$

A dinâmica das rodas do veículo pode ser descrita por meio da Eq. 7.7, na qual T_i é o torque no eixo da roda, c é o coeficiente de atrito viscoso, J é o momento de inércia polar da roda, r é o raio da roda do veículo e $\dot{\theta}$ sendo a velocidade angular da roda.

$$J\ddot{\theta}_i = T_i - c\dot{\theta} - F_{ix}r \quad (7.7)$$

O torque fornecido pelo motor elétrico depende da tensão de entrada, u que ele recebe de acordo com as equações Eq. 7.8 e Eq. 7.10. Sendo K_{bm} a constante de força contra eletromotriz, R a resistência elétrica do motor, L sendo a indutância da armadura, i a corrente elétrica do motor, K_t a constante de torque do motor, C_m o coeficiente de atrito viscoso e red a redução que existe entre o motor elétrico e o eixo da roda.

$$\frac{d}{dt} i_i = \frac{u_i - K_{bm}\dot{\theta}_i red - Ri_i}{L} \quad (7.8)$$

$$T_{motor} = K_t i_i - C_m \dot{\theta}_i red \quad (7.9)$$

Como existe uma redução entre o motor e a roda, o torque no eixo do carro será maior do que o do motor, de forma que:

$$T_{roda} = T_{motor} red \quad (7.10)$$

Na Tabela 1 estão indicados os valores dos parâmetros do veículo utilizado.

7.2 Síntese do Controlador

7.2.1 Representação na Forma Padrão

O sistema do veículo é composto por diversos subsistemas, cada um com as suas respectivas entradas, saídas e perturbações. Assim, para facilitar o desenvolvimento de controladores robustos, foi definida uma forma padrão. Nessa arquitetura as entradas passam a ser somente as entradas externas do sistema, denominadas exógenas, e as saídas passam a ser os sinais que se deseja controlar e medir. Na Figura 85 está explicitado a relação entre a forma convencional de representação do sistema por meio de diagrama

Tabela 1 – Dados do veículo utilizado.

Comprimento do Chassi	18.7 <i>cm</i>
Largura do Chassi	20.3 <i>cm</i>
Diâmetro da Roda	12 <i>cm</i>
Momento de Inércia do veículo	0..0338 <i>kgm²</i>
Redução do Motor	30
Massa do Veículo	3.044 <i>kg</i>
Coefficiente de Atrito Nominal	0.7
Momento de Inércia Polar da Roda	0.0000331 <i>kgm²</i>
Constante de Torque	005 <i>Nm/A</i>
Indutância do Motor	0.001 <i>H N/m</i>
Resistência Elétrica do Motor	40 Ω
Constante de Força Contra Eletromotriz	<i>V rad/s</i>
Coefficiente de Atrito Viscoso do Motor	0
Coefficiente de Atrito Viscoso da Roda	0.015

de blocos e a forma padrão para sistemas de controle robustos, na qual K representa o controlador e G representa a dinâmica do veículo.

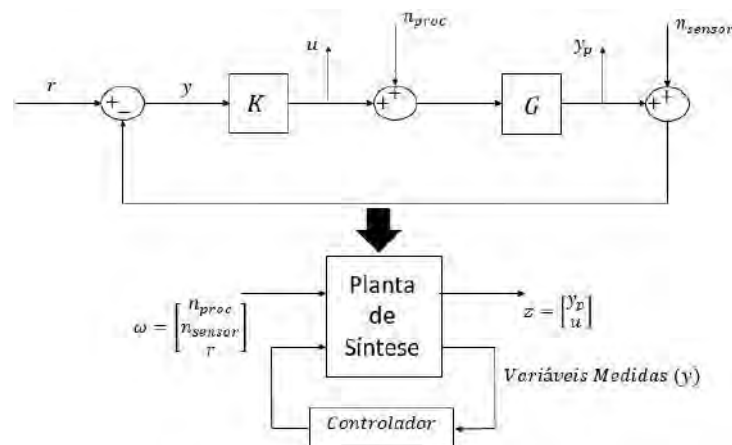


Figura 85 – Representação de sistema em diagrama de blocos e na forma padrão para a síntese do controlador

Portanto as entradas e saídas do sistema são relacionadas pela Eq. 7.11, onde $P(s)$ é fornecida pela Eq. 7.12.

$$\begin{bmatrix} y_p \\ u \\ y \end{bmatrix} = P(s) \begin{bmatrix} n_{proc} \\ n_{sensor} \\ r \\ u \end{bmatrix} \quad (7.11)$$

$$P(s) = \begin{bmatrix} G & 0 & 0 & G \\ 0 & 0 & 0 & I \\ -G & -I & I & -G \end{bmatrix} \quad (7.12)$$

Portanto a função de transferência em malha fechada do sistema é dado pela Eq. 7.13.

$$H = \begin{bmatrix} G & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} G \\ I \end{bmatrix} K(I + GK)^{-1} \begin{bmatrix} -G & -I & I \end{bmatrix} \quad (7.13)$$

Desenvolvendo a Eq. 7.13 é obtido a Eq. 7.14 que descreve a função de transferência em malha fechada do sistema da Figura 85.

$$H = \begin{bmatrix} G(I + GK)^{-1} & -GK(I + GK)^{-1} & GK(I + GK)^{-1} \\ K(I + GK)^{-1}G & -K(I + GK)^{-1} & K(I + GK)^{-1} \end{bmatrix} \quad (7.14)$$

Na análise de sistemas multivariáveis, para se mensurar o ganho do sistema, algo extremamente importante ao se analisar a resposta no domínio da frequência, deve-se utilizar o conceito de valores singulares da matriz de transferência do sistema. Assim, o maior ganho possível para sistemas multivariáveis está associado à norma H_∞ , dada pela definição a seguir (34).

Definição 7.1 Norma RMS (H_∞) a partir da decomposição de valores singulares
Seja $A \in F^{m \times n}$. Então as matrizes unitárias:

$$U = [u_1 \dots u_m] \in F^{m \times m} \quad (7.15)$$

$$V = [v_1 \dots v_n] \in F^{n \times n} \quad (7.16)$$

tal que:

$$A = U \Sigma V^T \quad (7.17)$$

onde

$$\Sigma = \begin{bmatrix} \Sigma_p & 0 \\ 0 & 0 \end{bmatrix} \quad (7.18)$$

com

$$\Sigma_p = \text{diag}(\sigma_1 \ \sigma_2 \ \dots \ \sigma_p) \quad (7.19)$$

e $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ são valores singulares de A e $p = \min(m, n)$. E portanto a norma H_∞ de A é definida por:

$$\|H\|_\infty \triangleq \sup \sigma_{\max}(A(j\omega)) \quad (7.20)$$

7.2.2 Representação das Incertezas

A teoria de controle robusto tem como objetivo o desenvolvimento de controladores que sejam menos sensíveis a divergências entre o modelo utilizado para a síntese e o sistema físico. As incertezas podem ser de diversas naturezas tais como possíveis variações e flutuações nos parâmetros físicos, não linearidades, dinâmicas negligenciadas, condições externas e entre outros. No presente trabalho, o objetivo é de sintetizar um controlador que suporte variações de massa de até 50% em relação ao seu valor nominal, representando diferentes carregamentos, e de até 40% no valor nominal de coeficiente de atrito, para representar diferentes tipos de terrenos.

Portanto, o objetivo do controlador é ser capaz de apresentar um desempenho aceitável para todas as plantas no conjunto das incertezas definidas. Para tanto, é importante definir os problemas de análise de estabilidade robusta. Um sistema tem estabilidade robusta quando, para um dado controlador K , o sistema em malha fechada permanece estável para todas as plantas no conjunto das incertezas.

A estratégia de síntese é baseada em se isolar as incertezas do sistema, definidas como uma matriz diagonal Δ , em uma função de transferência superior, como exemplificado na Figura 86.

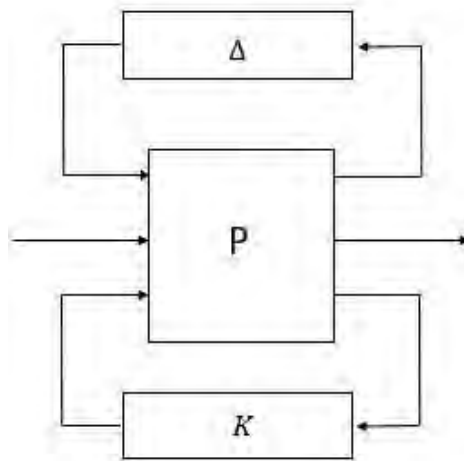


Figura 86 – Planta representada na forma padrão de controle com a representação das incertezas sendo uma LFT superior

A partir dessa estrutura de análise do sistema, pode-se aplicar o teorema dos pequenos ganhos desenvolvido em (35) para se analisar a estabilidade robusta do sistema. Tal teorema e, conseqüentemente, sua formulação matemática não são desenvolvidos, pois fogem demasiadamente do escopo do presente trabalho. Entretanto, com base nessa referência, chega-se que a estabilidade robusta do sistema pode ser garantida se a norma H_∞ das saídas na forma padrão de controle, tal qual exposta na Figura 86, sejam menores do que 1.

Exemplificando essa formulação para o sistema do sistema em questão se tem o diagrama de blocos da Figura 87. Aos sinais de entrada e de saída do controlador são aplicadas ponderações para modular esses sinais em determinadas frequências desejadas. Esse mesmo modelo pode ser reescrito na forma padrão de controle, como mostrado na Figura 88.

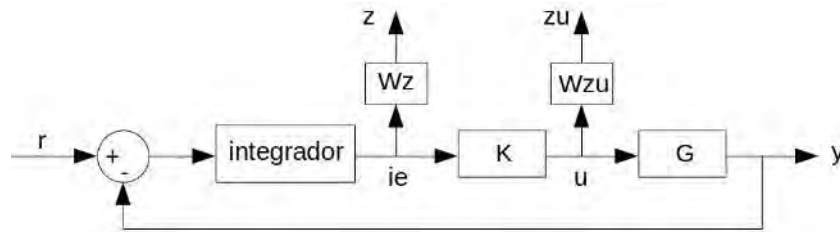


Figura 87 – Sistema em malha fechada para a síntese do controle

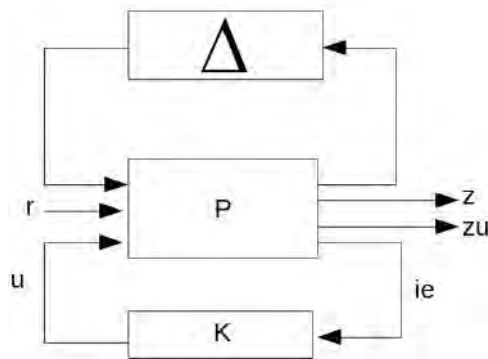


Figura 88 – Forma padrão de controle para o sistema do veículo

As funções de ponderação utilizadas estão indicadas nas Eq. 7.21 e Eq. 7.22. É importante ressaltar que ambas as funções de transferência agem em dois sinais de erro, velocidade linear e angular, e de sinais de controle, as duas correntes do sistema.

$$W_z = \frac{0.5 (s + 1)}{s + 10} \tag{7.21}$$

$$W_{zu} = \frac{0.6 s}{s + 100} \tag{7.22}$$

Um outro fator importante para se considerar na síntese de um controle robusto é a ordem do controlador, representada pela ordem da matriz A da representação do controlador no espaço de estados. Quanto menor a ordem menor o custo computacional para implementar esse controle em um sistema digital. Como o objetivo final de implementação é um arduino, um sistema de baixo poder de processamento, é de suma importância buscar um controlador com uma ordem baixa.

A síntese do controle por meio de um processo de otimização utilizando a biblioteca *systeme* do MATLAB, obtém-se o seguinte controlador no espaço de estados:

$$K.A = \begin{bmatrix} 55.65 & 5748 & 0 \\ -193.2 & -4307 & 481.8 \\ 0 & -2.462 \cdot 10^4 & -1153 \end{bmatrix} \quad (7.23)$$

$$K.B = \begin{bmatrix} -2062 & 2843 \\ 1538 & -2051 \\ 8859 & -1.235 \cdot 10^4 \end{bmatrix} \quad (7.24)$$

$$K.C = \begin{bmatrix} 13.91 & 249.1 & -1243 \\ -74.71 & 234.4 & -944.7 \end{bmatrix} \quad (7.25)$$

$$K.D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (7.26)$$

O critério de estabilidade robusta é baseado na norma H_∞ das saídas serem menores do que 1. Na Figura 89 está indicado que essas saídas apresentam normas menores do que a unidade para todo o domínio da frequência.

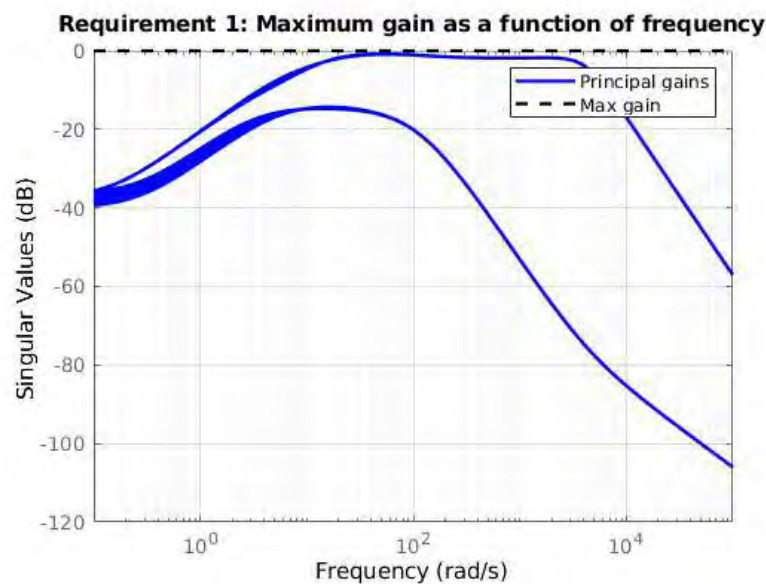


Figura 89 – Ganho das saídas ponderadas do sistema em todo o domínio da frequência

As respostas do sistema a uma entrada em degrau de velocidade linear e angular estão representadas nas Figura 90 e Figura 91, respectivamente. É possível notar que para o controlador em questão as incertezas geram maiores interferências na saída de velocidade longitudinal do veículo.

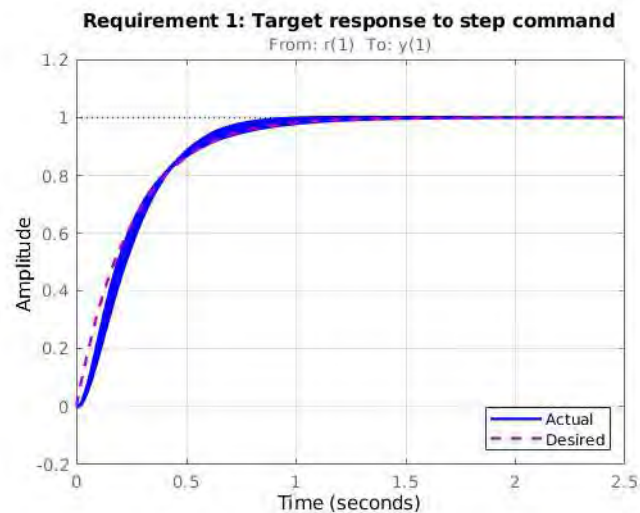


Figura 90 – Velocidade longitudinal do sistema em malha fechada quando submetido a um degrau de velocidade linear

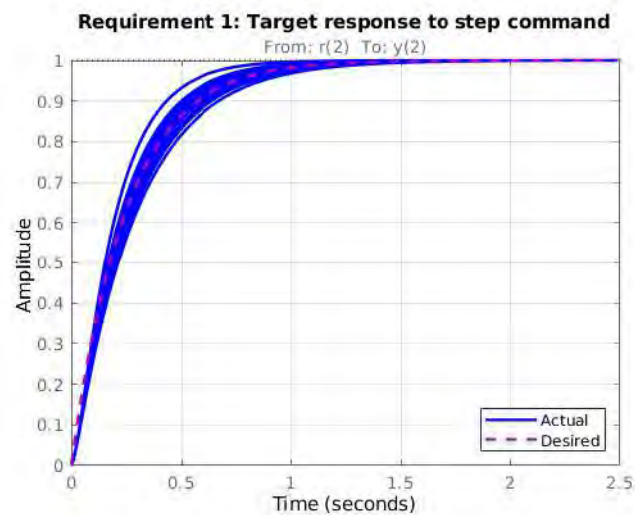


Figura 91 – Velocidade angular do sistema em malha fechada quando submetido a um degrau de velocidade angular

Além disso, outro requisito de desempenho é a baixa existência de interferência entre as entradas do modelo. Isso é importante para que um comando não cause excessivas perturbações em outra saída e invalide a aplicabilidade do controlador, por exemplo uma entrada de velocidade longitudinal que gere uma resposta de velocidade angular elevada por um período grande de tempo. Na Figura 92 esta indicada a resposta da velocidade longitudinal para uma entrada em degrau de velocidade angular e Figura 93 a situação inversa.

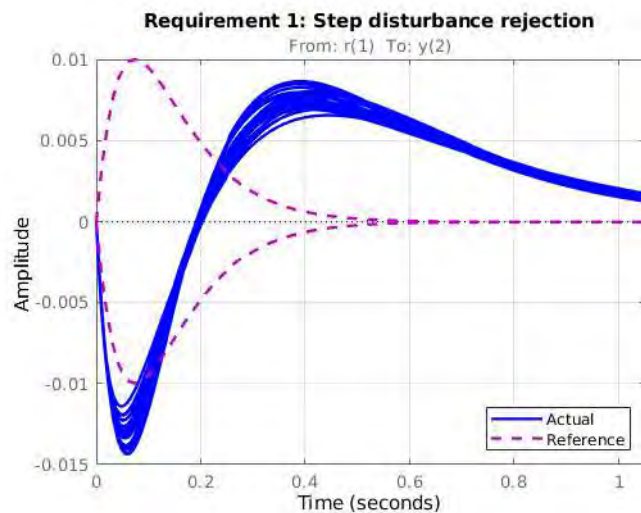


Figura 92 – Velocidade longitudinal do sistema em malha fechada quando submetido a um degrau de velocidade angular

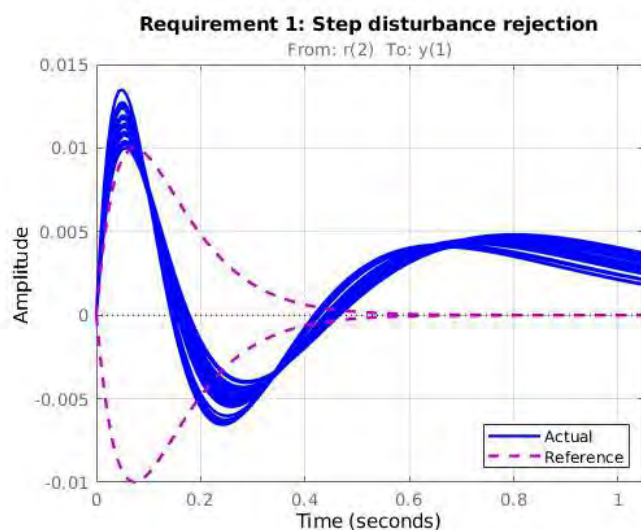


Figura 93 – Velocidade angular do sistema em malha fechada quando submetido a um degrau de velocidade longitudinal

7.3 Discretização do Controle

O objetivo deste controle é de ser aplicado no microcontrolador do veículo, um arduino AtMega, um componente digital. Portanto, para a integração há dois caminhos possíveis: utilizar o controlador no espaço contínuo utilizando o hardware a uma alta frequência, de forma a aproximar o sistema discreto do contínuo ou discretizar o controle. Apesar de acrescentar uma complexidade à síntese do controlador a solução ideal é discretizar o sistema para otimizar o desempenho do Hardware.

Existem diferentes maneiras de discretizar um controle. Entretanto, como nesse processo há perda de informação do sistema em relação ao contínuo é importante escolher o

método coerente. Para tanto é escolhido o método do *zero order hold* que tem seu domínio discreto z relacionado com o domínio s por meio da relação explicitada na Eq. 7.27. A sua resposta se caracteriza por manter o valor do sinal recebido entre dois instantes de tempo, como mostrado na Figura 94, esse é o comportamento que se espera do motor dc do veículo.

$$z = e^{sT} \quad (7.27)$$

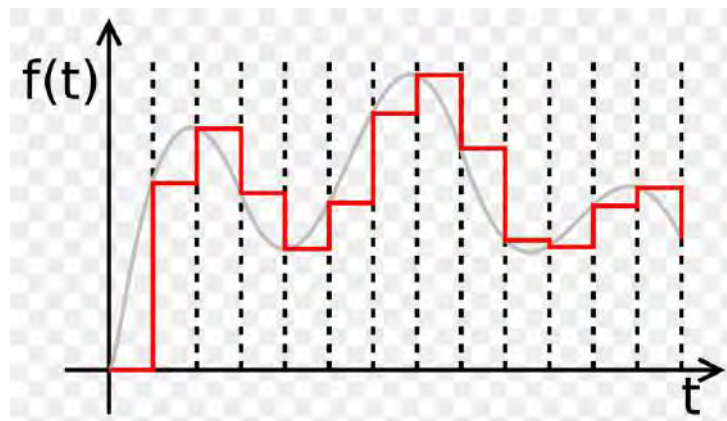


Figura 94 – Comportamento de um sistema discretizado por meio do método *zero order hold*

Além do método, é de extrema importância definir o período de amostragem correto para o sistema, pois tempos muito curtos podem exigir demais do *Hardware* e não serem condizentes com o tempo de medição dos sensores, contudo um tempo muito alto pode desestabilizar o modelo e causar grandes atrasos nas respostas. Para o sistema do veículo, considerando o arduino como *Hardware* esse passo é definido como 50 milissegundos. Esse passo de tempo é alto para o sistema e acarretará em perdas de informações como é discutido nas seções de resultados, entretanto permite que o Arduino, um *Hardware* de baixo custo e desempenho, funcione sem comprometimento a sua memória dinâmica. Aplicando-se esses conceitos o controlador no domínio discreto é definido por:

$$K.A = \begin{bmatrix} 0.3598 & 0.1402 & 0.0596 \\ -0.0047 & -0.0018 & -0.0008 \\ 0.1024 & 0.0399 & 0.0170 \end{bmatrix} \quad (7.28)$$

$$K.B = \begin{bmatrix} 0.0003 & 0.1796 \\ 0.3590 & -0.4964 \\ 0.0172 & -0.1094 \end{bmatrix} \quad (7.29)$$

$$K.C = 10^3 \begin{bmatrix} 0.0139 & 0.2491 & -1.2429 \\ -0.0747 & 0.2344 & 0.9447 \end{bmatrix} \quad (7.30)$$

$$K.D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (7.31)$$

Para comparação são realizados três testes: um degrau de velocidade linear de 0.3 m/s, um degrau de velocidade angular de 30 graus/s e uma combinação de ambos, utilizando valores nominais de massa e coeficiente de atrito, e após isso testes com valores críticos de massa e de atrito para testar a robustez do controlador digital.

7.3.1 Degrau de Velocidade Linear de 0.3 m/s

Na Figura 95 está presente a comparação entre as saídas das malha com controle contínuo e com o controle discreto, percebe-se que apesar do sistema apresentar um leve atraso para começar o seu movimento ele é capaz de atingir a velocidade mais rapidamente. Na Figura 96 é mostrado que até mesmo a interferência entre as saídas diminuiu com o controle discreto.

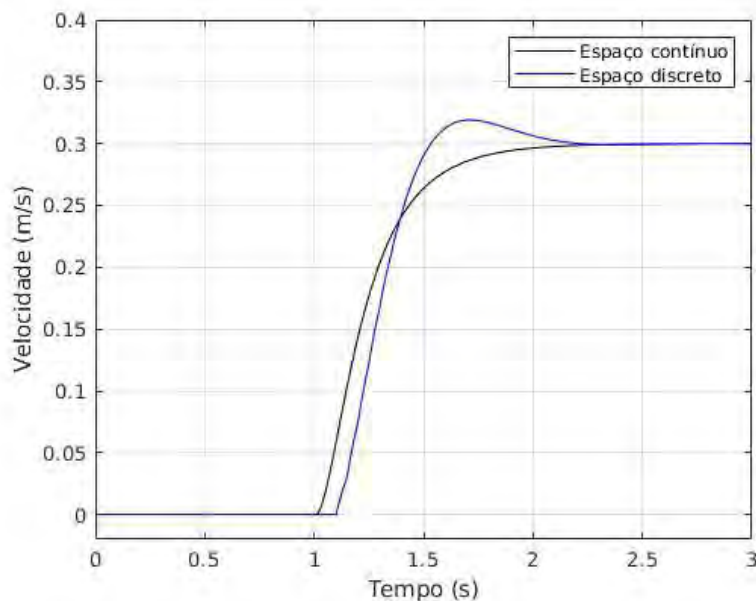


Figura 95 – Comparação entre as respostas de velocidade linear das malhas contínua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s

Ambos os resultados podem ser explicados por meio do sinal de controle de saída do controlador, representado na Figura 97. Percebe-se que o controle contínuo apresenta um desvio da sua resposta esperada nos primeiros momentos em que o sinal de referência é aplicado, isso é algo que não necessariamente pode ser explicado, pois é um comportamento matemático do controlador que foi sintetizado por uma método de otimização para cumprir determinados parâmetros de desempenho. No entanto, ao discretizar o sistema um atraso é adicionado, que possui o mesmo valor do passo de tempo do controlador, devido a

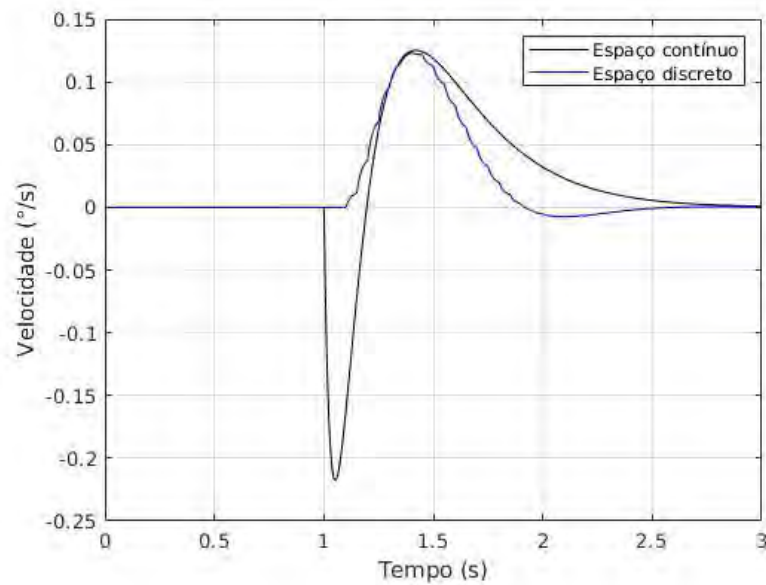


Figura 96 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s

esse atraso o controle discreto não experimenta essa situação específica e por conta disso apresenta uma resposta melhor.

Além disso, fica evidente nas três figuras que a perda de informação, inerente ao processo de discretização, causa *overshoots* nas saídas e nos sinais de controle, entretanto como esses valores estão dentro de magnitudes aceitáveis isso não é necessariamente um problema. Isso pode ser resolvido diminuindo o passo da discretização, entretanto isso acarretará em maior custo computacional para o microcontrolador.

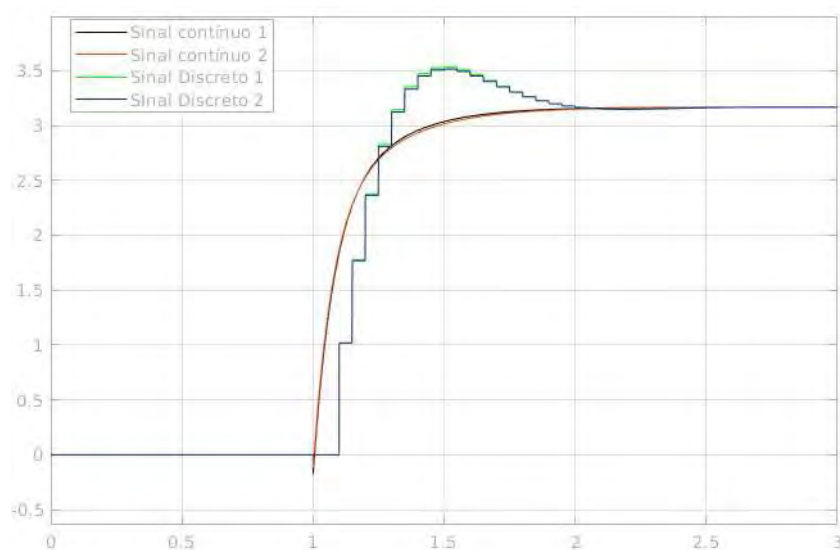


Figura 97 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidade de 0.3 m/s

7.3.2 Degrau de Velocidade Angular de $30^\circ/\text{s}$

Realizando os mesmos testes, mas para uma entrada de velocidade são obtidas a Figura 98 e Figura 99 que representam, respectivamente, as respostas da velocidade angular e linear do veículo. Percebem-se os mesmos efeitos relacionados ao atraso do sistema e de um maior *overshoot* na resposta de velocidade angular.

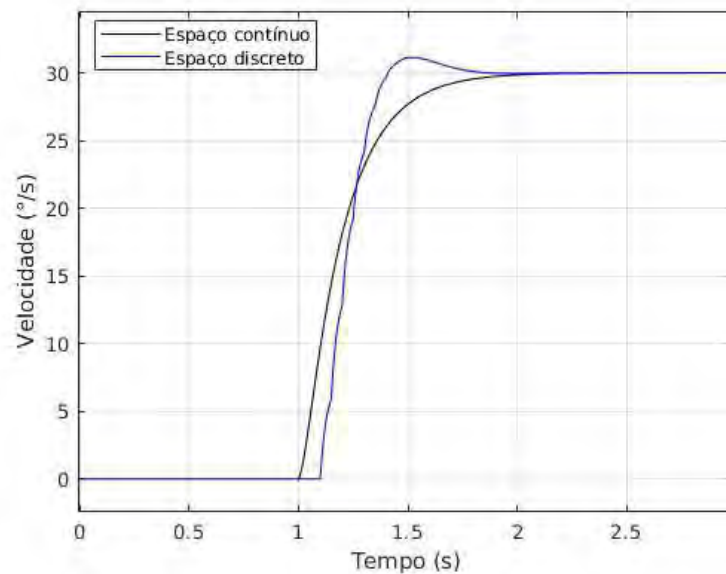


Figura 98 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidade de 0.3 m/s

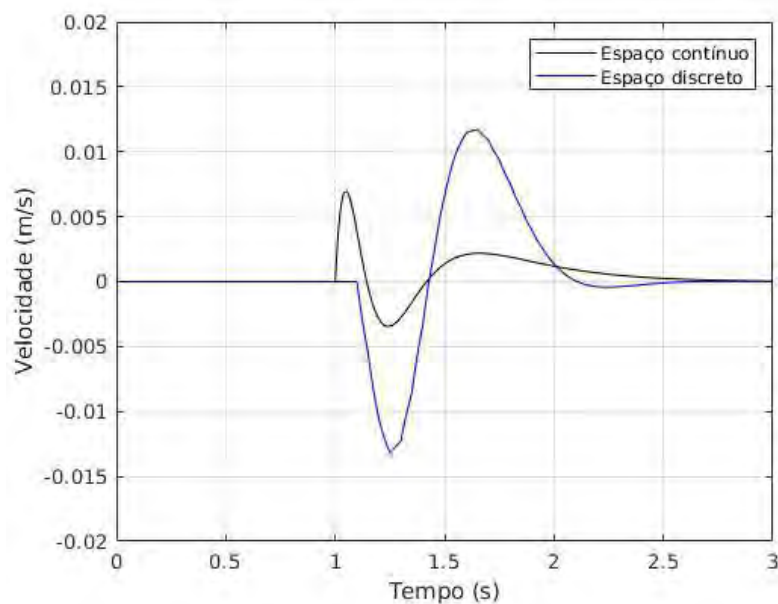


Figura 99 – Comparação entre as respostas de velocidade angular das malhas contínuas e discretas devido a uma entrada em degrau de velocidade de 0.3 m/s

Os sinais de controle contínuos e discretos estão expostos na Fig 100. Mais uma vez é possível observar que o atraso do sistema auxilia o sistema digital no que tange ao pequeno desvio que ocorre nos primeiros instantes do controlador contínuo. Entretanto, esse mesmo efeito causa um maior *overshoot* no sistema.

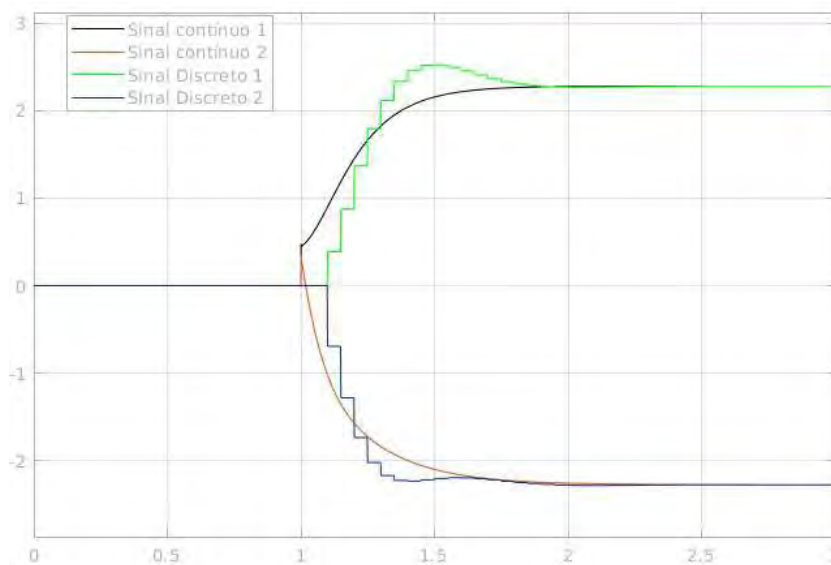


Figura 100 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidade de 30 °/s

7.3.3 Degrau de Velocidade Linear e Angular Combinados

À simulação é aplicado um degrau de velocidade linear de 0.2 m/s e angular de 20 °/s combinados ambos atuando a partir do mesmo instante de tempo. Os valores foram menores do que as simulações realizadas anteriormente para não saturar a capacidade do motor.

Na Figura 101 e na Figura 102 são mostradas as saídas de velocidade linear e angular da planta. Com ambas mostrando um comportamento muito semelhante ao já mostrado nas seções 7.3.1 e 7.3.2, algo já esperado pois a interferência entre as saídas é pequena.

De forma análoga a Fig 103 apresenta uma comparação entre os sinais de controle do modelo contínuo e discreto, estando mais uma vez presente os efeitos de perda de informação no sistema, o que causa os *overshoots* do sistema discreto.

Para testar a robustez do controle discreto é possível realizar testes com diferentes valores de massa e coeficiente de atrito, dentro dos limites estipulados na síntese do controlador, de forma a validar a sua robustez. Para tanto são realizados os testes em cinco situações distintas.

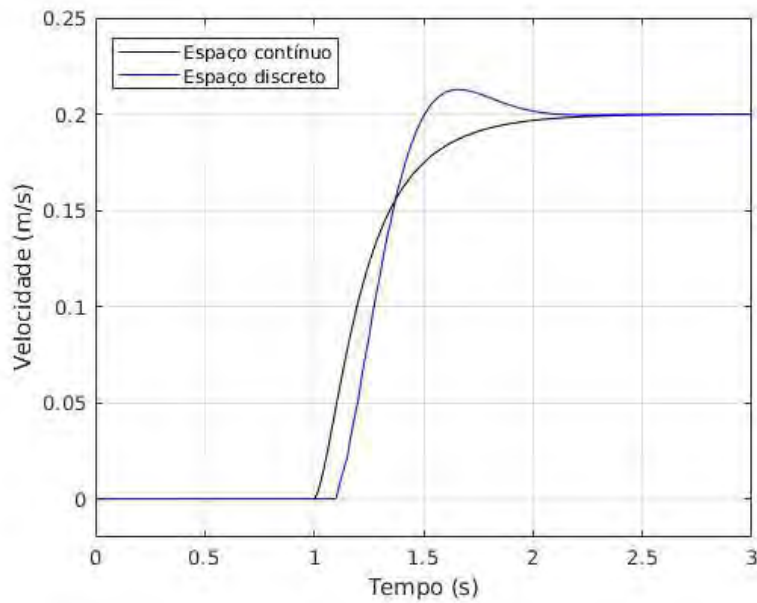


Figura 101 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas

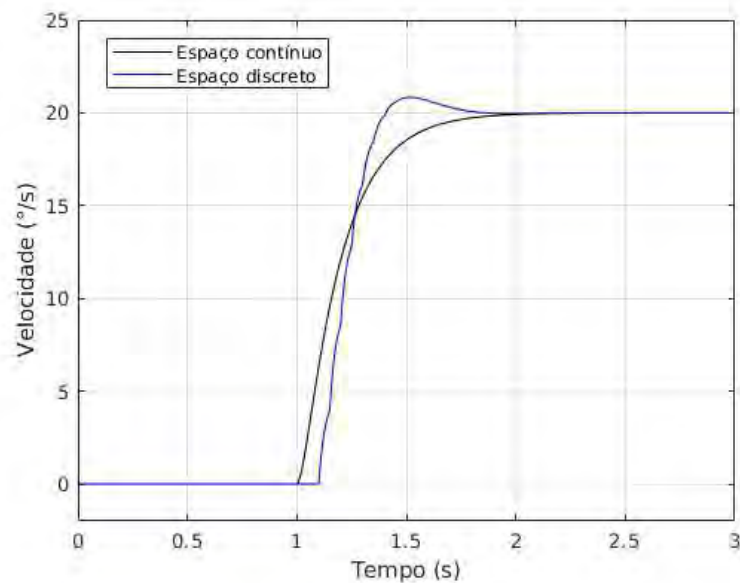


Figura 102 – Comparação entre as respostas de velocidade angular das malhas contínuas e discretas devido a uma entrada em degrau de velocidades combinadas

Situação 1: valores nominais que estão sendo utilizados até o momento com massa de 3 kg e coeficiente de atrito 0.7.

Situação 2: massa mínima de 3 kg e coeficiente de atrito mínimo de 0.56.

Situação 3: massa mínima de 3 kg e coeficiente de atrito máximo de 0.84.

Situação 4: massa máxima de 5 kg e coeficiente de atrito mínimo de 0.56.

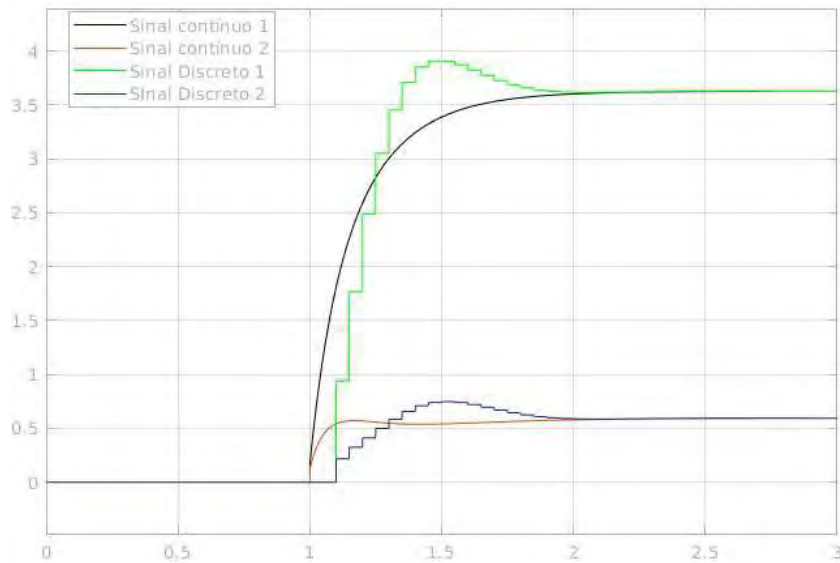


Figura 103 – Comparação entre os sinais de controle do controlador contínuo e discreto para uma entrada em degrau de velocidades combinadas

Situação 5: massa máxima de 5 kg e coeficiente de atrito máximo de 0.84.

Na Figura 104 e Figura 105 são mostradas as respostas do sistema em cada uma das situações, mostrando que as diferentes respostas, apesar de apresentarem variações diferentes, não interferem significativamente na performance do veículo. Por mais que esses resultados não comprovem matematicamente que o controle digital também é robusto, já que a teoria para provar isso está fora do escopo do trabalho, ele fornece uma verificação empírica dessa robustez.

7.4 Validação do Controle no Hardware (*Hardware in the Loop Simulation*)

Apesar do sistema apresentar bons resultados na simulação discreta em ambiente Simulink nada garante que o Hardware conseguirá realizar tais tarefas, já que isso depende de uma característica interna de velocidade de processamento. A fim de se avaliar essa possibilidade pode-se testar o Hardware dentro da própria malha do Simulink, um tipo de teste denominado *Hardware in Loop Simulation*. Esse modelo é mostrado na Figura 106, na qual o Simulink envia os sinais da integral do erro para o arduino, por meio de uma comunicação serial, este por sua vez realiza os seus cálculos internamente, por meio de um programa em C escrito no microcontrolador (que se encontra no Anexo B), e retorna os sinais de controle para a malha do Simulink. Enquanto que na Fig 107 se tem o subsistema que realiza a comunicação e conversão de sinais para a compatibilidade entre o arduino e o Simulink.

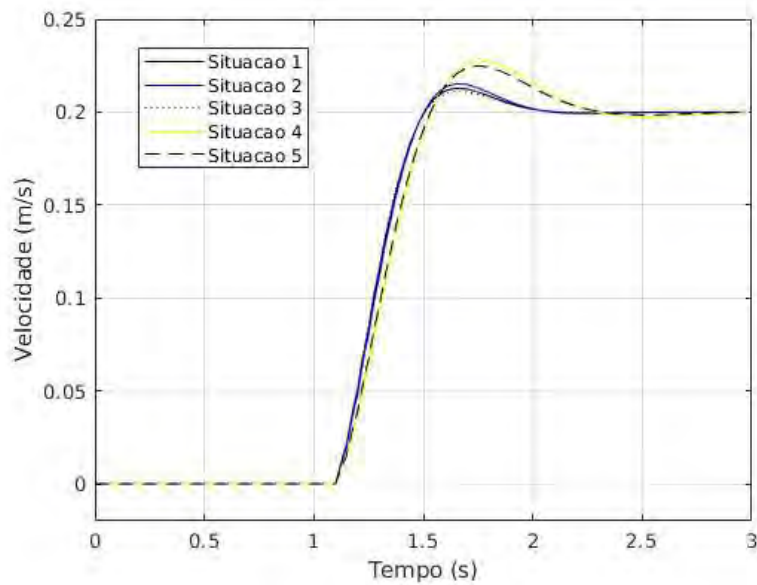


Figura 104 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas

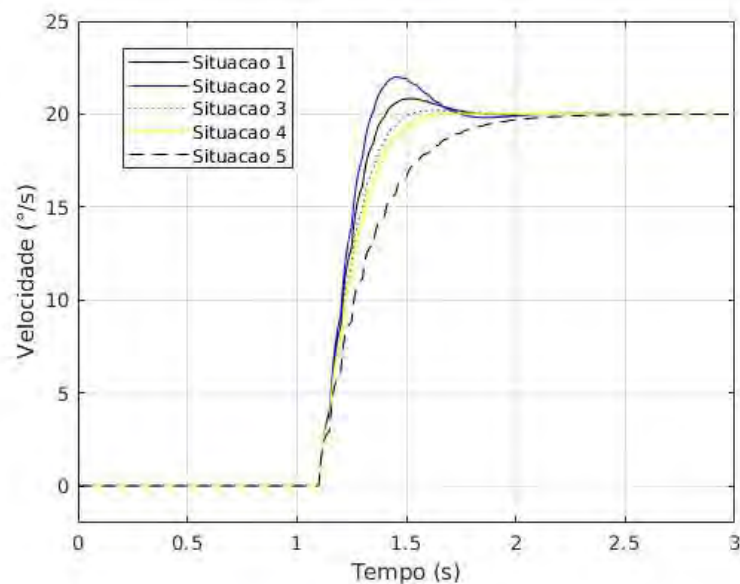


Figura 105 – Comparação entre as respostas de velocidade angular das malhas contínuas e discretas devido a uma entrada em degrau de velocidades combinadas

Esse método de verificação é de extrema importância por ser uma forma mais simples e que não exige que todo o sistema seja testado, isto é, com o veículo para que se analise a aplicabilidade do controle no *Hardware* que se deseja utilizar.

Percebe-se que não há diferenças significativas de desempenho entre o sistema no Matlab e o teste utilizando o arduino na malha. Pois ambas as curvas estão praticamente sobrepostas perfeitamente, como mostrado nas Figura 110 e Figura 111. Isso garante que

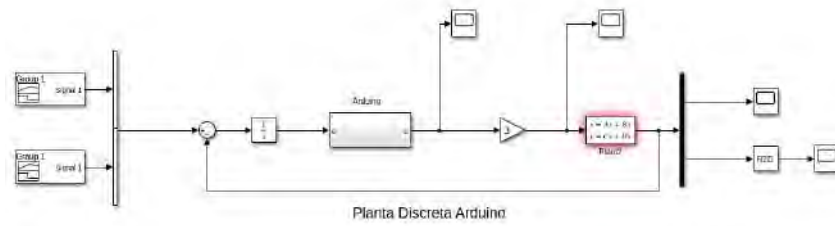


Figura 106 – Comparação entre as respostas de velocidade angular das malhas contínua e discreta devido a uma entrada em degrau de velocidades combinadas



Figura 107 – Comparação entre as respostas de velocidade angular das malhas contínuas e discretas devido a uma entrada em degrau de velocidades combinadas

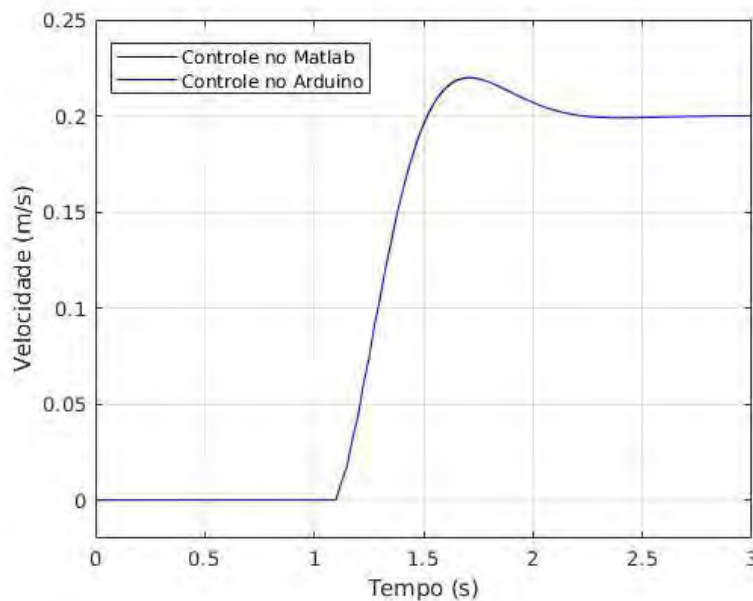


Figura 108 – Comparação entre as respostas de velocidade linear do controlador no Matlab e no Arduino devido a uma entrada em degrau de velocidades combinadas

o arduino possa ser utilizado como controlador digital.

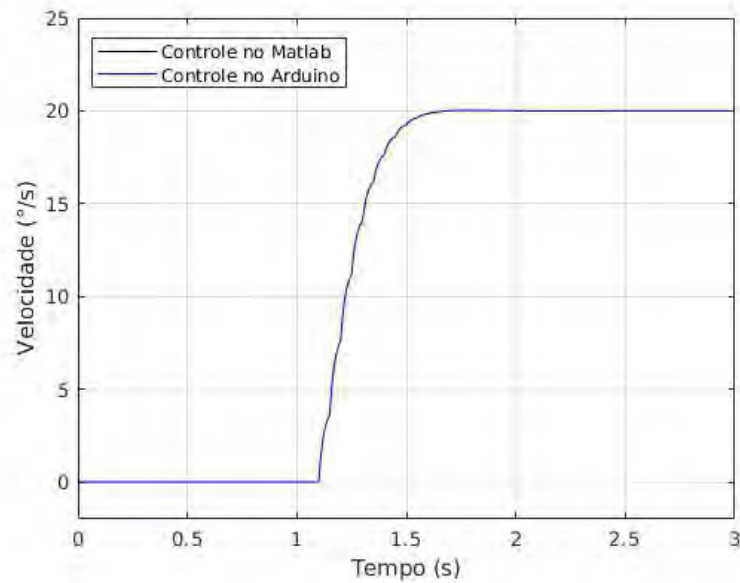


Figura 109 – Comparação entre as respostas de velocidade angular do controlador no Matlab e no Arduino devido a uma entrada em degrau de velocidades combinadas

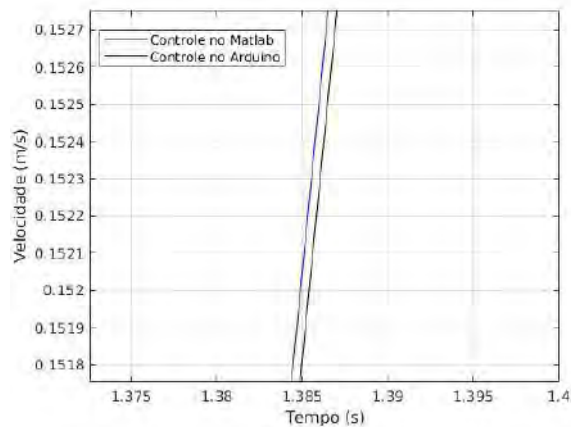


Figura 110 – Recorte das respostas dos sistemas para a saída de velocidade longitudinal

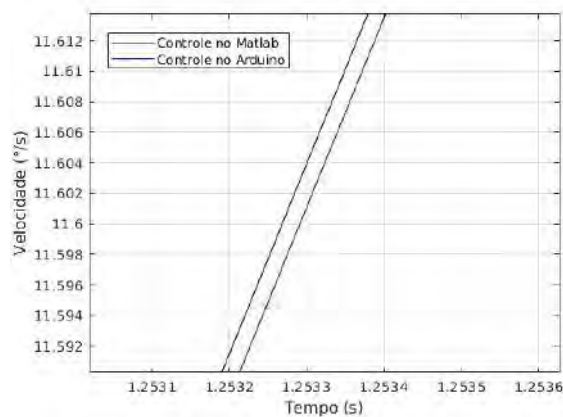


Figura 111 – Recorte das respostas dos sistemas para a saída de velocidade angular

8 TESTES NO VEÍCULO

A partir da construção do veículo desenvolvida no capítulo 6, chega-se ao resultado mostrado na Figura 112 e Figura 113. Na parte superior do veículo são colados a câmera, o *raspberry pi* e a alimentação do microcomputador. Os demais componentes, também descritos no capítulo 6, estão na parte de dentro do veículo.



Figura 112 – Visão frontal da montagem do veículo



Figura 113 – Visão lateral da montagem do veículo

Primeiramente, buscou-se realizar testes de funcionalidade do sistema descrito no capítulo 6, utilizando um computador remoto como mestre para o sistema ROS do *raspberrypi* cujo código implementado no controlador está presente no anexo C. Entretanto, tais testes apresentaram um grave problema de comunicação serial entre o arduino e o *raspberrypi*, com perdas frequentes de comunicação. Esse problema foi solucionado apenas quando o código presente no Anexo D foi implementado, sendo a sua única diferença o fato de não apresentar a comunicação com o sensor, ou seja, essa comunicação com o sensor IMU está causando perdas de informação no sistema. As causas disso podem ser diversas, mas muito provavelmente isso se deve ou a dificuldade do arduino em processar essa quantidade de informações ou a algum problema de frequência de operação do sistema com o sensor.

Devido a esse problema de conexão o controle desenvolvido no capítulo 7 não poderá ser testado, já que não há informação dos sensores. Portanto, são realizados apenas testes envolvendo o controle a distância do veículo, por meio das imagens da câmera.

Primeiramente, é realizado um teste de controle do veículo em um ambiente distante do operador. Na Figura 114 e Figura 115 estão os *qr codes* que levam aos vídeos do teste enquanto que na Figura 116 está presente o *qr code* que leva à visão do operador do veículo. Percebe-se o movimento de rotação do carro está muito agressivo, realizando o movimento de forma muito rápida. Apesar de apresentar problemas de controlabilidade, foi necessário implementar isso no código pois o veículo possui problemas de rotação do motor quando precisa rotacionar para a direita, não possuindo torque suficiente para girar o veículo. Portanto, para compensar esse problema o torque geral de rotação teve que ser aumentado.



Figura 114 – *QR code* relativo ao vídeo do teste do veículo controlado a distância

Outro teste realizado é o de velocidade máxima do veículo, já que ele apresenta uma função de turbo que o operador pode utilizar. Na Figura 117 está indicado o *qr code* que leva ao vídeo desse teste em um corredor, ou seja, não há a necessidade de se realizar curvas, apenas se movimentar para frente ou para trás. Enquanto que na Figura 118 está presente a visão do operador da situação.

Percebe-se que esse modo é muito útil já que permite que tanto ocasiões nas quais é necessário maior velocidade quanto naquelas em que é requerido maior precisão sejam



Figura 115 – *QR code* complementar relativo ao vídeo do teste do veículo controlado a distância



Figura 116 – *QR code* relativo à visão do operador do veículo



Figura 117 – *QR code* complementar relativo ao vídeo do teste do modo de aumento da velocidade do veículo



Figura 118 – *QR code* relativo à visão do operador do veículo no teste de aumento de velocidade

atendidas.

Além disso, é importante salientar que a operação do veículo dependerá diretamente da qualidade das imagens recebidas, sendo isso condicionado a dois fatores: a qualidade da câmera propriamente dita e a da conexão de rede entre o microcomputador do veículo e mestre. Portanto, alguns atrasos e travamentos que existem nas imagens, conforme pode ser verificado nos vídeos, podem ser amenizados melhorando esses fatores.

9 CONCLUSÕES

O objetivo do presente trabalho é de utilizar um ambiente de simulação para o teste de diversas funcionalidades de interesse militar e aplica-las a um rover comercial. Primeiramente, na análise da simulação, são modelados o veículo, que serve como base móvel, e o seu manipulador, que possui 4 graus de liberdade. Com o conjunto criado e montado foi necessário testar o *driver* de controle de velocidade do veículo, que faz o a relação entre a velocidade requerida do veículo e a atingida. Chegando-se a conclusão de que ele apresenta um resultado satisfatório, apesar das variações encontradas no deslocamento angular do veículo, já que isso em muito se deve a própria forma de cálculo do simulador.

Posteriormente, foram apresentadas as formas de manipulação possível do braço robótico: por meio da interface gráfica, do armazenamento de posições no banco de dados e controle junta a junta. Cada método possui seus pontos positivos e negativos e ficará a cargo do operador escolher a adequada de acordo com cada situação.

Nos testes de mapeamento foi possível perceber que em ambiente interno se tem como resultado um mapa mais rico em detalhes. Isso se deve a utilização de sensores a laser, que dependem das reflexão de seus sinais. Isso se deve a própria natureza do sistema e não necessariamente a problemas do modelo. E, além disso, por mais que a simulação em ambiente interno tenho tido um resultado melhor, aquela realizada em ambiente externo consegue fornecer informações importantes para um operador, no caso de um veículo de reconhecimento por exemplo. O deslocamento autônomo também apresentou bons resultados, tendo o veículo atingido o objetivo de forma satisfatória e evitando obstáculos no meio da caminho.

Até esse ponto, não há um sistema de controle para a velocidade do veículo, que possa garantir que o sistema atinja os comandos de velocidades especificados pelo operador ou computador de bordo. Para tanto, foi desenvolvido um sistema de controle que fornece comandos para os motores elétricos do veículo. Além disso, o modelo desenvolvido é baseado na teoria de controle robusto e portanto é sintetizado para resistir a variações do sistema, algo de suma importância em aplicações militares. No presente estudo foram considerados variações de massa, representando diferentes carregamentos, e de coeficiente de atrito, representando diferentes tipos de terrenos. O controle desenvolvido foi testado em ambiente computacional e apresentou uma boa robustez e baixa interferência entre as saídas, ambos requisitos de desempenho muito importantes.

O controle desenvolvido está no espaço contínuo, porém será aplicado em um computador digital e, para otimizar a utilização do Hardware, deve-se discretiza-lo. Tal processo utiliza o método de zero order hold para obter o controle no espaço discreto. Para o

passo de discretização foi escolhido um valor relativamente alto de 50 milisegundos, tal valor acarreta uma perda de informação elevada para o sistema e acarreta em pequenos overshoots nas respostas aos degraus de velocidade. Entretanto, isso facilita a implementação em um Hardware de baixo desempenho como o arduino.

Para comprovar a aplicabilidade do controle digital na placa foi realizado um teste de Hardware in Loop Simulation, que faz com que o sistema físico seja representado pela malha do Simulink, enquanto que os cálculos do controlador são realizados diretamente pelo arduino. Esse teste mostrou que a resposta foi adequada, praticamente não havendo diferenças entre esse teste e as simulações realizadas anteriormente.

Os testes do controlador não puderam ser realizados, devido a problemas de perda de comunicação serial entre o arduino e o *raspberrypi* quando o sensor inercial estava em funcionamento do arduino. Isso pode ser resolvido por meio da utilização de um microcontrolador mais robusto ou da implementação da leitura dos sinais por meio da GPI do *raspberrypi*. Entretanto, os testes de controle do veículo operado a distância, por meio de uma lei de controle em malha aberta, foram realizados e mostraram resultados satisfatórios.

Como oportunidades para trabalhos futuros destacam-se a identificação do sistema do veículo, já que isso pode gerar uma planta mais fiel ao sistema real e, por consequência, um controlador mais eficiente. Além disso, a aquisição de sensores *lidar* e de *encoders* possibilita que a simulação feita no gazebo possa ser implementada, desde os métodos de controle do manipulador até os algoritmos de deslocamento autônomo. Ainda, há a possibilidade de que, dado que existam esses sensores, se implemente sistema de múltiplos veículos autônomos, compartilhando informações para a execução de uma tarefa, desde uma função logística até de combate.

REFERÊNCIAS

- 1 JOSEPH, L. *Mastering ROS for Robotics Programming*. 2. ed. Birmingham B3 2PB, UK: Publishing Ltd, 2015. 481 p.
- 2 GANDHINATHAN, R. *ROS Robotics Projects*. 1. ed. [S.l.: s.n.], 2012. 449 p.
- 3 THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. 1. ed. [S.l.: s.n.], 2000. 492 p.
- 4 GAZEBO. *Physics Parameters*. 2022. 11 de maio de 2022. Disponível em: <https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics>.
- 5 ROBOTIS. *OpenManipulator-X*. 2022. 11 de maio de 2022. Disponível em: <https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/>.
- 6 CHIKURTEV, D. Indoor navigation for service mobile robots using robot operating system indoor navigation for service mobile robots using robot operating system (ros). 2020.
- 7 ROS, W. *Move_base*. 2022. 13 de maio de 2022. Disponível em: <http://wiki.ros.org/move_base>.
- 8 ZHENG, K. Ros navigation tuning guide. *Studies in Computational Intelligence*, v. 962, 2021.
- 9 CODESCRIPT. *2x12 Sabertooth Motor Driver*. 2022. 20 de maio de 2022. Disponível em: <<https://thecodescripts.com/product/212-sabertooth-motor-driver/>>.
- 10 RIVERA, Z.; GUIDA, M. de; DOMINICI, S. Unmanned ground vehicle modelling in gazebo/ros-based environments. *Machines*, v. 7, 2019.
- 11 HERNANDEZ-MENDEZ, S.; MALDONADO-MENDEZ, C.; MARIN-HERNANDEZ, A. Design and implementation of a robotic arm using ros and moveit! *2017 IEEE International Autumn Meeting on Power, Electronics and Computing, ROPEC 2017*, 2017.
- 12 MEGALINGAM, R. K.; KATTA, N.; GEESALA, R. Keyboard-based control and simulation of 6-dof robotic arm using ros. *2018 4th International Conference on Computing Communication and Automation, ICCCA 2018*, 2018.
- 13 GATESICHAPAKORN, S.; TAKAMATSU, J.; RUCHANURUCKS, M. Ros based autonomous mobile robot navigation using 2d lidar and rgb-d camera. *2019 1st International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics, ICA-SYMP 2019*, 2019.
- 14 YAGFAROV, R.; IVANOU, M.; AFANASYEV, I. Map comparison of lidar-based 2d slam algorithms using precise ground truth. *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, 2018.
- 15 BALASURIYA, B. L.; CHATHURANGA, B. A.; JAYASUNDARA, B. H. Outdoor robot navigation using gmapping based slam algorithm. 2016.

- 16 NORZAM, W. A.; HAWARI, H. F.; KAMARUDIN, K. Analysis of mobile robot indoor mapping using gmapping based slam with different parameter. *IOP Conference Series: Materials Science and Engineering*, v. 705, 2019.
- 17 GAZEBO. *Gazebo Tutorials*. 2022. 12 de maio de 2022. Disponível em: <<https://classic.gazebosim.org/tutorials?cat=physics>>.
- 18 DUDEK, G.; JENKIN, M. *Computational Principles of Mobile Robotics*. 1. ed. [S.l.: s.n.], 2010. 0 p.
- 19 ROS.ORG. *ros_control*. 2022. 11 de maio de 2022. Disponível em: <http://wiki.ros.org/ros_control>.
- 20 GALACTIC, M. *MoveIt 2 Documentation*. 2022. 11 de maio de 2022. Disponível em: <<https://moveit.picknik.ai/galactic/index.html>>.
- 21 GALACTIC, M. *Planning Scene Monitor*. 2022. 11 de maio de 2022. Disponível em: <https://moveit.picknik.ai/galactic/doc/concepts/planning_scene_monitor.html>.
- 22 SUCAN, I. *trajectory_processing::TimeOptimalTrajectoryGeneration*. 2022. 11 de maio de 2022. Disponível em: <https://docs.ros.org/en/noetic/api/moveit_core/html/cpp/classtrajectory__processing_1_1TimeOptimalTrajectoryGeneration.html>.
- 23 BALLARDINI, A. L.; FONTANA, S.; FURLAN, A.; SORRENTI, D. G. ira laser tools: a ros laserscan manipulation toolbox. *Studies in Computational Intelligence*, v. 962, 2014. Disponível em: <<https://arxiv.org/pdf/1411.1086.pdf>>.
- 24 ROS, W. *Gmapping*. 2022. 13 de maio de 2022. Disponível em: <<http://wiki.ros.org/gmapping>>.
- 25 ANSWERS, R. *Rotation error in Gazebo simulation*. 2022. 13 de maio de 2022. Disponível em: <<https://answers.ros.org/question/9640/rotation-error-in-gazebo-simulation/>>.
- 26 UBUNTU. *How to install Ubuntu Server on your Raspberry Pi*. 2022. 18 de maio de 2022. Disponível em: <<https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview>>.
- 27 ANSWERS, R. *ROS remote master: can see topics but no data*. 2022. 18 de maio de 2022. Disponível em: <<https://answers.ros.org/question/90536/ros-remote-master-can-see-topics-but-no-data/>>.
- 28 RASPBIAN. *Raspbian*. 2022. 18 de maio de 2022. Disponível em: <<http://www.raspbian.org/>>.
- 29 ANSWERS, R. *Raspicam node on ARM64 Raspberry pi*. 2022. 20 de maio de 2022. Disponível em: <<https://discourse.ros.org/t/raspicam-node-on-arm64-raspberry-pi/23311>>.
- 30 UBIQUITYROBOTICS. *UbiquityRobotics/raspicam node*. 2022. 20 de maio de 2022. Disponível em: <https://github.com/UbiquityRobotics/raspicam_node>.
- 31 INVENSENSE. *MPU-6000 and MPU-6050-Product Specification-Revision 3.4*. 2022. 20 de maio de 2022. Disponível em: <<https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>>.

- 32 ENGINEERING, D. *Sabertooth User's Guide*. 2022. 20 de maio de 2022. Disponível em: <<https://www.dimensionengineering.com/datasheets/Sabertooth2x12.pdf>>.
- 33 PIEDBOEUF JC., d. C. J. . H. R. Friction and stick-slip in robots: Simulation and experimentation. *Multibody System Dynamics*, v. 4, 2000. 341-354. Disponível em: <<https://arxiv.org/pdf/1411.1086.pdf>>.
- 34 FERNANDES, D. L. *Controle Robusto para Sistemas de Armas Embarcado em Veículo Terrestre*. 120 p. Mestrado em Engenharia Mecânica — Instituto Militar de Engenharia, Rio de Janeiro, 2018. Disponível em: <http://www.ime.eb.mil.br/images/arquivos/pos-graduacao/mecanica/2018_Dissertacao_Diogo_Lopes_Fernandes.pdf>.
- 35 GLOVER, K.; DOYLE, J. C.; ZHOU, K. *Robust and Optimal Control*. 1. ed. [S.l.: s.n.], 1996.

ANEXO A – PARÂMETROS DE DESLOCAMENTO AUTÔNOMO

DWAPlannerROS:

```
# Robot Configuration Parameters
max_vel_x: 0.50
min_vel_x: -0.50

max_vel_y: 0.0
min_vel_y: 0.0

# The velocity when robot is moving in a straight line
max_vel_trans: 0.50
min_vel_trans: 0.1

max_vel_theta: 1.82
min_vel_theta: 0.5

acc_lim_x: 2.5
acc_lim_y: 0.0
acc_lim_theta: 3.2

# Goal Tolerance Parametes
xy_goal_tolerance: 0.05
yaw_goal_tolerance: 0.17
latch_xy_goal_tolerance: false

# Forward Simulation Parameters
sim_time: 2.0
vx_samples: 20
vy_samples: 0
vth_samples: 40
controller_frequency: 10.0

# Trajectory Scoring Parameters
```

```
path_distance_bias: 32.0
goal_distance_bias: 20.0
occdist_scale: 0.02
forward_point_distance: 0.325
stop_time_buffer: 0.2
scaling_speed: 0.25
max_scaling_factor: 0.2

# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05

# Debugging
publish_traj_pc : true
publish_cost_grid_pc: true
```

ANEXO B – PROGRAMA DO ARDUINO PARA O TESTE DE HARDWARE

```

#include <BasicLinearAlgebra.h> //by default the matrices are floats

union BtoF
{
    byte b[16];
    float fval;
} u;

const int buffer_size = 16;
byte buf[buffer_size];

    float intErrorV;
    float intErrorW;

    BLA::Matrix<3, 3> A = {0.3598, 0.1402, 0.0596, -0.0047,
-0.0018, -0.0008,0.1024, 0.0399, 0.0170};
    BLA::Matrix<3, 2> B = {0.0003,0.1796,0.3590, -0.4964,0.0172, -0.1094};
    BLA::Matrix<2, 3> C = {13.9, 249.1, -1242.9, -74.7, 234.4, -944.7};

    BLA::Matrix<3> ActualSpaceState = {0, 0, 0};
    BLA::Matrix<3> PastSpaceState = {0, 0, 0};
    BLA::Matrix<2> U = {0, 0};
    BLA::Matrix<2> Y = {0, 0};

void setup() {
    Serial.begin(115200,SERIAL_8N1);
}

void loop() {
    if (Serial.available()>0)

```



```
{
  U(0) = readFromMatlab(0);
  U(1) = readFromMatlab(4);
  delay(0.1);

  ActualSpaceState = A*PastSpaceState + B*U;
  Y = C*ActualSpaceState;
  PastSpaceState = ActualSpaceState;
  writeToMatlab(Y(0), Y(1));
}
delay(50);

}

float readFromMatlab(int entrada)
{
  int reln = Serial.readBytes( buf, buffer_size);
  for (int i=0; i<4; i++)
  {
    u.b[i] = buf[i+entrada];
  }
  float output = u.fval;
  return output;
}

void writeToMatlab (float number1, float number2)
{
  byte *b1 = (byte *) &number1;
  byte *b2 = (byte *) &number2;
  Serial.write(b1,4);
  Serial.write(b2,4);
  Serial.write(13); //"\r" CR
  Serial.write(10); // "\n" LF
}
```

ANEXO C – PROGRAMA DO ARDUINO PARA O TESTE DO SISTEMA COM A LEITURA DOS SENSORES

```

#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Float64MultiArray.h>
#include <std_msgs/Empty.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>
// Inclui a biblioteca Wire que possui as funções de comunicação I2C:
#include <Wire.h>

Servo myservo_longitudinal;
Servo myservo_rotate;

int rotate = 7; // LED connected to pin 7
int foward = 6; // LED connected to pin 6

// Endereco I2C do sensor MPU-6050
const int MPU = 0x68;
// Variaveis para armazenar valores do sensor
float AccX, AccY, AccZ, Temp, GyrX, GyrY, GyrZ;

ros::NodeHandle nh;

void messageCb(const geometry_msgs::Twist& msg)
{
    float x = msg.linear.x;
    float z_rotation = msg.angular.z;
    int out_rotate = 98 + 30*z_rotation;
    int out_foward = 98 + 30*x;
    myservo_rotate.write(out_rotate);
    myservo_longitudinal.write(out_foward);
}

```

```
ros::Subscriber<geometry_msgs::Twist> sub("/cmd_vel", messageCb);
```

```
std_msgs::Float64MultiArray myData;
```

```
ros::Publisher chatter("IMU", &myData);
```

```
std_msgs::Float64 Tempe;
```

```
ros::Publisher temp("Temperatura", &Tempe);
```

```
void setup()
```

```
{
```

```
  Serial1.begin(9600);
```

```
  nh.initNode();
```

```
  nh.advertise(chatter);
```

```
  nh.advertise(temp);
```

```
  nh.subscribe(sub);
```

```
  pinMode(rotate,OUTPUT);
```

```
  pinMode(foward,OUTPUT);
```

```
// Inicializa o MPU-6050
```

```
Wire.begin();
```

```
Wire.beginTransmission(MPU);
```

```
Wire.write(0x6B);
```

```
Wire.write(0);
```

```
Wire.endTransmission(true);
```

```
// Configura Giroscopio para fundo de escala desejado
```

```
/*
```

```
  Wire.write(0b00000000); // fundo de escala em +/-250 /s
```

```
  Wire.write(0b00001000); // fundo de escala em +/-500 /s
```

```
  Wire.write(0b00010000); // fundo de escala em +/-1000 /s
```

```
  Wire.write(0b00011000); // fundo de escala em +/-2000 /s
```

```
*/
```

```
Wire.beginTransmission(MPU);
```

```
Wire.write(0x1B);
```

```
Wire.write(0b00000000); // Trocar esse comando para fundo de escala de
```

```
Wire.endTransmission();
```

```

// Configura Acelerometro para fundo de escala desejado
/*
  Wire.write(0b00000000); // fundo de escala em +/-2g
  Wire.write(0b00001000); // fundo de escala em +/-4g
  Wire.write(0b00010000); // fundo de escala em +/-8g
  Wire.write(0b00011000); // fundo de escala em +/-16g
*/
Wire.beginTransmission(MPU);
Wire.write(0x1C);
Wire.write(0b00000000); // Trocar esse comando para fundo de escala de
Wire.endTransmission();

myservo_rotate.attach(7); // Use PWM pin 7 to control Sabertooth.
myservo_longitudinal.attach(6);
}

void loop()
{
  // Comandos para iniciar transmiss o de dados
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 14, true); // Solicita os dados ao sensor

  // Armazena o valor dos sensores nas variaveis correspondentes
  AccX = Wire.read() << 8 | Wire.read(); //0x3B (ACCEL_XOUT_H) & 0x3C (AC
  AccY = Wire.read() << 8 | Wire.read(); //0x3D (ACCEL_YOUT_H) & 0x3E (AC
  AccZ = Wire.read() << 8 | Wire.read(); //0x3F (ACCEL_ZOUT_H) & 0x40 (AC
  Temp = Wire.read() << 8 | Wire.read(); //0x41 (TEMP_OUT_H) & 0x42 (TEMP
  GyrX = Wire.read() << 8 | Wire.read(); //0x43 (GYRO_XOUT_H) & 0x44 (GYF
  GyrY = Wire.read() << 8 | Wire.read(); //0x45 (GYRO_YOUT_H) & 0x46 (GYF
  GyrZ = Wire.read() << 8 | Wire.read(); //0x47 (GYRO_ZOUT_H) & 0x48 (GYF

  // Imprime na Serial os valores obtidos
  /* Alterar divis o conforme fundo de escala escolhido:
    Aceler metro
    +/-2g = 16384
    +/-4g = 8192
  */

```

```
+/-8g = 4096
+/-16g = 2048

Girosc pio
+/-250 /s = 131
+/-500 /s = 65.6
+/-1000 /s = 32.8
+/-2000 /s = 16.4
*/

// convers o para os valores requeridos
AccX = AccX / (1670.1325) - 0.5;
AccY = AccY / (1670.1325);
AccZ = AccZ / (1670.1325) - 9.81;

GyrX = GyrX / 131;
GyrY = GyrY / 131;
GyrZ = GyrZ / 131;

float sensor [6] = {AccX, AccY, AccZ, GyrX, GyrY, GyrZ};

myData.data = sensor;
myData.data_length = 6;

Temp = Temp / 340.00 + 36.53;
Tempe.data = Temp;

 chatter . publish (&myData );

 temp . publish (&Tempe);

nh . spinOnce ();
delay (50);
}
```

ANEXO D – PROGRAMA DO ARDUINO PARA O TESTE DO SISTEMA SEM A LEITURA DOS SENSORES

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Float64MultiArray.h>
#include <std_msgs/Empty.h>
#include <geometry_msgs/Twist.h>
#include <Servo.h>

Servo myservo_longitudinal;
Servo myservo_rotate;

int rotate = 7; // LED connected to pin 7
int foward = 6; // LED connected to pin 6

ros::NodeHandle nh;

void messageCb(const geometry_msgs::Twist& msg)
{
    float x = msg.linear.x;
    float z_rotation = msg.angular.z;
    int out_rotate = 98 + 60*z_rotation;
    int out_foward = 98 + 60*x;
    myservo_rotate.write(out_rotate);
    myservo_longitudinal.write(out_foward);
}

ros::Subscriber<geometry_msgs::Twist> sub("/cmd_vel", messageCb);

void setup()
{
    Serial.begin(9600);;
```

```
nh.initNode();
nh.subscribe(sub);

pinMode(rotate ,OUTPUT);
pinMode(foward ,OUTPUT);

myservo_rotate.attach(7); // Use PWM pin 7 to control Sabertooth.
myservo_longitudinal.attach(6);
}

void loop()
{
  nh.spinOnce();
  delay(50);
}
```