

Universidade Federal de Juiz de Fora  
Faculdade de Engenharia  
Engenharia Elétrica – Habilitação em Robótica e Automação Industrial

**Alexander Silva Barbosa**

**Robot One - Simulador Robótico**

Juiz de Fora  
2018

**Alexander Silva Barbosa**

**Robot One - Simulador Robótico**

Trabalho de Conclusão de Curso apresentado  
à Faculdade de Engenharia da Universidade  
Federal de Juiz de Fora, como requisito para  
obtenção do grau de Engenheiro Eletricista.

Orientador: Exuperry Barros Costa

Juiz de Fora

2018

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF  
com os dados fornecidos pelo(a) autor(a)

Barbosa, Alexander Silva.

Robot One - Simulador Robótico / Alexander Silva Barbosa. – 2018.  
59 f. : il.

Orientador: Exuperry Barros Costa

Trabalho de Conclusão de Curso – Universidade Federal de Juiz de Fora,  
Faculdade de Engenharia. Engenharia Elétrica – Habilitação em Robótica  
e Automação Industrial, 2018.

1. Robótica. 2. Simulador. 3. Robótica Móvel. I. Barros Costa,  
Exuperry, orient. II. Título.

**Alexander Silva Barbosa**

**Robot One - Simulador Robótico**

Trabalho de Conclusão de Curso apresentado  
à Faculdade de Engenharia da Universidade  
Federal de Juiz de Fora, como requisito para  
obtenção do grau de Engenheiro Eletricista.

Aprovado em: 05/12/2018

**BANCA EXAMINADORA**

---

Prof. Dr. Exuperry Barros Costa - Orientador  
Universidade Federal de Juiz de Fora

---

Professor Dr. Leonardo Rocha Olivi  
Universidade Federal de Juiz de Fora

---

M. Eng. Guilherme Marins Maciel  
Universidade Federal de Juiz de Fora



## ATA DE APRESENTAÇÃO DE TRABALHO FINAL DE CURSO

DATA DA DEFESA: 05/12/2018 – 15H00

CANDIDATO: ALEXANDER SILVA BARBOSA

ORIENTADOR: PROF. DR. EXUPERRY BARROS COSTA

TÍTULO DO TRABALHO: ROBOT ONE – SIMULADOR ROBÓTICO

BANCA EXAMINADORA/INSTITUIÇÃO:

PRESIDENTE: PROF. DR. EXUPERRY BARROS COSTA / UFJF

AVALIADOR 1: PROF. DR. LEONARDO ROCHA OLIVI / UFJF

AVALIADOR 2: M. ENG. GUILHERME MARINS MACIEL / UFJF

LOCAL: LABORATÓRIO DE ROBÓTICA E AUTOMAÇÃO (LABRA) – FACULDADE DE ENGENHARIA - UFJF

Nesta data, em sessão pública, após exposição oral de 23 minutos, o candidato foi arguido pelos membros da banca. Em decorrência desta arguição, a banca considerou o candidato:

() APROVADO

() REPROVADO

Na forma regulamentar foi lavrada a presente Ata que é abaixo assinada pelos membros da banca na ordem determinada e pelo candidato:

PRESIDENTE: Exuperry Barros Costa

AVALIADOR 1: Leonardo Rocha Olivi

AVALIADOR 2: Guilherme Marins Maciel

CANDIDATO: Alexander Silva Barbosa

Vistos: (coordenador do curso).



Prof. Exuperry Barros Costa  
Coordenador Engenharia Elétrica  
Robótica e Automação Industrial

## AGRADECIMENTOS

Inicialmente à Universidade Federal de Juiz de Fora pela excelente educação, estrutura e crescimento propiciados com ética e respeito. À todos os docentes e técnicos que tornaram tudo isto possível, dedicando seu tempo ensinando mais do que as áreas técnicas, mas nos ensinando a sermos adultos.

À meu orientador Exuperry Barros Costa, por todo o suporte, incentivo e correções mesmo que por um breve período de tempo, e também pela amizade e inspiração que levarei para o resto da vida.

Aos professores Leonardo Rocha Olivi e Ana Sophia Cavalcanti Alves Vilas Boas por todo o apoio, amizade e grandes ensinamentos que me propiciaram durante a graduação.

À toda equipe do Laboratório de Processamento de Sinais e Telecomunicações da UFJF, em especial aos professores Carlos Augusto Duque, Leandro Rodrigues Manso Silva e ao doutorando Carlos Henrique Nascimento Martins por todo apoio durante a graduação e inserção na área de pesquisa acadêmica.

Aos meus pais Edneia Farnese da Silva e Adalmo José Barbosa por todo o apoio durante minha graduação, pelo incentivo para não desistir mesmo quando as coisas não iam bem e pela inspiração diária.

Aos meus irmãos Igor Henrique Silva Barbosa e Jéssica da Silva Barbosa por estarem comigo sempre, dando apoio e suporte, mesmo quando estavam distantes fisicamente. Também por me inspirarem a continuar todos os dias.

À equipe Rinobot por todo o conhecimento e oportunidades no pouco tempo que estive na equipe.

Aos meus queridos amigos da Categoria SPL da Equipe Rinobot por cada momento de ensinamento, carinho e apoio no tempo que estive na categoria. Não consigo citar nenhum nome em especial pois todos os membros da categoria foram o melhor que eu poderia ter de amizade e apoio.

Ao meu fiel escudeiro José Humberto Barroso Bertelli por me apoiar sempre e estar comigo em todos os momentos, compartilhando comigo cada bom momento e sempre me ajudando a superar cada desafio, sendo meu ponto de apoio durante as incertezas.

“Technology is just a tool. In terms of getting the kids working together and motivating them, the teacher is the most important.”  
(Bill Gates)

## **RESUMO**

O estudo da robótica é algo que tem se tornado cada vez mais comum, uma vez que esta pode ser aplicada em praticamente todas as áreas da atualidade, desde a manipulação de objetos em uma linha de produção até o desenvolvimento de veículos autônomos. Apesar do desenvolvimento da robótica ter crescido consideravelmente, esta ainda é uma área pouco acessível, uma vez que, robôs no geral são equipamentos sensíveis e de custo elevado. Por serem equipamentos caros, durante a fase de implementação e testes iniciais os robôs podem ser substituídos por simuladores que replicam seu comportamento, permitindo assim que o robô possa ser utilizado virtualmente em um ambiente seguro e de custo reduzido. O grande desafio dos simuladores é replicar o comportamento do robô de forma realística, uma vez que no mundo real muitas vezes os robôs estarão sujeitos à diversos eventos que podem alterar sua percepção do mundo, como por exemplo: Erros na construção, erros em equipamentos, correntes de ar e fragmentos sólidos. O objetivo deste trabalho é o desenvolvimento um simulador robótico capaz de simular um robô móvel diferencial levando em consideração variáveis que alteram tanto o robô quanto o ambiente em que ele está inserido, tudo isto de uma forma de acesso simples para que este possa ser utilizado como ferramenta de ensino em aulas de robótica, além disso, o simulador deve ser multiplataforma e ser compatível com diversas linguagens de programação. O simulador desenvolvido se mostrou satisfatório para o que foi proposto, obtendo bons resultados em comparações com outras opções disponíveis.

Palavras-chave: Robótica. Simulador. Robótica Móvel.

## **ABSTRACT**

The study of robotics is something that has become increasingly common, since it can be applied in all areas, from the manipulation of objects on a production line to the development of autonomous vehicles. Although robotics has grown, this is an unobtainable area as robots are generally sensitive and costly devices. By being expensive equipments, during the implementation and testing phase, robots can be replaced by simulators that replicate their behavior, allowing that the robot can be used virtually in a safe and cost-effective environment. The great challenge of simulators is to replicate the behavior of the robot in a realistic way, since the real world robot will be prone to events that may change its perception of the world, such as: constructive errors, equipment errors, air currents and solid fragments. The objective of this work is the development of a robotic simulator able to simulate a differential mobile robot taking account the variables that changes the robot and the environment where it is inserted, all of this in a simple way of access so it can be used in robotics classes, in addition, the simulator must be multiplatform and be compatible with several programming languages. The developed simulator was satisfactory for what it was proposed, obtaining good results compared to other options available.

Key-words: Robotics. Simulator. Mobile Robotics.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Logo do Robot One . . . . .	15
Figura 2 – Interface do simulador Gazebo . . . . .	16
Figura 3 – Logo do framework ROS . . . . .	16
Figura 4 – Interface do simulador V-Rep . . . . .	17
Figura 5 – Tela de simulação do Player/Stage . . . . .	18
Figura 6 – Robô real: <i>Pioneer P3-DX</i> . . . . .	20
Figura 7 – Modelo 3D do robô simulado . . . . .	20
Figura 8 – Robô em um sistema de coordenadas cartesianas . . . . .	21
Figura 9 – Relações de velocidades cartesianas no robô . . . . .	23
Figura 10 – Leituras do GPS para um percurso circular . . . . .	26
Figura 11 – Feixes da leituras do lidar . . . . .	27
Figura 12 – Modelo 3D do <i>Lidar</i> . . . . .	28
Figura 13 – Leituras do lidar em frente à uma parede . . . . .	29
Figura 14 – Imagem do ambiente simulado, capturada com a câmera do robô . . . . .	30
Figura 15 – Representação da lei da gravitação universal para dois corpos . . . . .	31
Figura 17 – Exemplo de mapa simples para o robô . . . . .	35
Figura 18 – Exemplo de mapa simples para o robô . . . . .	36
Figura 19 – Exemplo de afinamento de imagens . . . . .	37
Figura 20 – Exemplo de mapa simples para o robô utilizando um arquivo de imagem	37
Figura 21 – Exemplo de mapa complexo para o robô utilizando um arquivo de imagem	38
Figura 22 – Exemplo de mapa complexo no simulador . . . . .	38
Figura 23 – Diagrama de blocos do controle PID clássico . . . . .	47
Figura 24 – Robô alcançando pontos utilizando o controle PID . . . . .	51
Figura 25 – Espaços de cores RGB e HSV . . . . .	53
Figura 26 – Processamento de imagem da câmera do robô . . . . .	54
Figura 27 – Imagem do robô seguindo a estrada no simulador . . . . .	55

## **LISTA DE TABELAS**

Tabela 1 – Estrutura de um elemento no arquivo de texto . . . . .	34
Tabela 2 – Pacote de dados enviado ao servidor . . . . .	41
Tabela 3 – Pacote de dados enviado ao cliente . . . . .	41
Tabela 4 – Funções para troca de dados entre o cliente e servidor . . . . .	43

## **LISTA DE ABREVIATURAS E SIGLAS**

ABNT Associação Brasileira de Normas Técnicas

UFJF Universidade Federal de Juiz de Fora

m Metro (Unidade de medida)

## LISTA DE SÍMBOLOS

$V_l$	Velocidade linear da roda esquerda
$V_r$	Velocidade linear da roda direita
$V$	Velocidade linear do robô
$\phi_l$	Velocidade angular da roda esquerda
$\phi_r$	Velocidade angular da roda direita
$\omega_l$	Contribuição da roda esquerda na velocidade angular do robô
$\omega_r$	Contribuição da roda direita na velocidade angular do robô
$\omega$	Velocidade angular do robô
$\dot{X}_I$	Velocidade no eixo X
$\dot{Y}_I$	Velocidade no eixo Y
$\dot{\theta}_I$	Velocidade rotacional
$X_r$	Posição do robô em X
$Y_r$	Posição do robô em Y
$\theta_r$	Ângulo do robô
$X_1$	Coordenada X do primeiro ponto
$Y_1$	Coordenada Y do primeiro ponto
$X_2$	Coordenada X do segundo ponto
$Y_2$	Coordenada Y do segundo ponto
$\sigma$	Desvio padrão
$\sigma_x$	Desvio padrão em X
$\sigma_y$	Desvio padrão em Y
$\sigma_\theta$	Desvio padrão rotacional
$\rho$	Distância
$Type$	Tipo do pacote de dados
$Status$	Status do pacote de dados

$Value$	Valor do pacote de dados
$Data_s$	Tamanho em bytes dos dados no pacote de dados
$Data$	Bytes do pacote de dados
$OP$	Operador do pacote de dados
$Name_s$	Tamanho em bytes do nome do atributo requisitado no pacote de dados
$Name$	Nome do atributo requisitado no pacote de dados

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>15</b>
<b>2</b>	<b>REVISÃO BIBLIOGRÁFICA . . . . .</b>	<b>16</b>
2.1	Gazebo . . . . .	16
2.2	V-Rep . . . . .	17
2.3	Player/Stage . . . . .	18
<b>3</b>	<b>SIMULADOR . . . . .</b>	<b>19</b>
3.1	Software de simulação . . . . .	19
3.1.1	Modelagem 3D . . . . .	19
3.1.2	Cinemática . . . . .	21
3.1.3	Sensores . . . . .	25
3.1.3.1	GPS . . . . .	25
3.1.3.2	<i>Lidar</i> . . . . .	27
3.1.3.3	Câmera RGB . . . . .	29
3.1.4	Física do Ambiente Simulado . . . . .	30
3.1.4.1	Gravitação . . . . .	30
3.1.4.2	Corpo rígido . . . . .	32
3.1.5	Modos de simulação . . . . .	33
3.1.5.1	Modo realístico . . . . .	33
3.1.5.2	Modo de estruturas dinâmicas (Maze) . . . . .	34
3.1.5.2.1	Arquivos de texto . . . . .	34
3.1.5.2.2	Arquivos de imagens . . . . .	36
3.2	Biblioteca de comunicação . . . . .	39
3.2.1	Servidor . . . . .	39
3.2.2	Protocolo de dados . . . . .	40
3.2.3	Funções . . . . .	42
<b>4</b>	<b>ESTUDOS DE CASO . . . . .</b>	<b>47</b>
4.1	Controle PID . . . . .	47
4.2	Seguidor de caminho . . . . .	53
<b>5</b>	<b>CONCLUSÕES E TRABALHOS FUTUROS . . . . .</b>	<b>57</b>
5.1	CONCLUSÕES . . . . .	57
5.2	TRABALHOS FUTUROS . . . . .	57



## 1 INTRODUÇÃO

Nas últimas décadas a robótica móvel demonstrou um notável desenvolvimento, sendo aplicada nas mais diversas áreas, como construção, montagem, lazer, agricultura e muitas outras áreas. Apesar do desenvolvimento rápido da robótica, no geral os equipamentos utilizados ainda são caros, o que muitas vezes os torna inacessíveis para grande parte da população.

Ao se trabalhar com robôs é essencial que diversos testes sejam realizados de forma a se garantir que tudo funcione de forma correta, neste contexto, ferramentas de simulação são extremamente valiosas pois permitem efetuar os testes com os robôs sem a necessidade da utilização do robô real, o que pode reduzir muito a possibilidade de problemas que possam danificar o robô.

Embora os simuladores sejam extremamente importantes, muitas vezes estas ferramentas são complicadas de serem utilizadas e requerem uma curva de aprendizado ingrime, além de algumas vezes requererem ferramentas específicas para sua utilização, levando à reescrita de códigos.

Nesta monografia foi proposto e desenvolvido um simulador capaz de simular com fidelidade um robô móvel em um ambiente tridimensional. Em um curso de robótica é imprescindível que os alunos tenham contato com robôs reais, porém, muitas vezes estes são caros e não estão disponíveis, portanto, uma ferramenta que permita a utilização do robô sem a necessidade de se ter um robô real em mãos é fundamental para que as aulas teóricas possam ser colocadas em prática.

Diversos simuladores estão disponíveis tanto de forma gratuita quanto paga, porém a maioria destes simuladores requer uma configuração precisa de diversos parâmetros o que muitas vezes impossibilita sua rápida utilização. Sendo assim, o simulador desenvolvido tem a proposta de ser facilmente utilizável e não requerer configurações sofisticadas, ao mesmo tempo que permita a simulação de ambientes complexos, com sensores geralmente não disponíveis na maioria dos simuladores.

O simulador desenvolvido foi nomeado *Robot One*, como um acrônimo da frase "*Robots for everyone*", que em português significa "Robôs para todos", uma vez que o simulador é gratuito, *open-source* e de uso simples para que possa ser utilizado por qualquer pessoa interessada em seu uso. O código fonte do *Robot One* está disponível em [1].



Figura 1 – Logo do Robot One

## 2 REVISÃO BIBLIOGRÁFICA

Este Capítulo tem como objetivo apresentar diversos simuladores amplamente utilizados em aplicações da robótica a fim de compará-los com o simulador desenvolvido, destacando suas vantagens e desvantagens. Diversos simuladores disponíveis permitem a simulação de robôs móveis com precisão, cada um tendo sua particularidade em relação aos outros. Alguns das principais plataformas serão descritas a seguir.

### 2.1 Gazebo

*Gazebo* [2] é um simulador geral de robótica, desenvolvido pela *Open Source Robotics Foundation*, e que permite simular diversos modelos de robôs e ambientes, como mostrado na Figura 2.

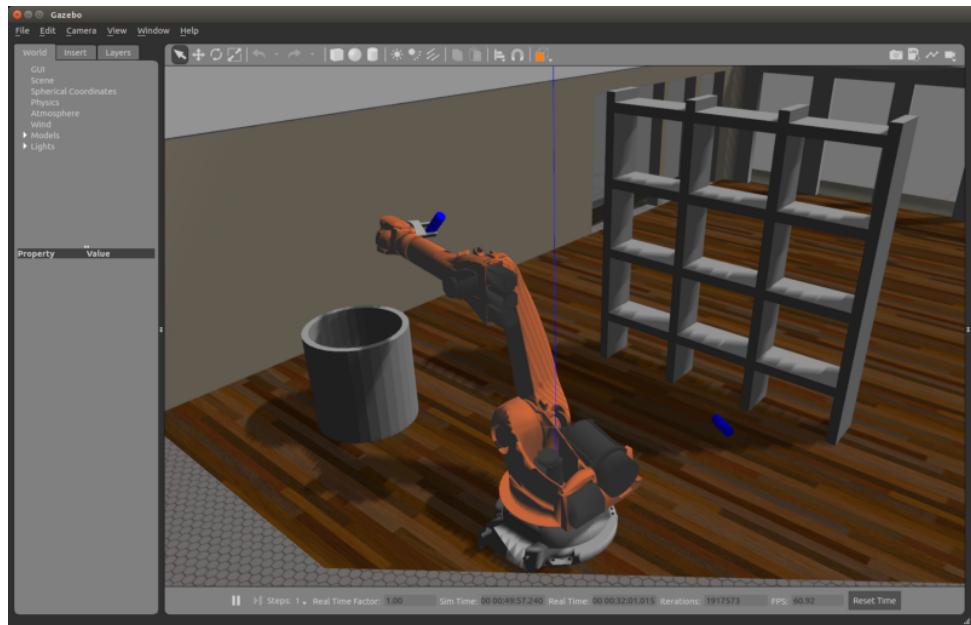


Figura 2 – Interface do simulador Gazebo

Por ser extremamente generalista, o Gazebo não permite a simulação sem prévia configuração do robô a ser usado. Uma desvantagem deste simulador é que o controle dos robôs no simulador deve ser feito com a *Gazebo API*, uma interface de comunicação e controle do simulador utilizando a linguagem C++, desta forma, não é possível utilizar nativamente outra linguagem que não seja o C++. Uma alternativa para se utilizar o gazebo com outras linguagens é utilizar o *ROS* (*Robot Operating System*) [3].



Figura 3 – Logo do framework ROS

O *ROS* é um *framework* para a criação de softwares para robôs. Ao utilizar o *ROS* com o *Gazebo* é possível utilizar de forma oficial, além do C++, as linguagens *Python* e *Lisp*.

Diferente do *Gazebo*, que foi idealizado para ter sua utilização totalmente em uma única linguagem, o simulador proposto pode nativamente ser utilizado em diversas linguagens de programação, porém, é notável que no atual estado, principalmente por sua maturidade, o *Gazebo* oferece uma vasta quantidade de recursos ainda não presentes no simulador desenvolvido.

## 2.2 V-Rep

O V-Rep [4] é um simulador de robótica baseado em uma arquitetura distribuída onde cada modelo ou objeto pode ser controlado individualmente através de scripts. A Figura 4 apresenta a interface do simulador V-Rep.

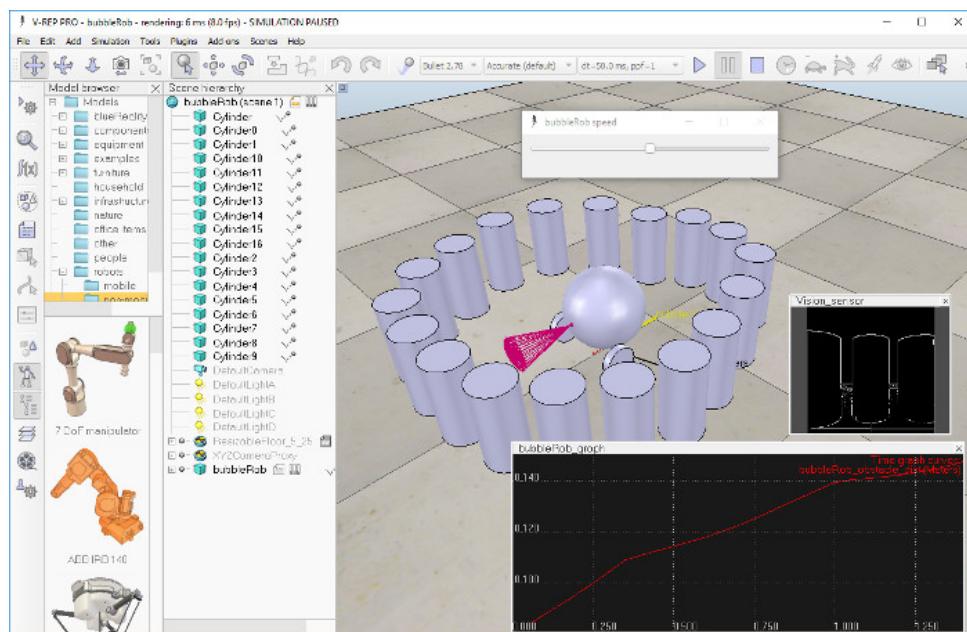


Figura 4 – Interface do simulador V-Rep

Como cada objeto pode ser controlado individualmente no V-Rep, criar uma simulação envolve mais do que simplesmente se conectar ao simulador e utilizar o robô, é necessário definir as juntas dos robôs de forma a se obter sua cinemática, o que muitas vezes faz com que haja um grande tempo configurando a simulação.

No simulador desenvolvido apenas os robôs são controlados externamente, sendo que o protocolo de controle do simulador oferece um acesso separado para controlar o robô já totalmente modelado ou cada uma de suas partes móveis. Objetos que não são o robô mas fazem parte da simulação não são acessíveis para controle, ficando à cargo da própria física do simulador cuidar destes objetos.

### 2.3 Player/Stage

*Player/Stage* [5] é um conjunto de simulador e um servidor em rede para o controle de robôs. O *Player* é a parte deste software responsável por criar um servidor TCP no robô ou no computador e permitir o seu controle através de uma *API*. Esta abordagem permite que o simulador não dependa de uma linguagem de programação específica.

O *Stage* é um simulador utilizado em conjunto com o *Player* e que permite a simulação de uma população de robôs móveis, objetos e sensores em um ambiente tridimensional ou bidimensional, como apresentado na Figura 5.

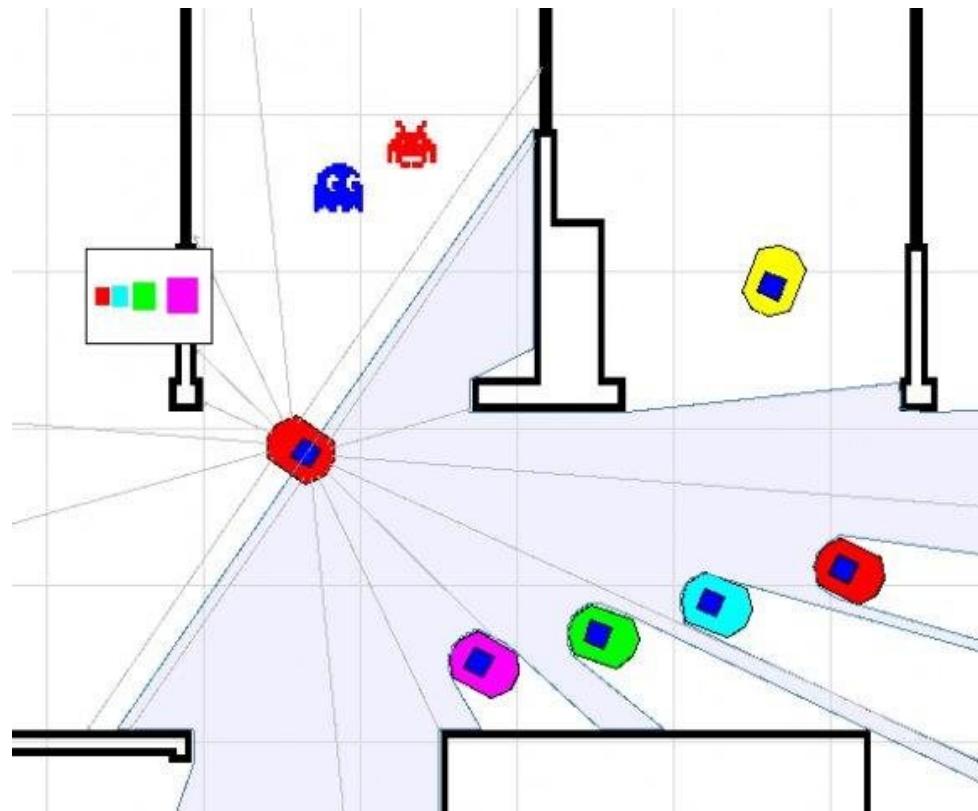


Figura 5 – Tela de simulação do Player/Stage

Este simulador é o que mais se aproxima do simulador desenvolvido, uma vez que ele utiliza um protocolo em rede para a comunicação. A diferença com este simulador vem do fato do *Player/Stage* não estar interessado na fidelidade física da simulação, portanto efetuar simulações próximas do mundo real pode ser complicado. No simulador desenvolvido a tarefa é mais simples, uma vez que este é um dos pilares utilizados na construção do simulador.

A plataforma *Player/Stage* pode ser utilizada para simulação tridimensional e com implementação física mais realista, através do uso do *Gazebo* em conjunto com *Player*. Isso contudo apresenta as mesmas limitações e desvantagens já descritas na Seção 2.1.

### 3 SIMULADOR

O simulador desenvolvido permite que um robô móvel possa ser controlado à partir de qualquer linguagem de programação com suporte à FFI da linguagem C [6].

Para que fosse possível o controle do simulador desta forma, o desenvolvimento do software foi dividido em duas partes distintas, o simulador e a biblioteca de comunicação. O simulador é responsável por efetuar as simulações físicas e estruturais do robô, bem como a cinemática do mesmo. Já a biblioteca de comunicação é responsável por permitir a interação com o simulador para a aquisição e configuração de parâmetros como velocidade, erros e câmera.

Este capítulo está dividido da seguinte maneira:

- A Seção 3.1 descreve o software de simulação, apresentando toda a sua modelagem e as funções desenvolvidas.
- A Seção 3.2 descreve o desenvolvimento e as técnicas utilizadas na biblioteca de comunicação.

#### 3.1 Software de simulação

O software de simulação foi construído utilizando a plataforma Unity 3D [7]. O objetivo deste software é fornecer uma simulação mais realística possível de um robô móvel, portanto, foram definidos 3 requisitos básicos para sua concepção.

1. Ambiente tridimensional com capacidade de simular sensores tridimensionais
2. Fidelidade aos parâmetros construtivos do robô na modelagem cinemática
3. Fidelidade na simulação física do ambiente onde o robô está inserido

##### 3.1.1 Modelagem 3D

O primeiro passo na construção do simulador foi o desenvolvimento do modelo tridimensional do robô móvel. O robô escolhido como base para a modelagem foi o robô diferencial *Pioneer P3-DX*. A escolha deste robô foi feita com base na sua grande difusão no meio acadêmico, bem como a facilidade de se encontrar materiais à respeito deste robô. O modelo tridimensional construído foi simplificado de forma que as partes mecânicas do robô que influenciam diretamente em sua cinemática ficassem mais evidentes. As Figuras 6 e 7 apresentam respectivamente o robô *Pioneer* real e o mesmo robô simulado no software desenvolvido.



Figura 6 – Robô real: *Pioneer P3-DX*

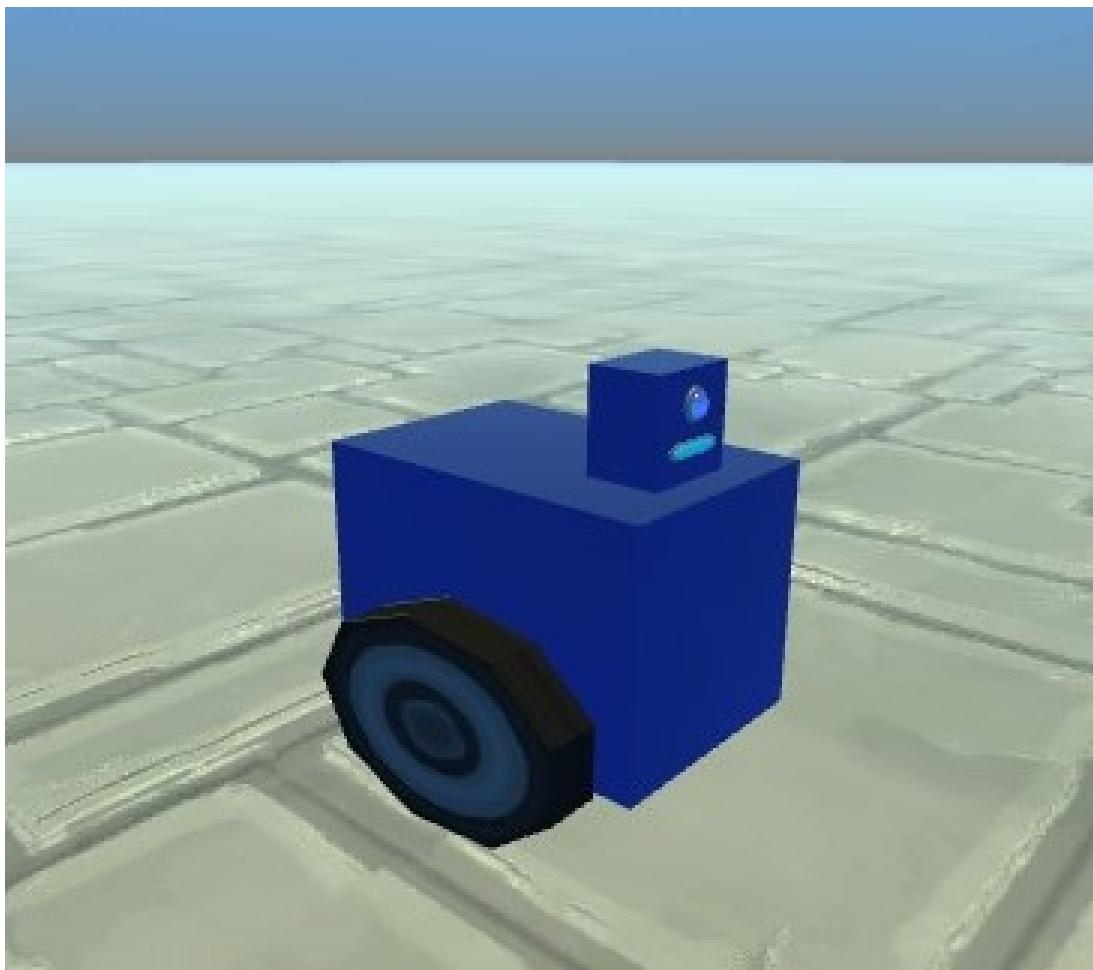


Figura 7 – Modelo 3D do robô simulado

O robô virtual foi modelado com as dimensões  $1.0m \times 0.7m \times 0.4m$ , ou seja,  $1.0m$  de comprimento,  $0.7m$  de largura e  $0.4m$  de altura. O comprimento foi escolhido por questões

de simplificação, de forma que uma das dimensões do robô servisse como referência para normalização das outras dimensões.

$$\begin{bmatrix} \text{Comprimento} \\ \text{Largura} \\ \text{Altura} \end{bmatrix} = \begin{bmatrix} 1.0m \\ 0.7m \\ 0.4m \end{bmatrix} \quad (3.1)$$

### 3.1.2 Cinemática

Devido à modelagem tridimensional do robô, pode-se verificar que este é um robô diferencial, portanto é possível modelar suas equações cinemáticas como tal. Em um robô diferencial, seu movimento é dado pela diferença de velocidade de suas duas rodas, sendo estas, montadas em um único eixo comum, onde cada roda é controlada por um motor individual.

Para encontrar as equações da cinemática [8] deste robô, inicialmente é definido um sistema de coordenadas. Por conveniência, o sistema de coordenadas escolhido foi o cartesiano, desta forma, foi definido que a frente do robô é o sentido positivo do eixo X, a lateral esquerda do robô é o sentido positivo do eixo Y, e a parte de cima do robô indica o sentido positivo do eixo Z, portanto, pela regra de Flemming, o sentido positivo de rotação do robô é um giro em torno do eixo Z, partindo do sentido negativo do eixo Y, passando pelo sentido negativo do eixo X e chegando ao sentido positivo do eixo Y. Considerando que a distância entre as rodas do robô seja  $2L$  e que o raio das duas rodas seja igual à  $r$ , pode-se representar o robô com suas dimensões e sistema de coordenadas como na Figura 8.

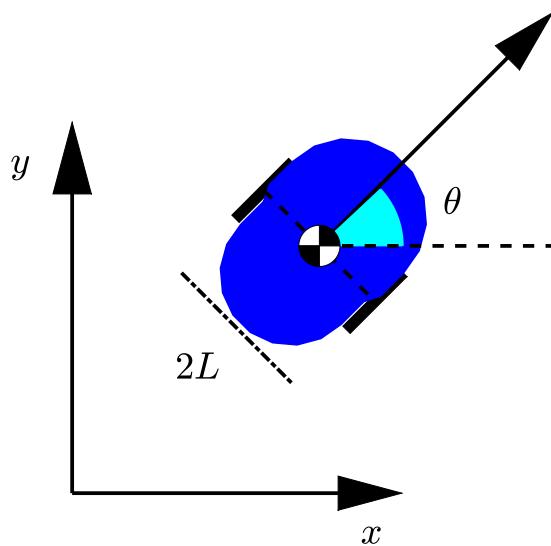


Figura 8 – Robô em um sistema de coordenadas cartesianas

Sendo as velocidades angulares das rodas do robô dadas por  $\dot{\phi}_l$  e  $\dot{\phi}_r$ , a velocidade linear das rodas será dada por:

$$V_l = \dot{\phi}_l r \quad (3.2)$$

$$V_r = \dot{\phi}_r r \quad (3.3)$$

A cinemática generaliza o robô à um único ponto médio que pode se mover no espaço, portanto, cada roda contribui com metade da velocidade linear do robô, ou seja:

$$\begin{aligned} V &= \frac{V_l}{2} + \frac{V_r}{2} \\ V &= \frac{\dot{\phi}_l r}{2} + \frac{\dot{\phi}_r r}{2} \\ V &= \frac{r}{2}(\dot{\phi}_l + \dot{\phi}_r) \end{aligned} \quad (3.4)$$

Considerando que o robô movimenta uma roda por vez segundo a regra de Flemming, sabendo que  $L$  é a distância de uma roda até o ponto médio entre as duas rodas, a contribuição de cada roda no movimento rotacional pode ser calculada como:

$$\begin{aligned} \omega_r &= \frac{\dot{\phi}_r r}{L} \\ \omega_l &= \frac{-\dot{\phi}_l r}{L} \end{aligned} \quad (3.5)$$

Portanto, a velocidade angular do robô será dada por:

$$\begin{aligned} \omega &= \frac{\omega_r}{2} + \frac{\omega_l}{2} \\ \omega &= \frac{\frac{\dot{\phi}_r r}{L} + \frac{-\dot{\phi}_l r}{L}}{2} \\ \omega &= \frac{r}{2L}(\dot{\phi}_r - \dot{\phi}_l) \end{aligned} \quad (3.6)$$

Tem-se que  $\dot{\phi}_l = \omega_l$  e  $\dot{\phi}_e = \omega_e$ , logo:

$$\begin{aligned} V &= \frac{r}{2}(\omega_l + \omega_r) \\ \omega &= \frac{r}{2L}(\omega_r - \omega_l) \end{aligned} \quad (3.7)$$

É possível decompor as velocidades do robô sobre o eixos cartesianos como:

$$\begin{bmatrix} \dot{X}_I \\ \dot{Y}_I \\ \dot{\theta}_I \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \quad (3.8)$$

Analizando a Figura 9, conclui-se que as componentes de velocidade  $v_x$  e  $v_y$  são dadas por:

$$\begin{aligned} v_x &= V \cos(\theta) \\ v_y &= V \sin(\theta) \end{aligned} \quad (3.9)$$

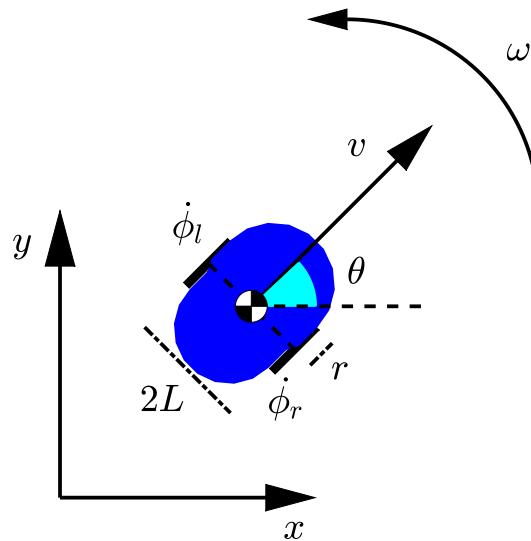


Figura 9 – Relações de velocidades cartesianas no robô

Sendo assim, é possível se escrever as equações das velocidades do robô como:

$$\begin{bmatrix} \dot{X}_I \\ \dot{Y}_I \\ \dot{\theta}_I \end{bmatrix} = \begin{bmatrix} \cos(\theta_I) & 0 \\ \sin(\theta_I) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \quad (3.10)$$

Se o robô se move com uma velocidade linear  $V$  e uma velocidade angular  $\omega$ , é possível calcular o deslocamento do robô integrando suas velocidades no tempo, ou seja:

$$\begin{aligned} \Delta S &= \int_0^t V dt \\ \Delta \omega &= \int_0^t \omega dt \end{aligned} \quad (3.11)$$

Considerando as velocidade constantes em um intervalo tempo  $\Delta t$ , reescreve-se a Equação 3.11 como:

$$\begin{aligned}\Delta S &= V\Delta t \\ \Delta\omega &= \omega\Delta t\end{aligned}\quad (3.12)$$

Da mesma forma que pode-se decompor as velocidades, o deslocamento do robô também pode ser decomposto como:

$$\begin{aligned}\Delta x &= \Delta S \cos(\Theta) \\ \Delta y &= \Delta S \sin(\Theta)\end{aligned}\quad (3.13)$$

Analizando a Equação 3.13, é possível perceber que existe uma dependência do parâmetro  $\Theta$ . Este parâmetro indica o ângulo do robô no *frame* inicial, portanto ele deve ser dado pelo ângulo do robô antes do movimento bem como a variação do ângulo com o movimento. Sabendo que cada roda contribui com metade da velocidade do robô, pode-se escrever o valor de  $\Theta$  como:

$$\Theta = \theta_{k-1} + \frac{\Delta\theta}{2} \quad (3.14)$$

Portanto, o deslocamento do robô no plano cartesiano é dado por:

$$\begin{aligned}\Delta x &= \Delta S \cos\left(\theta_{k-1} + \frac{\Delta\theta}{2}\right) \\ \Delta y &= \Delta S \sin\left(\theta_{k-1} + \frac{\Delta\theta}{2}\right)\end{aligned}\quad (3.15)$$

Logo, a posição do robô no plano cartesiano será dada por:

$$\begin{aligned}x_k &= x_{k-1} + \Delta S \cos\left(\theta_{k-1} + \frac{\Delta\theta}{2}\right) \\ y_k &= y_{k-1} + \Delta S \sin\left(\theta_{k-1} + \frac{\Delta\theta}{2}\right) \\ \theta_k &= \theta_{k-1} + \Delta\theta\end{aligned}\quad (3.16)$$

Expandindo os termos da Equação 3.16 e reescrevendo-a em formato matricial, partindo de um ponto inicial, a posição do robô pode ser calculada por:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} \left(\frac{V_r+V_l}{2}\Delta t\right) \cos\left(\theta_{k-1} + \frac{V_r-V_l}{2L}\Delta t\right) \\ \left(\frac{V_r+V_l}{2}\Delta t\right) \sin\left(\theta_{k-1} + \frac{V_r-V_l}{2L}\Delta t\right) \\ \frac{V_r-V_l}{2L}\Delta t \end{bmatrix} \quad (3.17)$$

### 3.1.3 Sensores

Para que o simulador fosse o mais próximo possível de um robô real, foram adicionados à ele diversos sensores que permitem que o mesmo perceba o mundo à sua volta. Assim como no mundo real, estes sensores apresentam erros e portanto suas medidas também apresentam erros. Para o robô modelado, foram adicionados 3 sensores distintos, cada um com um propósito: um GPS (*Global Positioning System*) acoplado à uma bússola, um *LiDAR* (*Light Detection and Ranging*) e uma câmera RGB.

#### 3.1.3.1 GPS

No mundo real, o sistema de posicionamento global (GPS) [9], é capaz de informar com precisão de alguns metros à centímetros, a posição global de um receptor, desde que este esteja no alcance de no mínimo 3 satélites. No simulador, a posição global do receptor (neste caso o robô), é inerente à simulação, portanto está não precisa ser calculada.

O sistema de GPS real, assim como qualquer tipo de sensor, apresenta erros que alteram o valor real das medidas. No simulador, como o posicionamento do robô é dado pela própria referência de posição dos objetos do simulador, a medida de posição retornada pelo sistema de posicionamento do simulador é exatamente a posição do robô, e portanto não apresenta erros. Como o objetivo do simulador é que o robô seja o mais real possível, incluindo os erros, foi adicionado ao sistema de GPS do simulador um erro que tenta se aproximar do sistema no mundo real.

O erro adicionado é um erro gaussiano, com média na posição real do robô e um desvio padrão ajustável para as coordenadas  $X$  e  $Y$ . Sendo  $X_{real}$  e  $Y_{real}$  as coordenadas reais do robô no simulador, é possível escrever as equações da posição retornada pelo GPS como:

$$\begin{aligned} X_r &= X_{real} + \sigma_x \cdot randn() \\ Y_r &= Y_{real} + \sigma_y \cdot randn(), \end{aligned} \quad (3.18)$$

onde  $randn()$  é uma função que retorna um número aleatório com distribuição normal, com média em 0 e desvio padrão 1.

Além do GPS, o robô também possuí uma bússola que indica sua orientação relativa ao norte magnético do ambiente virtual onde está inserido. Por conveniência, o norte magnético do mundo simulado foi definido nas coordenadas:

$$\begin{aligned} X_{polo} &= +\infty \\ Y_{polo} &= 0 \\ Z_{polo} &= 0 \end{aligned} \quad (3.19)$$

Esta escolha de coordenadas se deve ao fato de que o mundo simulado é plano, portanto o seu norte magnético deve estar ao infinito de uma das direções do plano cartesiano. Assim como o sensor GPS, a bússola real também contém um erro associado que não está presente no sensor simulado, portanto, para que o robô simulado seja condizente com a realidade, um erro gaussiano foi adicionado à bússola simulada. Desta forma, sendo a orientação real do robô dada por  $\theta_{real}$ , a orientação retornada pela bússola será dada por:

$$\theta_r = \theta_{real} + \sigma_\theta \cdot randn() \quad (3.20)$$

Unindo os dois sensores definidos acima e colocando as equações na forma matricial, obtém-se a seguinte equação de posicionamento global para o robô:

$$\begin{bmatrix} X_r \\ Y_r \\ \theta_r \end{bmatrix} = \begin{bmatrix} X_{real} \\ Y_{real} \\ \theta_{real} \end{bmatrix} + \begin{bmatrix} \sigma_x randn() \\ \sigma_y randn() \\ \sigma_\theta randn() \end{bmatrix} \quad (3.21)$$

Na Figura 10 pode-se ver as leituras dos sensores GPS e bússola do robô virtual para um percurso circular de raio  $R = 3m$ , considerando os desvios padrão do sistema como:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_\theta \end{bmatrix} = \begin{bmatrix} 0.3m \\ 0.3m \\ 2^\circ \end{bmatrix} \quad (3.22)$$

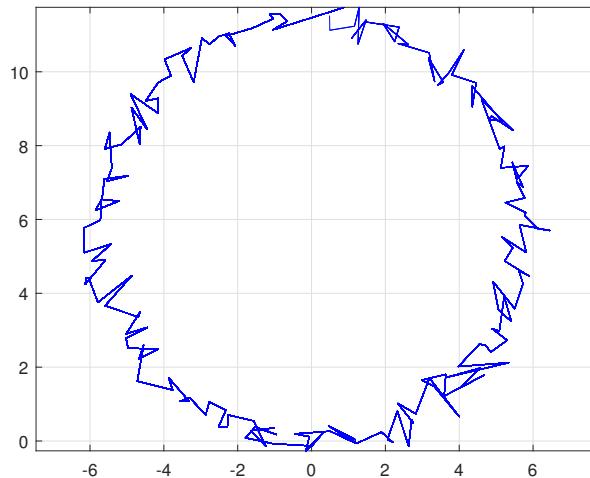


Figura 10 – Leituras do GPS para um percurso circular

### 3.1.3.2 Lidar

O *Lidar* (*Light Detection and Ranging*) [10] é uma tecnologia utilizada para medir as distância até objetos utilizando propriedades da luz refletida. O método usualmente utilizado é o de laser pulsado.

Para se medir a distância até um objeto, é medida a diferença de tempo entre a emissão de um pulso luminoso e a detecção do sinal refletido. Em um robô móvel real, o sensor lidar é montado na base do robô de forma que este obtenha as distâncias em um plano horizontal, geralmente à frente, do robô. As leituras são feitas como nos feixes exibidos na Figura 11.

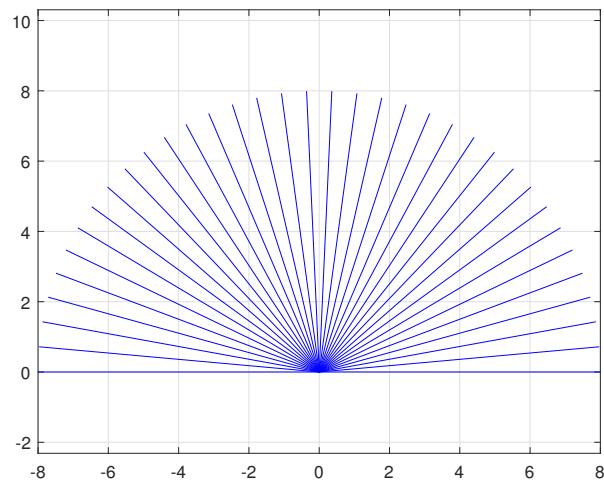


Figura 11 – Feixes da leituras do lidar

O desenvolvimento do lidar virtual para o simulador foi feito criando inicialmente um modelo 3D para o mesmo, como pode ser visto na Figura 12.



Figura 12 – Modelo 3D do *Lidar*

O *Lidar* virtual foi adicionado acima da base móvel do robô, com seu campo de visão direcionado para a frente do robô, de forma que o campo de visão seja uma semi-circunferência indo de  $-90^\circ$  até  $90^\circ$ , sendo que o ângulo  $0^\circ$  está exatamente na frente do robô. O sensor criado tem uma resolução de  $5^\circ$ , ou seja, fornece um valor de distância a cada  $5^\circ$ , desta forma, seu retorno é uma lista de 36 distâncias, uma para cada  $5^\circ$  do range de leituras do sensor. Como todos os sensores de distância, o *Lidar* também possui um limite de distância válido para as leituras. Acima deste limite a leitura não é mais confiável, pois atingiu-se o limite de saturação do sensor. No sensor virtual não existe nenhuma limitação para a distância medida, uma vez o mundo simulado é infinito e as medidas são feitas de forma matemática. Porém, para garantir fidelidade com o sensor real, foi adicionado um limite de distância de  $8m$  nas leituras do sensor. Em um sensor *Lidar* real também existem erros associados às medidas, que podem ser definidos como desvios gaussianos de média 0. Estes erros também foram adicionados ao sensor virtual, para que suas medidas fossem forçadamente não ideais.

Como o erro adicionado é do tipo gaussiano, as leituras do lidar podem ser definidas como a Equação 3.23.

$$\rho = \rho_{real} + \sigma_\rho \cdot randn() \quad (3.23)$$

onde  $\rho_{real}$  é leitura correta feita pelo lidar virtual e  $\sigma_\rho$  é o desvio padrão do sensor.

A Figura 13 apresenta as leituras do lidar quando o robô é colocado à distância de 3 metros de uma parede plana, com sua frente alinhada com a quina da parede e considerando um desvio padrão de  $\sigma_\rho = 0.1m$  para a leitura de distância.

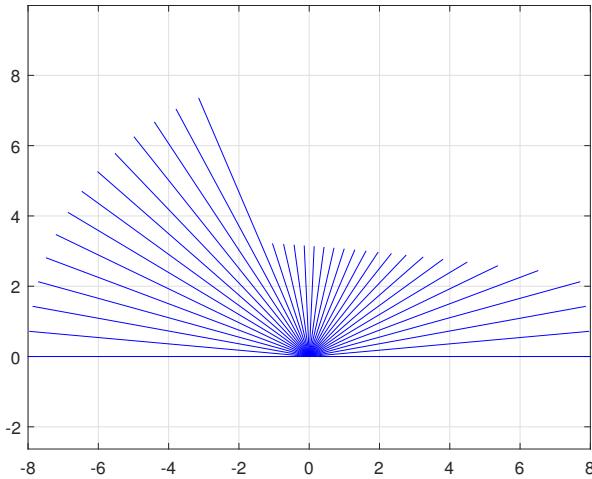


Figura 13 – Leituras do lidar em frente à uma parede

### 3.1.3.3 Câmera RGB

As câmeras RGB são dispositivos que utilizam sensores CMOS [11] padrão. Estes sensores são formados por uma matriz de transdutores fotossensíveis capazes de converter a energia luminosa de um ponto da imagem em uma carga elétrica que em seguida é convertida para valores numéricos e passam a representar uma imagem digital. Em câmeras RGB, para cada ponto da matriz, chamado de pixel, existem 3 sensores, cada um com um filtro para um cor. Como a carga elétrica dos sensores são convertidos em um valor numérico digital, os valores resultantes dependem do da quantidade de bits e da resolução do conversor, sendo que geralmente os valores para cada cor em uma câmera RGB são números de 8 bits, ou seja, podem variar entre 0 e 255.

Desta forma, com sensores para 3 cores básicas, vermelho (**Red**), verde (**Green**) e azul (**Blue**), e com conversores de 8 bits, é possível obter imagens com até aproximadamente 16 milhões de cores:

$$256 \times 256 \times 256 = 16777216$$

A utilização de câmeras em robôs é algo que tem se tornado cada vez mais comum, uma vez que esta é um sensor que permite que o robô perceba o mundo da mesma forma que os humanos, ou seja, à cores e em mais de uma dimensão. Utilizando uma câmera em um robô é possível criar aplicações simples como detectar um objeto de uma determinada cor até aplicações complexas como a pilotagem autônoma com detecção de obstáculos e pessoas.

Neste simulador, a câmera adicionada ao robô possui uma resolução de  $320 \times 240$ , ou seja, 320 pixels de largura e 240 de altura. A câmera virtual foi adicionada logo acima do

sensor *Lidar*. Desta forma a câmera também oferece uma visão do que está à frente do robô. A câmera possui uma abertura de 60° na horizontal e 47° na vertical, isto garante uma boa visão da frente do robô, sem que partes do robô estejam visíveis na imagem.

Esta câmera trabalha à taxa de 30fps, ou seja, uma nova imagem está disponível à aproximadamente cada 33 milisegundos. A Figura 14 apresenta uma imagem capturada pela câmera virtual adicionada ao robô.



Figura 14 – Imagem do ambiente simulado, capturada com a câmera do robô

### 3.1.4 Física do Ambiente Simulado

Como o simulador deve representar de forma fiel o mundo real, sua descrição física deve ser bem consistente. A Unity 3D contém em seu portfólio uma vasta gama de funções específicas para a simulação física, desde mecânicas simples como a gravidade até as mais complexas como a colisão de objetos tridimensionais.

Neste simulador foram adotadas 2 mecânicas físicas à serem simuladas, a gravitação e a dinâmica de corpos rígidos. Cada uma destas mecânicas é descrita abaixo:

#### 3.1.4.1 Gravitação

O mundo criado para este simulador é plano, e portanto, deve possuir uma aceleração da gravidade que mantenha os objetos próximos ao chão. Para que fosse possível definir a aceleração da gravidade, inicialmente foi construído o solo da simulação, sendo que este pode ser plano ou conter áreas em relevo, de forma à dificultar a locomoção do robô.

Como o mundo da simulação é um plano muito maior do que o robô, é possível considerar este plano como parte de um corpo rígido esférico, portanto, ele pode ser interpretado como um planeta e assim aplicar as leis da gravitação universal.



Figura 15 – Representação da lei da gravitação universal para dois corpos

Pela Lei da Gravitação Universal de Newton [12] tem-se que “Dois corpos atraem-se com força proporcional às suas massas e inversamente proporcional ao quadrado da distância que separa seus centros de gravidade”, portanto, é possível calcular a força de atração gravitacional entre o robô e o solo como:

$$F = G \frac{M_t \times m_r}{d^2}, \quad (3.24)$$

onde  $G$  é a Constante de gravitação universal, e possui o valor:

$$G \cong 6,67 \times 10^{-11} \frac{N \cdot m^2}{Kg^2}. \quad (3.25)$$

Como o simulador deve representar bem o comportamento do robô no planeta Terra, foi definido que os parâmetros do mundo simulado seriam os mesmos da Terra, ou seja, o raio e a massa da terra seriam considerados no cálculo da força de aceleração da gravidade.

O raio médio do planeta Terra é de aproximadamente  $R_t = 6371km = 6371000m$  e sua massa é de aproximadamente  $M_t = 5,972 \times 10^{24}kg$ , portanto, serão utilizados estes valores para o cálculo das forças.

De acordo com a Equação 3.1, a altura do robô é de  $h = 0.4m$ , portanto, a distância do chão até o seu centro de gravidade seria de  $r_r = 0.2m$ . Porém, para aumentar a estabilidade do robô no simulador, seu centro de gravidade foi alterado de forma à estar 10cm mais próximo do chão. Deste modo, o novo centro de gravidade tem a distância de  $r_r = 0.1m$  até o solo.

A massa do robô foi definida com base na massa do robô real (Pioneer P3-DX), portanto,  $m_r = 25kg$ .

Aplicando os dados acima na Equação 3.24, a força de atração gravitacional do solo simulado e o robô será de aproximadamente:

$$F \cong 245.341N \quad (3.26)$$

Portanto, de acordo com a segunda Lei de Newton (Princípio Fundamental da Dinâmica):

$$F = m \times a, \quad (3.27)$$

onde ' $F$ ' é uma força, ' $m$ ' é uma massa e ' $a$ ' é uma aceleração.

Aplicando a Equação 3.27 nos resultados obtidos para o robô obtém-se:

$$a = \frac{F}{m} \cong \frac{245.341N}{25Kg} \cong 9.81365m/s^2 \quad (3.28)$$

Sendo assim, para a simulação, é aplicada ao robô uma força, e consequentemente uma aceleração de:

$$\begin{aligned} F_g &= -245.341N \\ g &= -9.81365m/s^2 \end{aligned} \quad (3.29)$$

A aceleração é negativa já que é desejado que a força de atração gravitacional mantenha o robô no chão.

### 3.1.4.2 Corpo rígido

Um corpo rígido [13] é definido como um conjunto finito de partículas, cada uma com uma massa e posição, sendo que a distância entre cada uma destas partículas é constante no tempo. Sendo assim, foi definido que cada objeto na simulação seria um corpo rígido. Considerando que dois corpos não podem ocupar o mesmo espaço, pela Segunda Lei de Newton, o Princípio da Ação e Reação, tem-se que sempre haverá uma força com módulo e direção iguais à uma força atuando em um corpo, apenas com o sentido inverso. Sendo assim, será partido deste pressuposto para se definir as colisões entre os objetos no mundo simulado.

Pela Equação 3.29, verifica-se que uma força irá atuar sobre o robô fazendo com que o mesmo tenda a se aproximar cada vez mais do solo da simulação, portanto, se o robô e o solo são corpos rígidos, quando o robô estiver em contato com o solo, uma força será aplicada no mesmo de forma a mantê-lo no chão, sem que ocupem o mesmo espaço, esta força será dada por:

$$F_{-g} = 245.341N \quad (3.30)$$

Além do solo, o mundo simulado também apresenta outros corpos rígidos como árvores e construções civis, portanto, o robô também irá eventualmente colidir com estes objetos e o princípio de ação e reação deve continuar válido.

Durante uma colisão os módulos das forças trocadas pelos agentes da colisão são muito superiores aos módulos das forças externas, portanto pode-se considerar o sistema

mecanicamente isolado, ou seja, apenas as forças trocadas entre os agentes da colisão são consideradas.

Pela conservação do momento linear tem-se que, para dois objetos colidindo:

$$M_1 V_1 + M_2 V_2 = M_1 V'_1 + M_2 V'_2 \quad (3.31)$$

Onde  $M_1$  e  $M_2$  são as massas dos dois objetos colidindo,  $V_1$  e  $V_2$  são suas velocidades antes da colisão e  $V'_1$  e  $V'_2$  são suas velocidades após a colisão.

No simulador, para um primeiro momento considera-se que apenas o robô estará em movimento, portanto pode-se assumir que:

$$\begin{aligned} m_r v_r + M_2 0 &= m_r v'_r + M_2 V'_2 \\ m_r v_r &= m_r v'_r + M_2 V'_2 \end{aligned} \quad (3.32)$$

Ou seja, após uma colisão do robô com algum objeto, dependendo da velocidade do robô,  $v_r$ , este objeto também pode entrar em movimento.

No simulador, foi definido que nenhum objeto pode sofrer deformação devido às colisões, portanto, apenas suas velocidades podem ser afetadas pela dinâmica entre os corpos rígidos.

### 3.1.5 Modos de simulação

Além de simular o robô virtual, o simulador também deve permitir simular estruturas para interagir com o robô, e, para tal, o simulador possui dois modos de simulação, um modo realístico e um modo de estruturas dinâmicas, cada um sendo mais indicado em determinadas situações.

A troca entre os dois modos de simulação é feito através de botões indicados na interface do simulador.

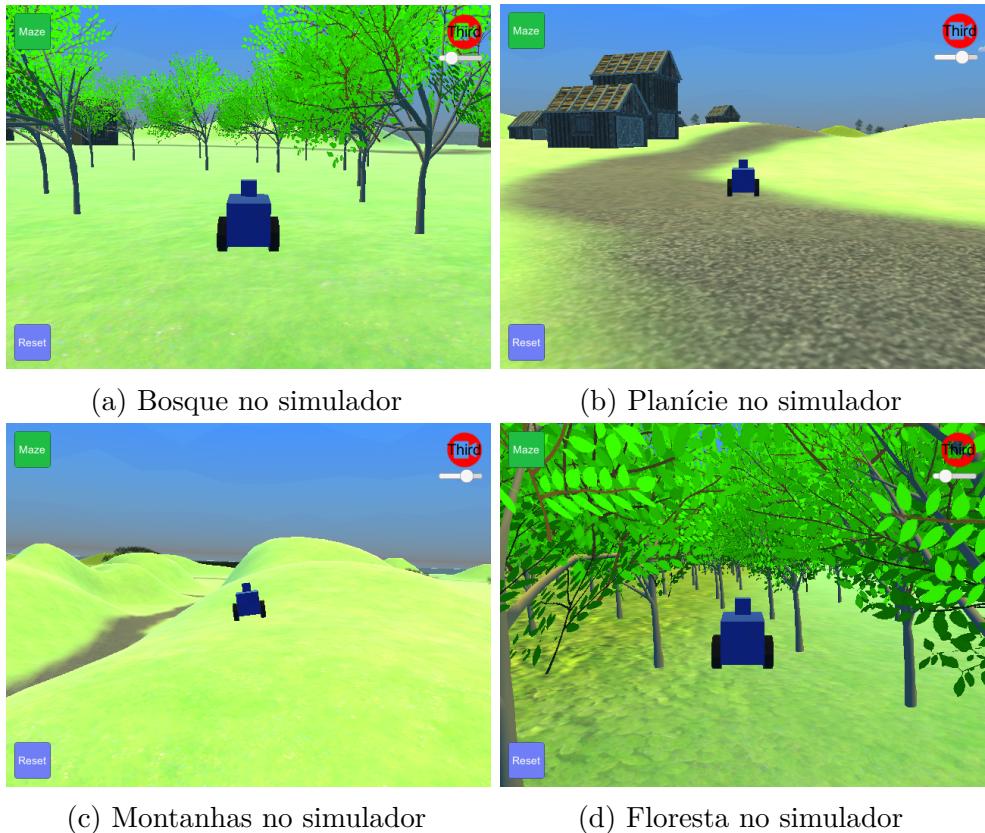
#### 3.1.5.1 Modo realístico

No modo realístico, o robô é inserido em um vasto mapa com diversos ambientes. Este é o modo padrão do simulador.

Neste modo, o robô estará inicialmente em uma estrada e terá a opção de explorar diversos ambientes diferentes, como florestas, planícies e montanhas. O robô poderá encontrar construções civis e objetos espalhados pelo cenário.

A estrada encontrada no modo realístico forma um caminho fechado, portanto independente do lado que o robô seguir na estrada nunca chegará em algum ponto sem estrada.

Além da estrada, o robô também poderá encontrar gramíneas, flores e outras marcações na superfície do solo, isto permite que durante a simulação o robô seja exposto à elementos comuns em um ambiente real.



### 3.1.5.2 Modo de estruturas dinâmicas (Maze)

No modo Maze do simulador o robô estará inicialmente em uma planície vazia, porém, elementos podem ser adicionados dinamicamente à esta planície.

O simulador suporta a adição de elementos ao cenário utilizando arquivos como entrada de dados. Dois formatos de arquivo são suportados, arquivos de texto e imagens. No modo Maze o simulador exibirá um botão para a abertura de um arquivo, de forma que cada tipo de arquivo será carregado de forma diferente.

#### 3.1.5.2.1 Arquivos de texto

No caso de arquivos de texto, cada linha do arquivo será interpretada como um objeto à ser adicionado à cenário. Cada linha deve possuir várias componentes separados por espaço, onde cada componente indica uma informações sobre o objeto, como pode ser visto na tabela abaixo:

Tipo	$X_1$	$Y_1$	$X_2$	$Y_2$
------	-------	-------	-------	-------

Tabela 1 – Estrutura de um elemento no arquivo de texto

O 'Tipo' de cada elemento é representado por uma única letra, que é a inicial do nome do elemento em inglês. As opções são:

- L - Linha
- W - Parede
- R - Robô
- B - Bola
- C - Cubo

Alguns dos elementos descritos acima não precisam dos parâmetros  $X_2$  e  $Y_2$ , pois para tais elementos estes parâmetros não são necessários, é o caso dos elementos:

- R - Robô
- B - Bola
- C - Cubo

Esta forma de criação de elementos permite que rapidamente seja possível criar estruturas simples onde o robô pode ser colocado. Por exemplo, deseja-se criar um mapa como o da Figura 17, com o robô posicionado no centro da estrutura, de forma a estudar técnicas de localização.

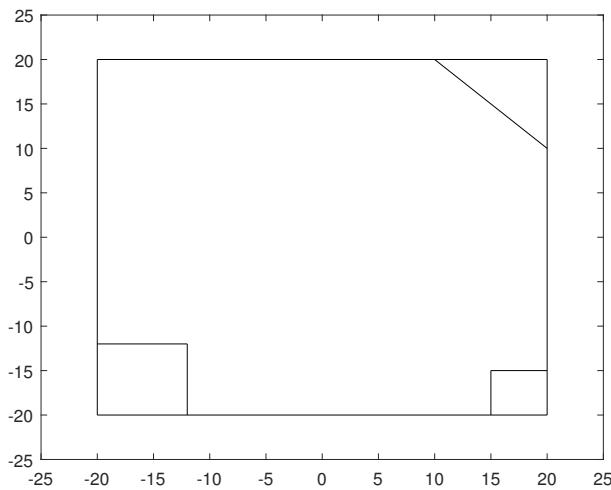


Figura 17 – Exemplo de mapa simples para o robô

Para uma estrutura como esta, o seguinte arquivo de texto seria criado:

```

W -20 -20 20 -20
W -20 -20 -20 20
W -20 20 20 20
W 20 -20 20 20
W -20 -12 -12 -12
W -12 -20 -12 -12
W 15 -20 15 -15
W 15 -15 20 -15
W 20 10 10 20
R 0 0

```

Mapa simples para localização

Este arquivo de texto irá gerar um mapa no simulador da seguinte forma:

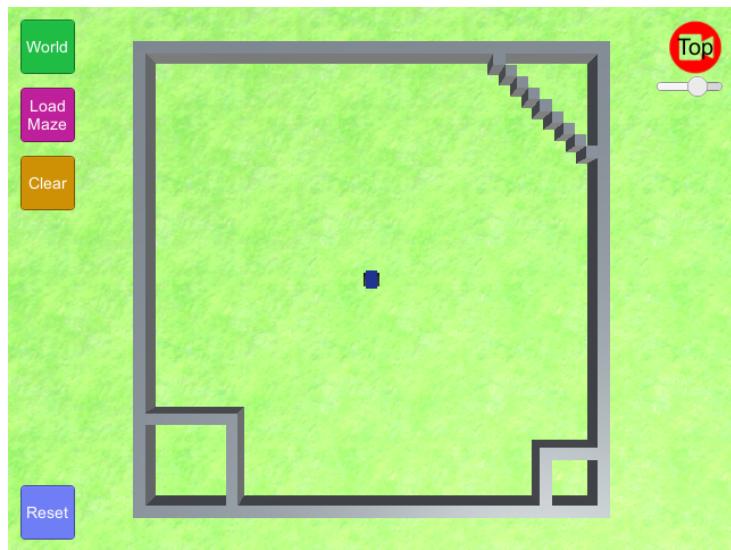


Figura 18 – Exemplo de mapa simples para o robô

### 3.1.5.2.2 Arquivos de imagens

Supõe-se que nem sempre os mapas serão simples, portanto, a adição de elementos no cenário através de imagens se torna uma opção mais viável.

Ao carregar um arquivo de imagem, o simulador irá exibir uma interface onde é possível selecionar o tamanho real do mapa na imagem, isto permite que as imagens não tenham que corresponder ao tamanho real do mapa.

Diferente dos arquivos de texto, as imagens tem os diversos elementos identificados por cores. Como as imagens não possuem necessariamente o tamanho real do mapa, é preciso agrupar regiões semelhantes na imagem para que estas se resumam à um único

ponto ou linha, para isto, é utilizado o algoritmo Flood fill de forma à segmentar a imagem em diversos elementos. Quando os elementos estão segmentados, é aplicado à imagem um operador morfológico de afinamento, assim qualquer linha ou parede que contenha mais um pixel de largura é reduzida, de forma que apenas linhas e paredes de um pixel estejam presentes na imagem resultante. Este processo pode ser visto na Figura 19.

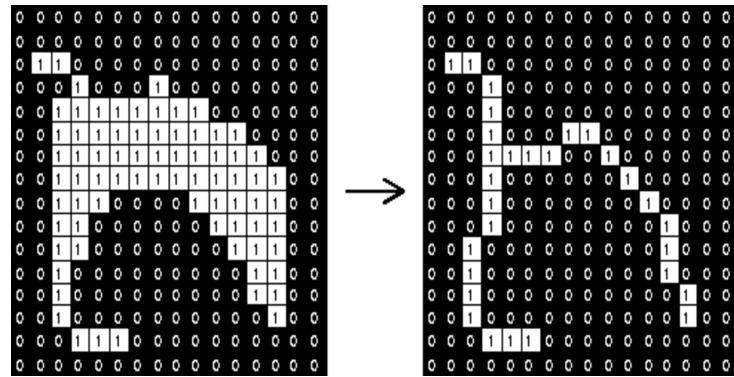


Figura 19 – Exemplo de afinamento de imagens

Cada tipo de elemento é representado por uma cor, sendo que as opções de cores são:

- Amarelo - Linha
- Preto - Parede
- Vermelho - Robô
- Azul - Bola
- Verde - Cubo

Utilizando este modo de adição de elementos, o mapa da Figura 17 pode ser representado como:

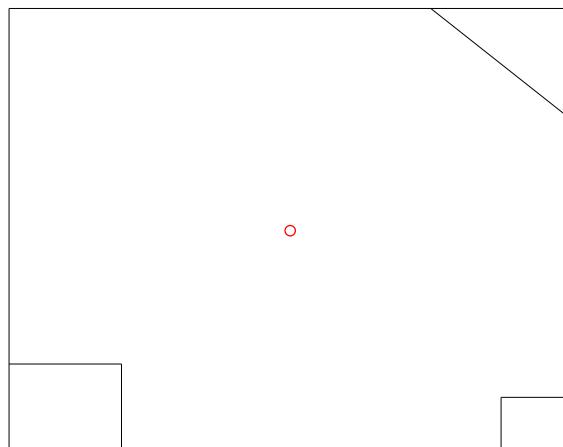


Figura 20 – Exemplo de mapa simples para o robô utilizando um arquivo de imagem

Para este mapa a representação por arquivos de texto é mais simples de ser feita, o que não justifica o método de adição por imagens, porém, para casos mais complexos, como um labirinto, o método por arquivos de texto se torna inviável pois seria necessário declarar diversos segmentos, e neste caso, o método por imagens torna a criação do mapa bem mais simples, viabilizando criar mapas como o da Figura 21.

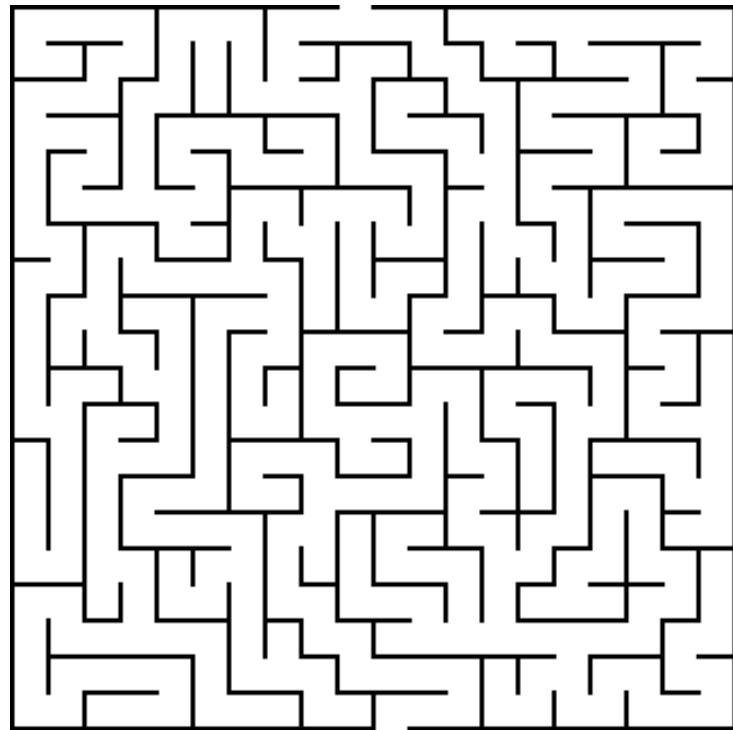


Figura 21 – Exemplo de mapa complexo para o robô utilizando um arquivo de imagem

Quando carregada no simulador, esta imagem irá gerar um mapa que pode ser visto parcialmente na Figura 22.

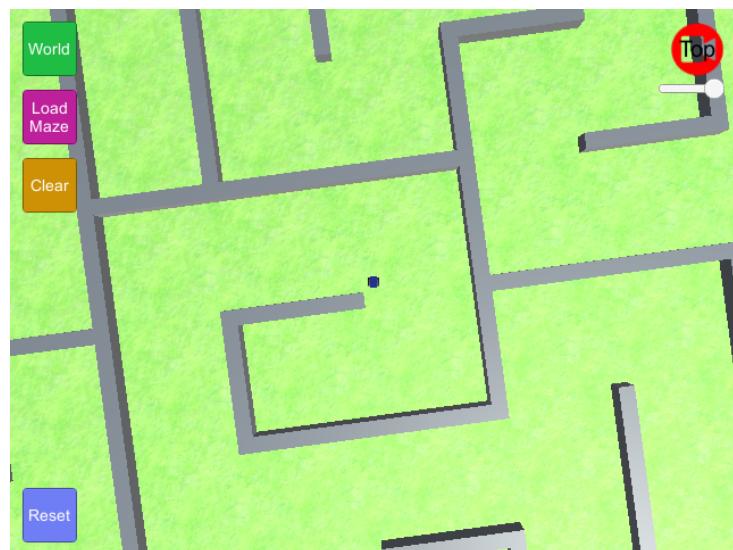


Figura 22 – Exemplo de mapa complexo no simulador

### 3.2 Biblioteca de comunicação

O simulador por si só é capaz apenas de criar o mundo onde o robô será inserido e simular o robô e todas as suas propriedades físicas, portanto, é necessário haver uma maneira de interagir com este robô e com o mundo de forma fácil para que sejam executadas as mais diversas tarefas com o robô.

Uma das alternativas para o controle do robô seria a criação de uma interface no simulador que permitisse controlar totalmente o robô, porém esta forma não seria prática pois a quantidade de controles seria infinita e a necessidade de se clicar em cada controle faria com que este processo fosse extremamente lento, portanto, uma forma de interagir com o robô por código seria a melhor opção.

Poderia ter sido criada uma interface com entrada de textos no simulador, assim poderiam ser escritos programas para interagir com o robô, porém, desta maneira o usuário do simulador deveria escrever ou reescrever o código em uma linguagem que ele poderia não saber, elevando a curva de aprendizado do simulador. Sendo assim, foi escolhida uma forma de controle mais complexo do ponto de vista do simulador, porém mais simples do ponto de vista do usuário.

O método escolhido foi a utilização de uma estratégia Cliente-Servidor, onde o servidor está integrado ao simulador e uma biblioteca que deve ser integrada aos programas do usuário faz o papel de cliente. Esta estratégia se mostra eficiente pois permite controlar o robô facilmente, ao mesmo tempo que permite que a biblioteca de comunicação seja implementada em um robô real, e desta forma, os códigos desenvolvidos para o simulador possam ser utilizados diretamente no robô real, sem alterações.

Para que a biblioteca funcione com basicamente qualquer linguagem de programação atual, foi utilizada a metodologia FFI (Foreign function interface), onde funções possuem uma definição de chamada em uma linguagem *Guest*, neste caso, a linguagem C, e a linguagem *Host* precisa ser compatível com a linguagem *Guest* de forma que a chamada seja possível. A biblioteca foi inteiramente escrita a linguagem C++, porém, suas funções foram declaras como funções na linguagem C, desta forma, qualquer linguagem que tenha suporte à FFI com a linguagem C pode invocar as funções. A linguagem C foi escolhida para a interface FFI pois praticamente todas as linguagens de programação possuem suporte à FFI com C, sendo alguns exemplos: C++, Python, C#, Java, Matlab, D, Go, Fortran e muitas outras.

#### 3.2.1 Servidor

Durante o desenvolvimento, dois protocolos de comunicação foram avaliados para serem implementados no simulador, sendo estas duas opções os protocolos TCP e UDP. As duas opções são protocolos de rede que enviam os dados em forma de pacotes, de forma

que o receptor tem a obrigação de agrupar os pacotes para obter os dados completos.

No protocolo TCP cada pacote tem uma identificação única, de forma que o receptor possa organizar os pacotes na ordem correta e requisitar o reenvio caso algum pacote não seja recebido, este protocolo garante a chegada de informações, porém é mais lento.

No protocolo UDP os pacotes não possuem uma identificação para garantir a ordem e nem um sistema de requisição dos pacotes perdidos, ou seja, quando um pacote é enviado, o emissor não se preocupa se o pacote foi recebido ou não pelo emissor, fazendo com que este protocolo seja mais rápido que o TCP.

O limite para um pacote de dados no protocolo UDP é de 65507 bytes e já que o mesmo não possui um sistema de reorganização e requisição de pacotes perdidos, este protocolo se torna inviável para a transmissão de grandes conjuntos de dados, como as imagens geradas pela câmera do robô, sendo assim, o protocolo escolhido foi o TCP.

O servidor TCP implementado no simulador utiliza uma porta fixa usualmente não utilizada e está atrelado ao endereço IP da máquina que está executando o simulador, isto permite que clientes possam utilizar o simulador mesmo que este esteja em outra máquina, bastando que os dois dispositivos estejam na mesma rede.

Uma vez que o simulador é executado, ele automaticamente inicia o servidor e começa à esperar conexões de clientes. Quando nenhum cliente está conectado ao simulador, ele permite que o robô seja controlado por teclado ou por um joystick conectado, desta forma é possível explorar o simulador mesmo sem criar nenhum programa específico para tal. Quando um cliente se conecta ao servidor, o mesmo para de esperar novos clientes e o controle manual do robô é desativado e todo o controle passa a ser proveniente dos dados enviados pelo cliente, que devem estar em um formato especificado pelo protocolo de dados. Quando não há mais nenhum cliente conectado, o controle manual é reabilitado e o servidor começa a esperar novos clientes.

O servidor permite apenas um cliente por vez para garantir que apenas um usuário esteja utilizando o robô por vez e desta forma, usuários diferentes não solicitem tarefas diferentes ao mesmo tempo para o robô.

O servidor foi criado de forma que este seja um agente passivo na comunicação, e os clientes sejam os agentes ativos, desta forma, o servidor apenas recebe as requisições dos clientes e envia uma resposta quando esta requisição tenha um dado de retorno.

### 3.2.2 Protocolo de dados

Para garantir que as informações trocadas entre os clientes e o servidor do simulador sejam interpretadas corretamente, foi criado um protocolo de dados que rege o formato de todos os pacotes trafegados na comunicação. Este protocolo define um formato de requisição feita pelo cliente e um formato de resposta enviada pelo servidor.

O primeiro especificador do protocolo de dados é o formato de entrada de dados no servidor, ou seja, como os dados são enviados da biblioteca de comunicação para o servidor. Todos os pacotes de enviados pela biblioteca de comunicação para o servidor seguem o seguinte padrão:

<i>Op</i>	<i>Name<sub>s</sub></i>	<i>Name</i>	<i>Value</i>	<i>Data</i>
-----------	-------------------------	-------------	--------------	-------------

Tabela 2 – Pacote de dados enviado ao servidor

A primeira informação enviada no pacote de dados é o campo "Op", ele representa o operador à ser utilizado. O operador possui apenas um byte de tamanho, podendo ter os valores abaixo:

- 0. Leitura
- 1. Escrita
- 2. Escrita de vetor de dados

O campo *Name<sub>s</sub>* do pacote de dados indica o tamanho do nome do atributo a ser lido/escrito. Este campo é um número inteiro de 32 bits, e portanto, possui 4 bytes de tamanho.

O campo *Name* é responsável por armazenar o atributo solicitado, onde cada byte representa uma letra do nome do atributo, portanto, este campo tem o tamanho *Name<sub>s</sub>*.

Os últimos campos do pacote de dados enviado ao servidor são os campos *Value* e *Data*. O campo *Value* é responsável por armazenar o valor desejado para o atributo *Name*, portanto, no caso de pacotes do tipo Leitura, este campo não precisa estar presente no pacote. Para pacotes do tipo Escrita, este campo é um número decimal de 32 bits, logo, ele ocupa 4 bytes no pacote de dados. Quando o tipo do pacote é Escrita de vetor de dados, este campo armazena o tamanho do vetor de dados em um número inteiro de 32 bits, ou seja, utiliza 4 bytes do pacote. Neste caso o campo *Data* também estará preenchido, onde seu tamanho será *Value* e ele armazenará uma lista de bytes representando o vetor de dados.

Quando o servidor recebe um pacote de dados como o apresentado na Tabela 2, este analisará o pacote e irá responder com um pacote de dados como o apresentado na Tabela 3.

<i>Type</i>	<i>Status</i>	<i>Value</i>	<i>Data<sub>s</sub></i>	<i>Data</i>
-------------	---------------	--------------	-------------------------	-------------

Tabela 3 – Pacote de dados enviado ao cliente

O campo *Type* indica o tipo de dados presente no pacote. Este campo possui apenas um byte, com seus valores podendo ser:

3. Valor único
4. Vetor de dados

Os pacote de Valor único contém apenas os campos *Type*, *Status* e *Value* da Tabele 3, enquanto que os pacotes do tipo Vetor de dados possuem também os campos *Data<sub>s</sub>* e *Data*.

O campo *Status* também possui apenas um byte e ele indica se a operação foi bem sucedida, sendo o valor 1 utilizado para indicar sucesso na operação e o valor 0 utilizado para indicar falha.

O campo *Value* indica o valor atual do atributo solicitado, sendo que se a requisição é do tipo escrita, o valor retornado é o mesmo valor enviado pelo cliente. Este campo é um número decimal de 32 bits, e portanto, utiliza 4 bytes do pacote de dados.

Alguns atributos retornam mais do que um único valor numérico, podendo retornar milhares de bytes adicionais, como é o caso do atributo de captura da câmera, e, para este tipo de atributo, os campos *Data<sub>s</sub>* e *Data* são utilizados.

O campo *Data<sub>s</sub>* representa a quantidade de bytes a serem enviados ao cliente, sendo portanto um número inteiro de 32 bits e ocupando 4 bytes no pacote.

O campo *Data* representa os bytes de dados enviados ao cliente. Este campo tem o tamanho *Data<sub>s</sub>* e os bytes enviados podem estar representar qualquer informação, o que será definido pelo atributo solicitado.

O protocolo de dados deve ser implementado em qualquer robô real onde deseja-se que o mesmo seja controlado da mesma forma que o simulador. Como a biblioteca de comunicação fica no cliente, os robôs devem implementar o servidor TCP e os pacotes enviados ao cliente do protocolo de dados.

### 3.2.3 Funções

Com o servido TCP e o protocolo de dados definido, eles foram utilizados para se criar uma tabela de funções que deve ser conhecida tanto pelo servidor no simulador quanto pela biblioteca de comunicação no cliente. Ao todo foram definidas 40 funções, como pode ser visto na Tabela 4.

Cliente		Servidor	Cliente		Servidor
Name	Op	Type	Name	Op	Type
Pose.X	1/0	0	Camera.Capture	0	1
Pose.Y	1/0	0	Lidar.Read	0	1
Pose.Theta	1/0	0	Lidar.Std	1/0	0
Pose	0	1	GPS.X	0	0
Velocity.Linear	1/0	0	GPS.Y	0	0
Velocity.Angular	1/0	0	GPS.Theta	0	0
Velocity	0	1	GPS	0	1
Velocity.Angular.Left	1/0	0	GPS.Std.X	1/0	0
Velocity.Angular.Right	1/0	0	GPS.Std.Y	1/0	0
Velocity.Wheels	0	1	GPS.Std.Y	1/0	0
Controller.LowLevel	1/0	0	GPS.Std	0	1
Odometry.X	1/0	0	World.Camera.Mode	1/0	0
Odometry.Y	1/0	0	World.Camera.Zoom	1/0	0
Odometry.Theta	1/0	0	World.Camera.Zoom	1/0	0
Odometry	0	1	World.Mode	1/0	0
Odometry.Std.Linear	1/0	0	World.Robot	1/0	0
Odometry.Std.Angular	1/0	0	World.Robot	1/0	0
Odometry.Std	0	1	World.Reset	1	0
Trace	1/0	0	World.Maze.Clear	1	0
Time.Wait	0	0	World.Maze.Load	2	0

Tabela 4 – Funções para troca de dados entre o cliente e servidor

Cada uma das funções acima lidam com uma característica específica do simulador, onde algumas delas permitem alterar o comportamento do simulador e outras do robô.

Cada uma das funções da Tabela 4 são descritas à seguir:

- **Pose:** A função Pose e suas especializações (Pose.X, Pose.Y e Pose.Theta) permite obter e configurar a posição global do robô. A posição utilizada por esta função é a posição real do robô na simulação, sem a adição de erros.
- **Velocity:** A função Velocity permite obter e configurar as velocidades aplicadas no robô. O robô suporta dois tipos de controle de velocidade, o controle de baixo nível, onde a velocidade angular de cada roda é configurada, e o controle de alto nível, onde a cinemática do robô é utilizada para que seja possível configurar a velocidade angular e linear do robô. O modo de controle padrão é o controle de alto nível.
- **Controller.LowLevel:** Esta função permite alterar entre os modos de controle de baixo e alto nível.
- **Odometry:** A função Odometry permite obter a posição que o robô acredita estar baseando-se na sua movimentação. A movimentação do robô está sujeita à erros,

definidos pela função Odometry.Std, que fazem com que ao mover-se, a posição que ele acredita estar seja diferente da posição que ele realmente estar.

- **Odometry.Std:** A função Odometry.Std permite obter e configurar os erros associados à movimentação do robô. Com esta função é possível efetuar uma simulação bem mais precisa do robô, uma vez que no robô real os erros sempre estarão presentes.
- **Trace:** A função Trace faz com que o simulador desenhe dois caminhos associados ao robô, um para indicar sua posição real e outro para indicar sua posição segundo a odometria. Com esta função se torna fácil analisar os efeitos da consideração dos erros de movimentação do robô.
- **Time.Wait:** Esta função permite que o código sendo executado no cliente aguarde o próximo ciclo de simulação do simulador, desta forma, é possível aguardar eventos que não são imediatos, como capturar um novo frame da câmera ou efetuar uma nova leitura com o lidar. Esta função irá retornar o tempo gasto pelo ciclo de simulação atual
- **Camera.Capture:** Esta função utiliza a câmera presente no robô para efetuar uma captura e retornar a imagem no formato RGB. A câmera presente no robô possui a resolução de 320 pixels de largura por 240 pixels de altura. Como cada linguagem de programação tem seu próprio formato de imagens, a função Camera.Capture não força nenhum formato específico, ao invés disso, retorna no campo *Data<sub>s</sub>* a quantidade de bytes da imagem ( $320 \times 240 \times 3 = 230400$ ), e no campo *Data* a sequência de todos os bytes da imagem, ficando como responsabilidade do biblioteca de comunicação no cliente converter estes dados para o formato de imagens da linguagem de programação utilizada.
- **Lidar.Read:** A função Lidar.Read retorna as últimas leituras do Lidar presente no robô. Esta função, por padrão irá retornar no campo *Data<sub>s</sub>* a quantidade de leituras feitas pelo Lidar (36), e no campo *Data* um vetor de números decimais de 32 bits indicando a distância para cada um dos ângulos. Por questões de facilidade a biblioteca de comunicação fará a conversão para pontos cartesianos, portanto, esta função irá retornar um objeto contendo quatro informações:
  - Lista de ângulos
  - Lista de distâncias
  - Lista de posições cartesianas relativas ao robô

Considerando um ângulo  $\alpha_i$  e uma distância  $\rho_i$ , a posição relativa ao robô  $XY_i$  é calculada como:

$$\begin{aligned} X_i &= -\rho_i \cos\left(\frac{\pi}{2} + \alpha_i\right) \\ Y_i &= \rho_i \sin\left(\frac{\pi}{2} + \alpha_i\right) \end{aligned} \tag{3.33}$$

- **Lidar.Std:** A função Lidar.Std permite obter e configurar os erros associados ao Lidar do robô, desta forma, é possível simular o Lidar tanto com erros associados à distância quanto como um sensor ideal, sem erros adicionados.
- **GPS:** Esta função permite obter a posição GPS do robô, bem como seu ângulo segundo a buscula virtual. Esta posição se aproxima bastante da posição real do robô, sendo influenciada apenas pelos erros do próprio sensor GPS e da buscula.
- **GPS.Std:** A função GPS.Std permite obter e configurar os erros associados ao sensor GPS e à buscula virtual do robô. Se estes parâmetros forem iguais à 0, o GPS irá retornar a posição real do robô, enquanto que se este valor for aumentado, a posição retornada também terá seu erro aumentado.
- **World.Camera.Mode:** Esta função permite alterar via código o tipo de câmera utilizada pelo simulador para visualizar o robô, da mesma forma que o botão *Camera* presente na interface do simulador. Estarão disponíveis 3 opções de câmera, sendo elas:
  0. Câmera em 3ºpessoa
  1. Câmera de topo
  2. Câmera em 1ºpessoa

Ao se utilizar a câmera em 1º pessoa, a visão o simulador será a mesma da imagem retornada pelo comando Camera.Capture.

- **World.Camera.Zoom:** Esta função permite alterar o zoom da câmera do simulador. O valor para esta função pode variar de 0 à 1, onde 0 indica muito próximo do robô e 1 indica muito longe.
- **World.Mode:** A função World.Mode permite alterar entre os dois modos de simulação, da mesma forma que pode ser feito pelos botões presentes na interface do simulador. As opções são:
  0. Modo realístico
  1. Modo maze
- **World.Robot:** Esta função permite selecionar o robô atualmente em uso. Na versão atual do simulador apenas um robô está disponível, portanto o único valor disponível para esta função é 0, porém, como descrito na seção *Trabalhos Futuros*, quando outros robôs forem adicionados ao simulador será possível alterar entre o controle de cada um deles.
- **World.Reset:** Esta função permite reinicializar a simulação, fazendo com que o robô volte ao seu estado inicial, ou seja, posição  $0 \times 0$  e todas as velocidades nulas.

Esta função também possui um botão na tela do simulador, uma vez que ela é muito útil caso o robô caia ou fique preso em alguma estrutura.

- ***World.Maze.Clear:*** Esta função só é válida se o simulador estiver no modo maze. Ela fará com que o simulador limpe qualquer mapa previamente carregado, deixando a tela de simulação apenas com o robô e o solo.
- ***World.Maze.Load:*** Esta função também só é válida se o simulador estiver no modo maze, onde ela irá fazer com que o simulador limpe o mapa atualmente carregado e carregue um novo, dado pelo valor *Data* do pacote de dados.

## 4 ESTUDOS DE CASO

Neste capítulo serão descritos dois estudos de caso para o simulador desenvolvido. Será descrito o desenvolvimento de duas aplicações, portanto, este capítulo será dividido da seguinte maneira:

- A seção 4.1 descreve o desenvolvimento de um controlador PID para o robô utilizando a linguagem Python.
- A seção 4.2 descreve o desenvolvimento de um algoritmo capaz de fazer o robô identificar um caminho e seguí-lo utilizando técnicas de visão computacional.

### 4.1 Controle PID

Neste estudo de caso será feito o controle de posição do robô utilizando um controlador Proporcional-Integral-Derivativo (PID) na linguagem de programação Python.

Em sua forma clássica, o controlador PID possui a seguinte forma:

$$u(t) = K_p e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{\partial e(t)}{\partial t} \quad (4.1)$$

A Equação 4.1 pode ser representada através do diagrama de blocos da Figura 23.

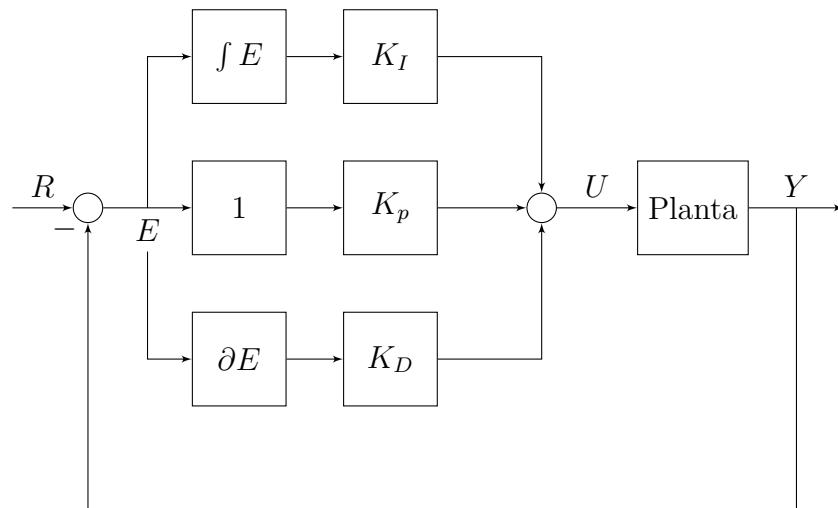


Figura 23 – Diagrama de blocos do controle PID clássico

Para encontrar os parâmetros para o controle do robô, inicialmente considera-se este robô em um plano cartesiano, onde o robô está localizado no ponto  $X_R, Y_R$  com a orientação  $\theta_R$ , e, o ponto objetivo está localizado no ponto  $X_2, Y_2$ . Define-se a diferença entre a posição do robô e do ponto objetivo como:

$$\begin{aligned}\Delta_X &= X_2 - X_1 \\ \Delta_Y &= Y_2 - Y_1\end{aligned}\quad (4.2)$$

Considerando este modelo em coordenadas polares, é possível definir a distância do robô até o ponto objetivo e o ângulo de orientação do robô até este ponto como:

$$\begin{aligned}\rho &= \sqrt{(\Delta_X^2 + \Delta_Y^2)} \\ \gamma &= \operatorname{tg}^{-1}\left(\frac{\Delta_Y}{\Delta_X}\right) \\ \alpha &= \gamma - \theta_R\end{aligned}\quad (4.3)$$

Considerando o modelo cinemático do robô e derivando a Equação 4.3, encontra-se o modelo dinâmico do robô em malha aberta, que será descrito por:

$$\begin{vmatrix} \dot{\rho} \\ \dot{\alpha} \end{vmatrix} = \begin{vmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \end{vmatrix} \begin{vmatrix} V \\ \omega \end{vmatrix} \quad (4.4)$$

Com base nisto, define-se a lei de controle para o robô como:

$$\begin{aligned}V_t &= \min(K_p \rho_t, V_{max}) \\ \omega_t &= K_P \alpha_t + K_I \sum_{k=0}^t \alpha_k + K_D (\alpha_t - \alpha_{t-1})\end{aligned}\quad (4.5)$$

Tem-se que a velocidade linear no instante  $t$ ,  $V_t$  é dada por uma proporção da distância até o ponto objetivo no instante  $t$ , ou seja, para a velocidade linear é feito o um controle Proporcional. Se o robô tiver que passar por mais de um ponto objetivo, faz-se sua velocidade linear ser igual à velocidade máxima para atingir todos os pontos objetivos, com exceção do último, onde o controle proporcional irá garantir que o robô não passe do ponto. Já a velocidade angular no instante  $t$ ,  $\omega_t$ , depende do ângulo  $\alpha_t$  no instante  $t$ , do ângulo  $\alpha_{t-1}$  no instante  $t-1$  e do somatório de todos os ângulos  $\alpha$  desde o inicio da contagem de tempo, desta forma, pode-se ver que para a velocidade angular é feito um controle PID.

Com a lei de controle do robô definida, ja é possível implementar a comunicação com o simulador para aplicar esta lei de controle. Inicialmente, importa-se a biblioteca *robotone* e assim obtém-se as funções para conectar ao simulador. Importa-se também a biblioteca *math* para utilizar as funções *atan2* e *sqrt*. Utiliza-se a função *connectRobotOne* para conectar ao simulador. Esta função recebe como parâmetro o endereço do simulador na rede e retorna um handler, que representa a comunicação do código com o simulador. Se nenhum

endereço é passado para esta função, o endereço do próprio cliente é utilizado, ou seja, se o simulador e o código de controle estiver sendo executado no mesmo computador, não é preciso passar um endereço. Como foi utilizada a função *connectRobotOne* para conectar ao simulador, utiliza-se a função *disconnectRobotOne* para desconectar do simulador, deixando-o livre para outros clientes. Além de conectar e desconectar do simulador, também utiliza-se a função *traceRobotOne* para habilitar o traçado da movimentação do robô.

---

```

1 import math
2 from robotone import *
3 handler = connectRobotOne("127.0.0.1")
4 traceRobotOne(handler, true)
5 # Código de controle
6 traceRobotOne(handler, false)
7 disconnectRobotOne(handler)

```

---

Em seguida, define-se a máxima velocidade linear do robô, que para este caso será  $V_{max} = 1m/s$ . Também deve-se definir qual deve ser a distância máxima que o robô pode estar do ponto objetivo, isto garante que o robô pare próximo o suficiente do ponto objetivo mas ao mesmo tempo não demore efetuando o ajuste de posição até ponto, para este caso, a distância máxima escolhida foi  $\epsilon = 0.05m$ .

Analizando a Equação 4.5 percebe-se também que é preciso definir os parâmetros do controlador. Deseja-se que o robô se move rápido até o ponto objetivo e também que ajuste seu ângulo com o ponto objetivo de forma rápida, portanto, são definidos os parâmetros  $K_\rho = 1$  e  $K_P = 1$ . Utiliza-se valores pequenos para os outros parâmetros pois deseja-se que o somatório dos erros e a diferença com o estado anterior tenha uma pequena contribuição no controle, desta forma, definem-se os valores  $K_I = 0.01$  e  $K_D = 0.01$ . Como no controle PID tem-se um somatório dos ângulos  $\alpha$  passados e a diferença do ângulo  $\alpha$  da iteração passada, define-se estes valores inicialmente nulos para garantir que nenhum erro seja adicionado. No código em matlab, estas variáveis são declaradas como:

---

```

1 Vmax = 1.0
2 erro = 0.05
3 Krho = 1
4 Kp = 1
5 Ki = 0.01
6 Kd = 0.01
7 sum_alpha = 0

```

---

```
8 alpha_1 = 0
```

---

O próximo passo é definir o ponto de partida do robô e os pontos objetivos. Define-se o robô na posição 0,0 e os pontos objetivo formando um quadrado de lado 10m, onde um dos vértices é a posição inicial do robô. Também utiliza-se a função *pose* passando uma posição para configurar a posição inicial do robô no simulador.

---

```
1 P = [0, 0, 0]
2 GS = [[0, 0], [10, 0], [10, 10], [0, 10]]
3 pose(handler, P)
```

---

Para cada ponto objetivo  $i$  calcula-se a diferença de posições entre o robô e o ponto objetivo, de forma a se calcular a distância entre os dois. Para obter a posição atual do robô utiliza-se a função *pose* sem passar uma posição .

---

```
1 G = GS[i]
2 P = pose(handler)
3 dx = G[0] - P[0]
4 dy = G[1] - P[1]
5 rho = math.sqrt(dx*dx + dy*dy)
```

---

Enquanto a distância  $\rho$  for maior do que o erro  $\epsilon$ , calcula-se os parâmetros  $\rho$  e  $\alpha$ . É desejado que o módulo do ângulo  $\alpha$  esteja entre  $0^\circ$  e  $180^\circ$ , para garantir isto, utiliza-se função *fixAngle*.

---

```
1 P = pose(handler)
2 dx = G[0] - P[0]
3 dy = G[1] - P[1]
4 rho = math.sqrt(dx*dx + dy*dy)
5 gama = math.atan2(dy, dx)
6 alpha = fixAngle(gama - P[2])
```

---

Com os valores de  $\rho$  e  $\alpha$  definidos aplica-se a lei de controle utilizando a Equação 4.5. Como existe mais de um ponto objetivo, utiliza-se a velocidade linear máxima em todos os pontos anteriores ao último. Após o cálculo das velocidades também altera-se as variáveis de auxílio no somatório e na diferença dos ângulos  $\alpha$ . Para passar a velocidade calculada para o robô utiliza-se a função *velocity*.

---

```

1 if i < len(GS)-1:
2     v = Vmax
3 else:
4     v = min(Krho*rho, Vmax)
5 w = Kp*alpha + Ki*sum_alpha + Kd*(alpha - alpha_1)
6 alpha_1 = alpha
7 sum_alpha = sum_alpha + alpha
8 velocity(handler, (v w))

```

---

Após aplicar as velocidades no robô, utiliza-se a função *waitRobotOne* forçar o código a esperar o próximo ciclo de simulação e, desta forma, evita-se que o código aplique a mesma velocidade antes do robô se mover.

Na Figura 24 visualiza-se o robô executando o código descrito neste estudo de caso. O trajeto feito pelo robô pode ser visto no rastro deixado pelo robô.

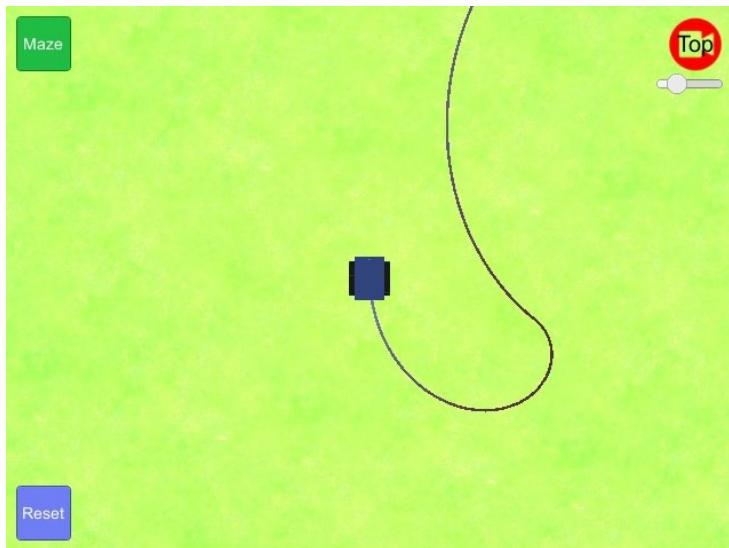


Figura 24 – Robô alcançando pontos utilizando o controle PID

O código em Python completo para o controle PID no robô é:

---

```

1 import math
2 from robotone import *
3 handler = connectRobotOne("127.0.0.1")
4 traceRobotOne(handler, true)
5
6 Vmax = 1.0
7 erro = 0.05
8 Krho = 1

```

```

9 Kp = 1
10 Ki = 0.01
11 Kd = 0.01
12 sum_alpha = 0
13 alpha_1 = 0
14
15 P = [0, 0, 0]
16 GS = [[0, 0], [10, 0], [10, 10], [0, 10]]
17 pose(handler, P)
18
19 for i in range(len(GS)):
20     G = GS[i]
21     P = pose(handler)
22     dx = G[0] - P[0]
23     dy = G[1] - P[1]
24     rho = math.sqrt(dx*dx + dy*dy)
25
26     while rho < erro:
27         P = pose(handler)
28         dx = G[0] - P[0]
29         dy = G[1] - P[1]
30         rho = math.sqrt(dx*dx + dy*dy)
31         gama = math.atan2(dy, dx)
32         alpha = fixAngle(gama - P[2])
33
34     if i < len(GS)-1:
35         v = Vmax
36     else:
37         v = min(Krho*rho, Vmax)
38         w = Kp*alpha + Ki*sum_alpha + Kd*(alpha - alpha_1)
39         alpha_1 = alpha
40         sum_alpha = sum_alpha + alpha
41         velocity(handler, (v w))
42         waitRobotOne(handler)
43
44 traceRobotOne(handler, false)
45 disconnectRobotOne(handler)

```

---

## 4.2 Seguidor de caminho

O segundo estudo de caso será uma implementação na linguagem Matlab capaz fazer com que o robô siga a estrada cinza presente no modo realístico do simulador.

Para permitir que o robô siga a estrada utiliza-se a câmera presente no robô para capturar imagens e processá-las, com o intuito de identificar a estrada e fazer o robô se mover acima dela. Para capturar a imagem da câmera utiliza-se a função *captureCamera*.

As imagens capturadas pela câmera do robô estão no espaço de cores RGB, porém este espaço de cores é pouco eficiente para efetuar a separação de cores, uma vez que a iluminação não está separada das cores. Para resolver este problema, efetua-se a conversão do espaço de cores RGB para o espaço de cores HSV, onde as cores estão separadas da iluminação e da saturação, e para tal, utiliza-se a função *rgb2HSV*.

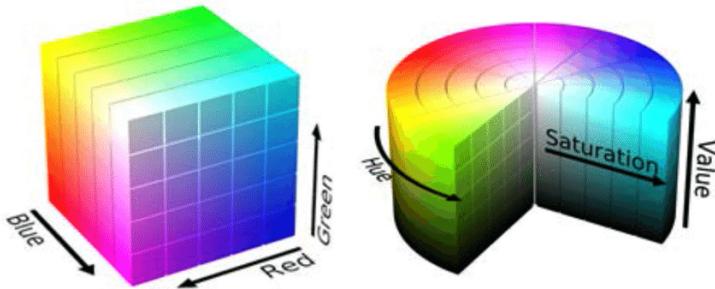


Figura 25 – Espaços de cores RGB e HSV

Para separar a cor cinza das outras cores, definem-se os limites dos canais *Hue*, *Saturation* e *Value*. O canal *Hue* indica a matriz de cores, portanto não é desejado que este canal seja limitado. Analisando a Figura 25 é possível notar que quando menor o valor dos canais *Saturation* e *Value*, mais escura será a cor, portanto, limita-se estes canais para selecionar apenas o cinza. Os valores de seleção da cor foram escolhidos de forma experimental. Vale notar que os valores devem ser normalizados pois a função *rgb2HSV* normaliza os valores da imagem.

```

1 minval = [0 0 0]/255;
2 maxval = [255 90 180]/255;
```

Após definir os intervalos dos canais para a cor cinza, deve-se selecionar os pixels da imagem que estão dentro destes intervalos, em seguida, obtém-se uma faixa central na imagem onde é analisada a quantidade de pixels cinza.

```

1 mask = true(size(hsv,1), size(hsv,2));
2 for p = 1 : 3
3     mask = mask & (hsv(:,:,p) >= minval(p) & hsv(:,:,p) <= maxval(p));
```

```

4 end
5
6 lineMask = mask(150:160, :);

```

Após selecionar a área central da imagem, se houver algum pixel cinza nesta área, calcula-se o ponto médio da área cinza e desta forma é possível calcular a velocidade angular necessária para fazer o robô seguir este ponto médio. Para calcular a velocidade angular, utiliza-se a Equação 4.6, onde  $x, y$  é o ponto médio da área cinza da faixa central da imagem.

$$\omega = 0.6 \tan^{-1} \left( \frac{160 - x}{80} \right) \quad (4.6)$$

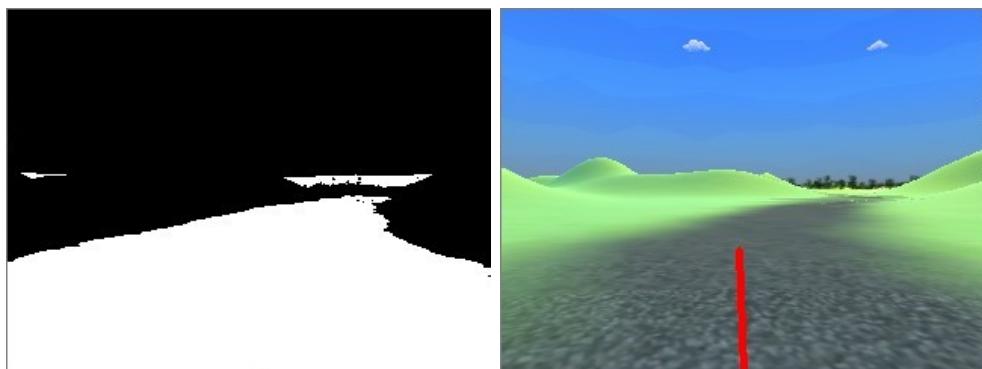
```

1 mrow = sum(row)/length(row);
2 mcol = sum(col)/length(col);
3 x = mcol;
4 y = mrow + 150;
5 w = 0.6*atan2(160-x, 80);

```

Aplica-se a velocidade angular calculada utilizando a função *velocity*. A velocidade linear do robô não é calculada, apenas um valor fixo é aplicado. Utiliza-se a função *waitRobotOne* para aguardar mais um ciclo de simulação do simulador, pois, desta forma o simulador poderá aplicar a velocidade calculada e poderá capturar uma nova imagem.

Na Imagem 26a é possível ver o processo de separação da cor cinza na imagem, e na Imagem 26b pose-se verificar a direção do robô sendo calculada, representada pela linha vermelha.



(a) Separação da cor cinza na imagem (b) Identificação da direção da estrada

Figura 26 – Processamento de imagem da câmera do robô

Na Figura 27 é possível ver o robô seguindo corretamente a estrada utilizando a técnica descrita neste estudo de caso. É possível ver pelo rastro deixado pelo robô que o

caminho foi seguido corretamente.

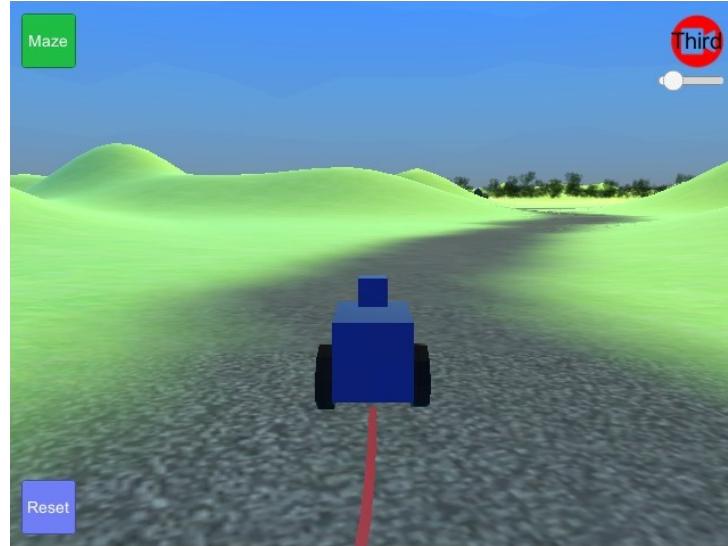


Figura 27 – Imagem do robô seguindo a estrada no simulador

O código completo para seguir um caminho no Matlab é apresentado abaixo:

```

1 handler = connectRobotOne('127.0.0.1');
2 traceRobotOne(handler, true);
3
4 MaxV = 2.0;
5
6 for n = 1:1000
7     rgb = captureCamera(handler);
8     hsv = rgb2hsv(rgb);
9
10    minval = [0 0 0]/255;
11    maxval = [255 90 180]/255;
12
13    mask = true(size(hsv,1), size(hsv,2));
14    for p = 1 : 3
15        mask = mask & (hsv(:,:,p) >= minval(p) & hsv(:,:,p) <= ...
16                      maxval(p));
17    end
18
19    lineMask = mask(150:160, :);
20    [row, col] = find(lineMask);
21
22    w = 0;
23    if length(row) > 0
24        mrow = sum(row)/length(row);
25        mcol = sum(col)/length(col);
26        x = mcol;

```

```
26         y = mrow + 150;
27         w = 0.6*atan2(160-x, 80);
28         disp(w)
29     end
30     velocity(handler, [MaxV w]);
31
32     figure(1)
33     imshow(rgb)
34     figure(2)
35     imshow(mask)
36     waitRobotOne(handler)
37 end
38
39 traceRobotOne(handler, false);
40 disconnectRobotOne(handler);
```

## 5 CONCLUSÕES E TRABALHOS FUTUROS

### 5.1 CONCLUSÕES

Neste trabalho foi desenvolvido um simulador que permite a simulação de robôs móveis em um ambiente realístico. Foi verificado que a plataforma desenvolvida se mostrou eficiente no desenvolvimento das técnicas utilizadas nas aulas de robótica móvel ministradas na Universidade Federal de Juiz de Fora, e portanto este simulador pode ser utilizado como ferramenta de auxílio durante aulas. O simulador ainda carece de recursos presentes em outros simuladores do gênero como o V-Rep, porém possui uma interface de programação mais simples e portanto tem uma curva de aprendizado menos ingrime.

O desenvolvimento deste simulador permitiu que diversas disciplinas estudadas no Curso de Graduação em Engenharia Elétrica fossem aplicadas na prática, uma vez que no simulador é preciso implementar diversos recursos além do próprio robô, como por exemplo a física, captura de imagens, entre outros.

### 5.2 TRABALHOS FUTUROS

- Suporte à robôs manipuladores no simulador.
- Suporte à adição dinâmica de robôs no simulador, tendo o modelo tridimensional e o modelo cinemático como entradas.
- Desenvolvimento de um robô real que implemente a biblioteca de comunicações, de forma que os mesmos códigos desenvolvidos para o simulador sejam utilizáveis neste robô.
- Criação de novos sensores para o robô.
- Suporte à edição de sensores e itens nos robôs.
- Edição do mapa utilizando o mouse e teclado, podendo adicionar textures, relevo, objetos e outros robôs.
- Implementação de um interpretador de alguma linguagem de programação ou de uma linguagem desenvolvida especificamente para o simulador, de forma que seja possível desenvolver códigos utilizando o próprio simulador e desta forma ser possível executar códigos nos robôs sem a necessidade de um programa externo, podendo assim criar simulações com agentes inteligentes agindo na simulação enquanto o robô principal é controlado pelo código do usuário.
- ...

## REFERÊNCIAS

- [1] A. S. Barbosa. (2018) Robot one repository. [Online]. Available: <https://github.com/AlexanderSilvaB/Robot-One>
- [2] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.
- [3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.
- [4] E. Rohmer, S. P. N. Singh, and M. Freese, “V-rep: a versatile and scalable robot simulation framework,” in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.
- [5] B. P. Gerkey, R. Vaughan, and A. Howard, “The player/stage project: Tools for multi-robot and distributed sensor systems,” *Proceedings of the International Conference on Advanced Robotics*, 08 2003.
- [6] A. Larmuseau and D. Clarke, “Formalizing a secure foreign function interface,” in *Software Engineering and Formal Methods*. Springer, 2015, pp. 215–230.
- [7] U. Technologies. (2018) Unity 3d. [Online]. Available: <https://unity3d.com>
- [8] L. R. Olivi, “Cinemática,” 2018, [Parte 2 dos slides de cinemática do material didático da disciplina ENE122 - Robótica Móvel].
- [9] A. El-Rabbany, *Introduction to GPS: The Global Positioning System*, ser. Artech House mobile communications series. Artech House, 2002. [Online]. Available: <https://books.google.com.br/books?id=U2JmghrrB8cC>
- [10] J. Shan and C. Toth, *Topographic Laser Ranging and Scanning: Principles and Processing, Second Edition*. Taylor & Francis, 2018. [Online]. Available: [https://books.google.com.br/books?id=N\\_ErDwAAQBAJ](https://books.google.com.br/books?id=N_ErDwAAQBAJ)
- [11] O. Yadid-Pecht and R. Etienne-Cummings, *CMOS Imagers: From Phototransduction to Image Processing*, ser. Fundamental Theories of Physics Series. Springer US, 2007. [Online]. Available: <https://books.google.com.br/books?id=5dQRBwAAQBAJ>
- [12] N. Croce, *Newton and the Three Laws of Motion*, ser. Primary sources of revolutionary scientific discoveries and theories. Rosen Publishing Group, 2005. [Online]. Available: <https://books.google.com.br/books?id=D-S8c9AYjzIC>

- [13] J. Neto, *Mecânica Newtoniana, Lgrangiana e Hamiltoniana*. Editora Livraria da Física, 2004. [Online]. Available: [https://books.google.com.br/books?id=FyHOW\\_tvT8YC](https://books.google.com.br/books?id=FyHOW_tvT8YC)