CENTRO UNIVERSITÁRIO INTERNACIONAL UNINTER
ESCOLA SUPERIOR POLITÉCNICA

SLAM MONOCULAR UTILIZANDO ROS

AMOS JUNIOR
ORIENTADOR: CHARLES WAY HUN FUNG
ENGENHARIA DA COMPUTAÇÃO
10º PERÍODO

CURITIBA - PR
2020

RESUMO

Este trabalho busca poder realizar o processo de SLAM (Localização e mapeamento simultâneo) utilizando apenas uma câmera monocular, o algoritmo desenvolvido será feito utilizando o framework ROS (*Robot Operating System*). Para a varredura do ambiente será utilizado um robô Alphabot2 Pi, todo seu controle será feito via rede distribuída onde existirá uma máquina virtual para realizar os processos de cálculos referentes ao ambiente e distância dos objetos, armazenamento das informações de mapeamento e determinação do posicionamento do robô, este qual servirá como um atuador, se deslocando até os pontos definidos e coletando dados a partir da sua câmera monocular.

Palavras-chave: ROS, Varredura de ambiente, Alphabot2.

1. INTRODUÇÃO

Desde o momento em que a humanidade começou a utilizar a razão, foi iniciado um processo de evolução que foi tendo sucesso principalmente por observar o ambiente e a vida interagindo com este. Mesmo hoje as pessoas continuam aprendendo apenas observando, contudo, o avançar do tempo nos possibilitou utilizar a tecnologia para realizar este processo, isso ocorreu a partir de todo conhecimento adquirido anteriormente.

Uma das principais áreas que aplica tecnologia para obtenção de conhecimento de ambiente é a robótica, dentro desta, é chamado de localização e mapeamento simultâneo ou abreviadamente do inglês SLAM a técnica utilizada para a obtenção deste conhecimento.

A utilização dos robôs começou principalmente na indústria após a ocorrência da primeira guerra mundial, estes começaram a substituir os humanos em processos repetitivos e sequencias, pois, a produção não seria interrompida por necessidades biológicas, como dormir, comer ou atividades de lazer. Este termo robô surgiu pela definição feita por Karel Capek em sua peça teatral "Os Robôs Universais de Rossum" que contava a história de uma fábrica que fazia pessoas artificiais, chamadas de "*roboti",* podendo ser visto hoje nos livros de mesmo nome (Capek, 2004).

No decorrer dos anos a área de atuação dos robôs se expandiu para fora das indústrias, como na automação residencial, tecnologia espacial e até mesmo na medicina. Estas foram tendo sucesso principalmente devido a assertividade da Lei de Moore, a qual dizia que a cada período de 18 meses o número de transistores (componente essencial para hardware usado para amplificar ou trocar sinais eletrônicos e potência elétrica) dobrar dentro dos chips de computador (Gregersen, 2015), isto é um processo de miniaturização progressivo.

A navegação autônoma é outra área da robótica que teve uma expansão no último século, está consiste na capacidade de robôs poderem se locomover em um ambiente qualquer sem nenhuma intervenção humana de forma congruente (Thrun et al., 2000). O SLAM consiste na construção de um mapa a partir de uma varredura do ambiente com base na coleta de dados por sensores, a partir deste poder determinar a localização do autônomo no local, assim como objetos.

Normalmente a técnica de SLAM depende de um conjunto de sensores, isto acaba elevando o custo dos projetos, assim como impondo necessidade de espaço, portas e pinos de propósito geral das placas. Uma técnica que busca o mesmo propósito do SLAM é a Visual SLAM onde o objetivo é o mesmo, porém sendo realizado apenas pela captura de imagens, como a maioria dos robôs

responsáveis por este papel comumente já contarão com um módulo de câmera se torna uma técnica interessante.

1.1 PROBLEMA

A robótica inteligente e futurista como vista em filmes de ficção para seu sucesso depende que os robôs possam ser verdadeiramente autônomos, isto é, os robôs necessitam da capacidade de entender o ambiente, estimar sua localização e coletar dados deste para que assim possa realizar a função para que foi designado.

Com o avançar da tecnologia nas últimas décadas tivemos a possibilidade de cada vez mais construir este tipo de robô, porém ainda estamos suscetíveis a diversas limitações. Uma desta continua sendo é em relação ao tamanho dos robôs, quando falamos de um autônomo este normalmente conterá um conjunto de sensores para que possa com precisão realizar seu papel, normalmente isto acaba sendo um fator limitante. Outro ponto seria a limitação em criar vários robôs deste tipo visto que a necessidade de diversos tipos de sensores elevaria o custo do projeto.

1.2 JUSTIFICATIVA

Na robótica a técnica de SLAM possibilita que um robô possa se localizar e orientar de forma totalmente autônoma (Thrun et al., 2000), a partir disto podemos desenvolver diversas aplicações específicas dependendo do tamanho do robô, seu hardware e os sensores disponíveis. Processos não viáveis para os hu-

manos ou a tecnologia atual se tornam possíveis, como identificação dos ambientes subaquáticos, varredura de superfície de planetas ou asteroides. Atividades cotidianas também podem ser automatizadas, como exemplo a demanda de robôs autônomos para realizar limpeza de ambientes sem interferência humana.

Para realizar está técnica diversos tipos de sensores podem ser empregados, contudo visto que os recursos podem ser limitados principalmente em robôs pequenos, poder realizar este processo de maneira eficiente utilizando o mínimo de processamento, tamanho e consumo se torna extremamente importante, possibilitando maior alocação de recursos para outras funções. A possibilidade de utilizar apenas uma câmera para o processo acaba se tornando uma solução eficaz visto que a maiorias dos robôs responsáveis pela realização desta técnica já contarão com um módulo de câmera, isto possibilita a diminuição das placas utilizadas, ou a possibilidade da utilização de recursos que normalmente seriam descartados dos projetos pelas limitações de hardware impostas pela utilização de um conjunto de sensores para este objetivo.

1.3 OBJETIVOS

Objetivo geral: Realização da técnica de SLAM monocular.

Objetivos específicos:

- Controlar o robô alphabot2 pi com o framework ROS;
- Utilizar conexão SSH com ROS para controle distribuído do robô;
- Mapeamento de ambiente e geração de mapa;

- Geração de dados de Odometria;
- Algoritmo para navegação autônoma;
- Utilização de módulo de câmera para captura de ambiente;
- Utilização de motores para movimentação;
- Utilização de servos para angulação de câmera;
- Comparação com técnica de mapeamento a laser;

2 REVISÃO BIBLIOGRÁFICA

2.1 ALPHABOT 2 PI

Para poder realizar o processo de SLAM monocular utilizando um robô é necessário que este contemple os módulos necessários:

- Módulo para movimentação;
- Sensor para leitura do ambiente;
- Hardware para processamento;
- Formas de comunicação;
- Tamanho adequado;

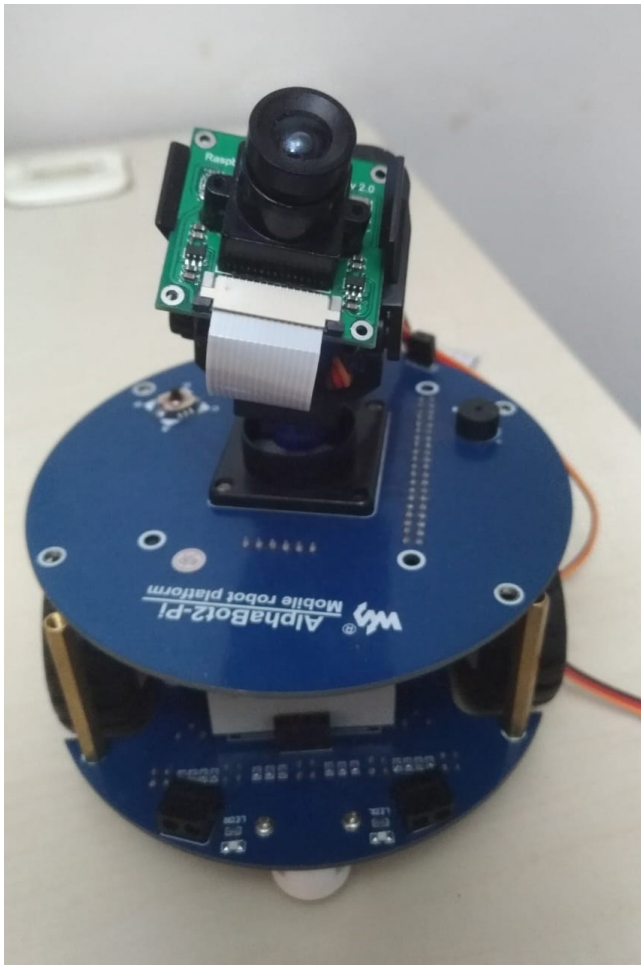
O AlphaBot2 Pi é um kit versátil que contempla diversos módulos conforme a figura 1, é projetado para suportar uma Raspberry, conta com motores de corrente contínua para movimentação, assim como uma base com câmera conectada a dois servos motores que fazem a angulação desta de forma precisa.

Suas demais configurações são as seguintes conforme dita o manual (Waveshare, 2017):

- 1- AlphaBot2 control interface: for connecting sorts of controller adapter board
- 2- Ultrasonic module interface
- 3- Obstacle avoiding indicators
- 4- Omni-direction wheel
- 5- ST188: reflective infrared photoelectric sensor, for obstacle avoiding
- 6- ITR20001/T: reflective infrared photoelectric sensor, for line tracking
- 7- Potentiometer for adjusting obstacle avoiding range
- 8- TB6612FNG dual H-bridge motor driver
- 9- LM393 voltage comparator
- 10- N20 micro gear motor reduction rate 1:30, 6V/600RPM
- 11- Rubber wheels diameter 42mm, width 19mm
- 12- Power switch
- 13- Battery holder: supports 14500 batteries
- 14- WS2812B: true color RGB LEDs
- 15- Power indicator
- 16- AlphaBot2 control interface: for connecting AlphaBot2-Base
- 17- Raspberry Pi interface: for connecting Raspberry Pi 3 Model B
- 18- Servo interface
- 19- USB TO UART: easy for controlling the Pi via UART
- 20- LM2596: 5V voltage regulator

- 21- TLC1543: 10-bit AD acquisition chip, allows the Pi to use analog sensors
- 22- PCA9685: servo controller, make it more smoothly to rotate the pan head
- 23- CP2102: USB TO UART converter
- 24- Joystick
- 25- IR receiver
- 26- Buzzer

Figura 1 – Alphabot2 Pi.

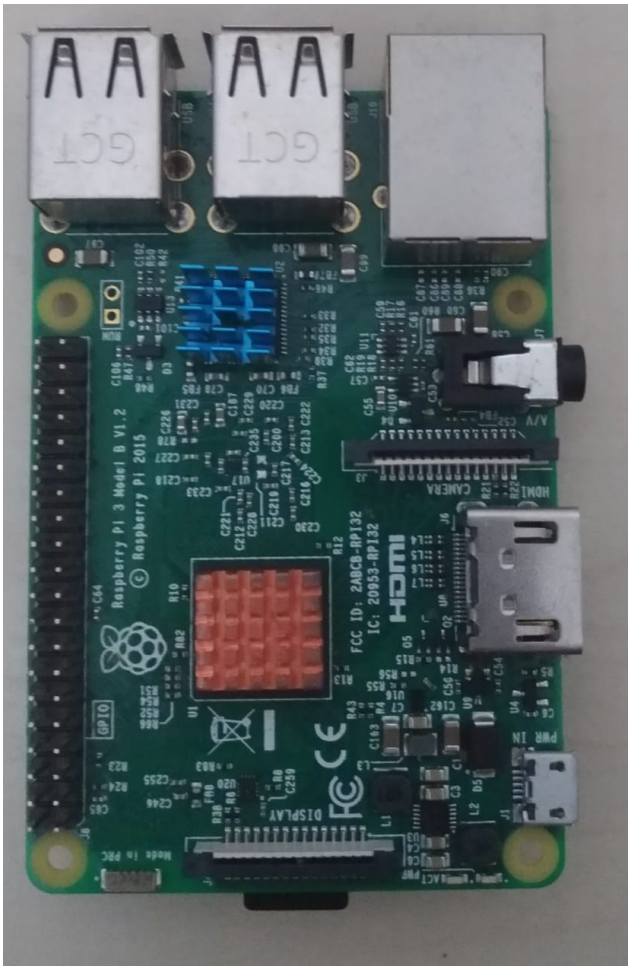


Fonte: Autor.

2.2 RASPBERRY

A Raspberry é um computador de baixo custo que surgiu com o propósito de trazer mais pessoas para explorar o mundo da computação, criada por quatro alunos da Universidade de Cambridge conforme dito por (Siqueira, 2018). Consiste em uma placa com rede Wi-fi e ethernet, que conectado a um monitor, e a utilização teclado e mouse funcionaria como um computador normal com um certo limite computacional. Porém a grande versatilidade é o fato de possuir diversos GPIO (pinos de entrada e saída de propósito geral), por meio destes é possível conectar diversos módulos como câmeras, sensores a laser, dispositivos de emissão sonora. O controle destes pode ser feito utilizando diversas linguagens de programação assim como diversas bibliotecas devido ao sistema operacional, atualmente podem ser utilizadas distribuições específicas Windows e diversas baseadas em Unix, tendo sua própria distribuição Raspberry OS (Raspberry.org, 2020), assim como a distribuição Ubiquity (Ubiquity_Robotics, 2018) específica para desenvolvimento robótico com o framework ROS.

Figura 2 – Raspberry.



Fonte: Autor.

O modelo atual com maior demanda da Raspberry Pi é o PI 3 B V1.2 exemplificado na figura 2, ele contém as seguintes especificações conforme o mencionado no manual do usuário (Raspberry.org, 2020):

- 1- BCM2837B0, 1.2GHz 64-bit quad-core ARM Cortex-A53
- 2- 1GB RAM
- 3- 10/100 Ethernet port
- 4- 802.11n WiFi NIC
- 5- Bluetooth 4.1 & Bluetooth Low Energy (BLE)

- 6- HDMI port
- 7- USB 2.0 interface x 4
- 8- Micro SD card slot
- 9- Combined 3.5mm audio jack and composite video
- 10- 40-pin GPIO interface
- 11- Camera interface (CSI)
- 12- Display interface (DSI)

2.3 CÂMERA

O modulo de câmera Raspberry pi conta com as seguintes configurações:

- Tamanho de 25 x 24 x 9 mm;
- Resolução de 8 MP
- Captura de vídeo de até 180p;
- Tamanho óptico de ¼";
- Lente focar de 3,04 mm;
- Sensor Sony IMX219;

Este módulo é próprio para este computador, isto proporciona algumas facilidades, como já possuir a entrada flat específica para este modulo não utilizando nenhum porta usb ou GPIO, assim como contar com diversos comandos de terminal para gravação de vídeo ou fotos, também encontramos uma documentação oficial (Pi_Camera_Module, 2020) sobre este modulo facilitando o uso e desenvolvimento, podemos observar o modulo na figura 3.

Figura 3 – Câmera Raspberry.

Fonte: Autor.

2.4 OPENCV

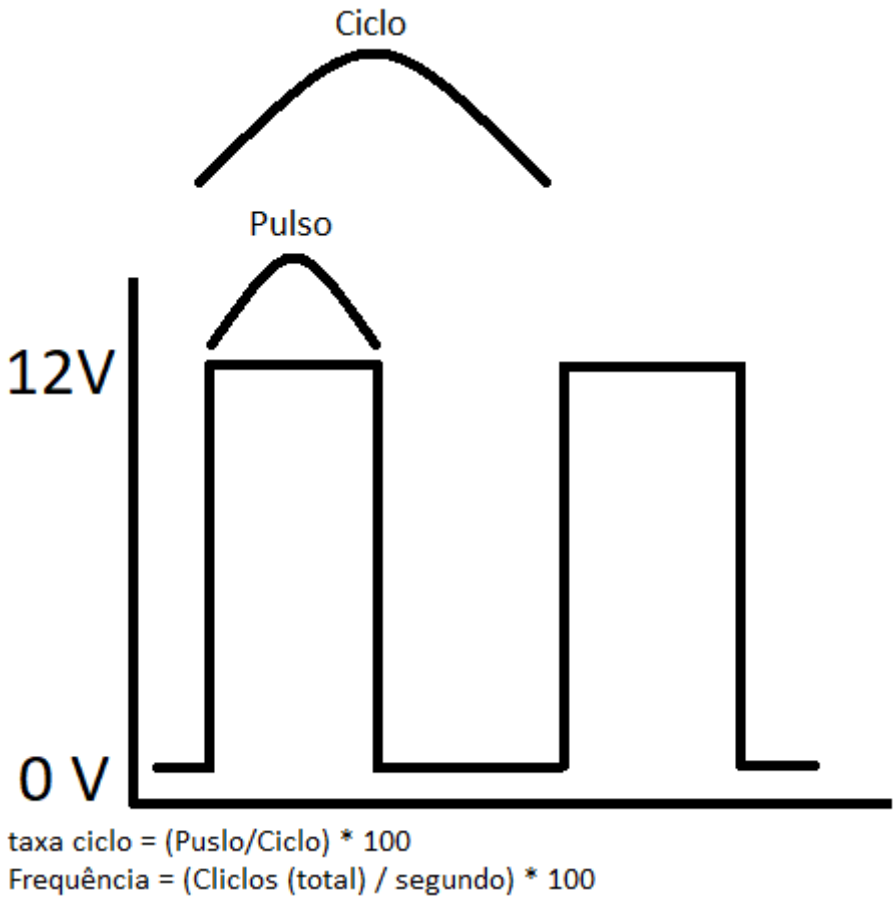
A principal biblioteca utilizada para o processamento de imagens é o OpenCV (OpenCv, 2020), está surgiu no início dos anos 2000, nas primeiras versões em nome da empresa Intel. Está biblioteca é *open source,* ou seja, código aberto, isto é, pode ser utilizado e modificado por qualquer pessoa, por meio disso sua comunidade cresceu muito rápido. Com esta contribuição diversos

módulos foram desenvolvidos para processamento de imagens, cada uma das implementações conta com diversas técnicas diferentes que podem ser utilizadas, assim diversos códigos para aplicações diversas acabam surgindo de maneira simples e eficiente.

2.5 SERVO MOTOR

Este tipo de motor recebe este nome pois é utilizado a partir de um controle, isto é, a partir de um sinal a posição deste é modificada para equivaler ao sinal emitido conforme programado podemos observar na figura 5 o servo motor. Devido a este tipo de funcionamento este tipo de motor possui um controle preciso de angulação. Os servos possuem bibliotecas em diversas linguagens para o controle, estas se baseiam no conceito de PWM, o que significa a possibilidade de controle do ângulo/posição do eixo, para tal é necessário definir um *duty cycle* (ciclo de trabalho), sendo este o tempo em que ocorre a ativação do servo motor, a taxa em que ocorre pode ser encontrada relacionando a sua ativação que acontece na no nível lógico alto (12V no caso dos servos) pelo tempo total de um ciclo, um ciclo é um período de onda conforme podemos observar na figura 4, este período é determinado quando ocorre um nível lógico alto e em seguida um nível lógico baixo (0V), ele se encerra quando retorna novamente ao nível lógico alto, podemos observar a ocorrência destes níveis nos pulsos da figura 4. Já a frequência em que ocorrem os ciclos pode ser definida pela contagem total de ciclos ocorridos por segundo conforme capítulo 3 (soares, 2017).

Figura 4 – Exemplificação do PWM.



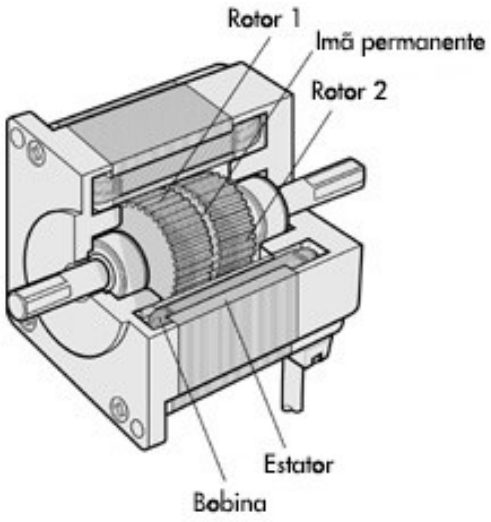
Fonte: Autor.

2.6 MOTOR CORRENTE CONTINUA

Para a movimentação os motores recomendados são de corrente contínua, este tipo de motor tem o potencial definido pelo seu tamanho e tensão. A energia elétrica utilizada para alimentar este é convertida em mecânica, esta conversão gera um campo magnético que atua entre os imãs, essa variação gera o chamado torque que faz o rotor girar no seu próprio eixo produzindo os movimentos de rotação conforme demonstrado na figura 5.

O uso para movimentação se deve, pois, apenas variando a tensão de entrada conseguimos controlar a velocidade do motor, o torque pode ser calculado pela relação da potência e a velocidade de rotação conforme definido por (Bertulucci, 2017).

Figura 5 – Rotor e imã motor elétrico.



Fonte: OrientalMotor, 2017.

2.7 FRAMEWORK ROS

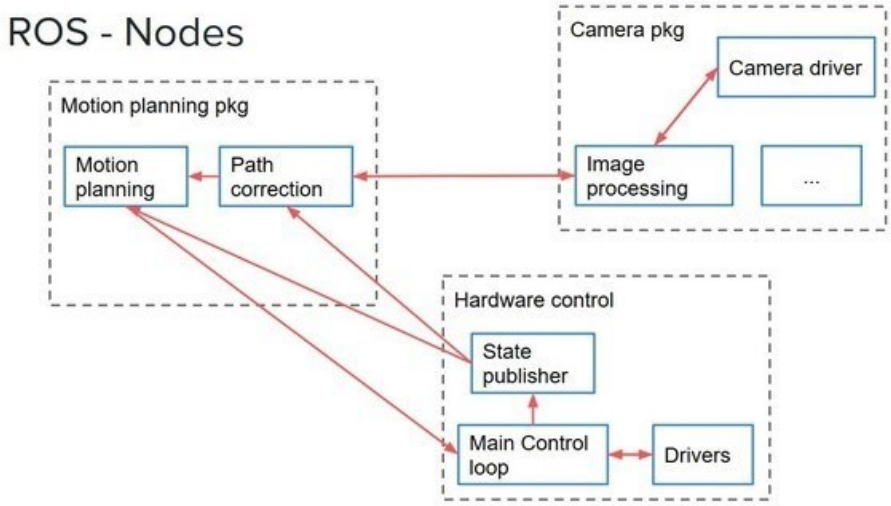
Para conseguir integrar todos os módulos necessários para realizar o processo de SLAM é de extrema importância ter uma ferramenta que facilite este processo, um framework que ficou popular para desenvolvedores robóticos é o ROS. Este foi desenvolvido para ajudar na construção de aplicações para robôs, ideal para projetos que envolvem modularidade por possuir uma estrutura simples para integração de diversos módulos. Conta com diversos pacotes que abstraem complexidade por possuir diversas funções que realizam controle de mo-

tor, leitura de sensores, ferramentas gráficas para monitoramento e possibilitar a criação de um código limpo devido a estruturação.

Este framework funciona com base em um serviço Mestre (Roscore), o serviço mestre aloca um IP que será utilizado pela rede, isso é necessário pois integrantes da rede irão precisar trocar mensagens entre eles, então a função do Mestre é garantir que todos recebam as mensagens. Os nós são programas desenvolvidos em linguagem de programação para realizar determinadas funções normalmente relacionadas ao controle de algum sensor. Quando o dado de um nó é pertinente para outro eles se comunicam por um sistema de mensageria denominado tópicos, a mensageria é composta por nós publicadores e ouvintes, um nó que precisa de informações de outro é um ouvinte, em quanto o nó que fornece as informações é chamado de publicador. Os nós são organizados em estruturas chamadas pacotes, um pacote é constituído de diversos nós que cabem ao propósito empregado, este também é responsável por definir todas as dependências de bibliotecas dos nós que o compõe. Também é possível utilizar serviços, neste framework os nós também suportam uma comunicação de cliente/servidor, que diferentemente dos tópicos, devem ser acessados por meio de uma requisição, está sempre retornará o sucesso ou fracasso na requisição (interface) que pode ser ocasionado por tipagem incorreta ou permissão negada. A estrutura citada pode ser observada pela exemplicação da figura 6.

Este framework conta com o sistema operacional Ubiquity (Ubiquity_Robotics, 2018) exclusivo para sua utilização, diversas versões estão disponíveis, uma distribuição exclusiva é feita para Raspberry Pi, a versão do framework é composta sempre pela versão equivalente do sistema operacional Ubuntu, isto porque este é utilizado como base para o Ubiquity.

Figura 6 – Exemplificação de um pacote constituído de nós realizando troca de mensagens.



Fonte: RoboticsBackend, 2020.

2.8 SSH (SECURE SHELL)

A Raspberry por mais que conte com um processamento suficiente para comportar um sistema operacional e módulos conectados à mesma, pode sofrer com as várias leituras realizadas pelo modulo de câmera, uma forma de solucionar este problema é a conexão a uma outra rede ROS que fique responsável pelo processamento massivo dos dados, está conexão pode ser feita via SSH, este protocolo de aplicação permite acessar, administrar e alterar configurações dá máquina conectada. Todo este processo ocorre de maneira segura via criptografia gerada pelo protocolo, todo acesso é realizado via terminal como o nome sugere *"Shell"*, todo processo padrão para conexão SSH foi realizada conforme especificado por (Barret et.al, 2001).

2.9 VIRTUAL BOX

Pode-se criar novas máquinas ROS com um processamento elevado utilizando virtualização em computadores, a virtualização consiste em criar máquinas virtuais com qualquer sistema operacional disponível a partir do software, a máquina virtual permite configurar o quanto será dedicado às imagens criadas, isto é processamento de CPU (Unidade Central de Processamento), memória, GPU (Unidade de Processamento Gráfico). Estas configurações apresentam determinados limites relacionados a máquina hospedeira. Também é possível definir o tipo de rede que será utilizada, podendo fazer parte da rede local, ou de configurações customizadas.

É adequado utilizar a virtualização ao invés de máquinas reais, pois utilizando as imagens em software tem-se controle sobre maior parte das execuções do sistema operacional, com isto é possível direcionar quanto recurso estiver disponível e for necessário ao ROS, um software que permite criar as máquinas virtuais com muitas possibilidades de configuração é o Oracle VirtualBox (Oracle, 2018), sendo necessário apenas a imagem do sistema operacional desejado.

Este software conta como uma interface de configuração, configurações padrões são encontradas com base na configuração dá máquina atual. Ao configurar uma máquina de uma forma customizada o software indica quando é selecionada uma configuração inválida.

2.10 PADRÕES DE PROJETO

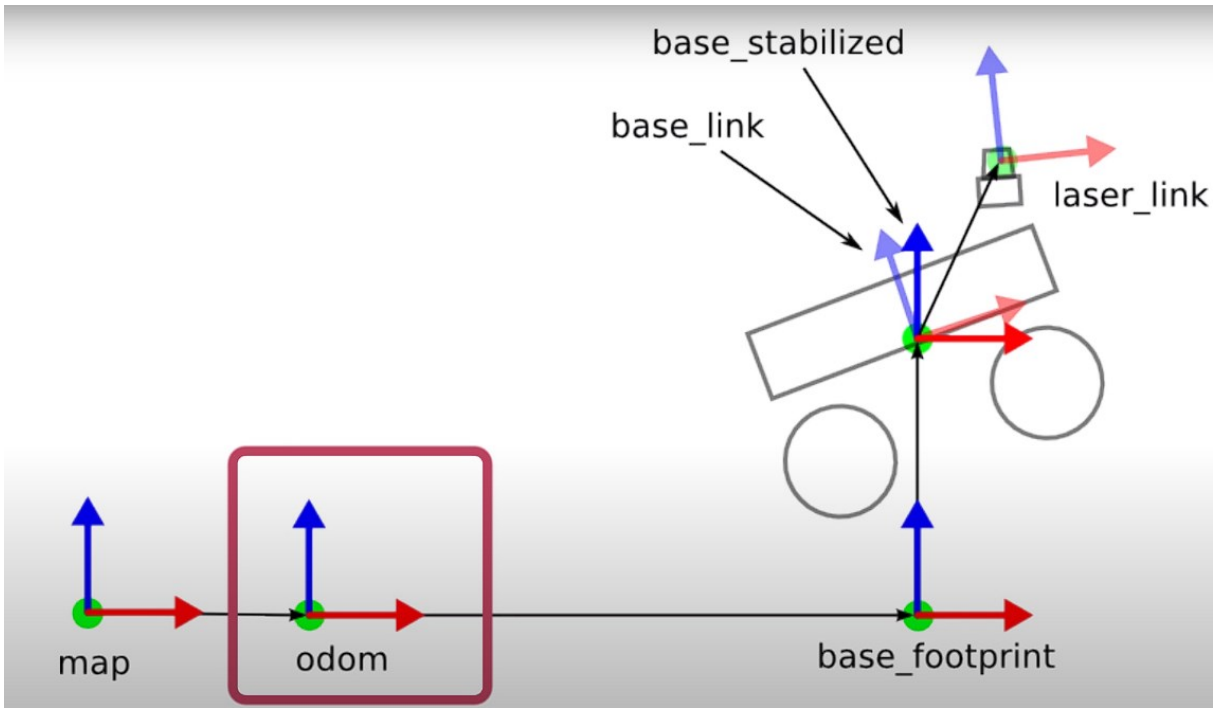
Ao desenvolver um projeto escalonável e modular é comum para organização do projeto utilizar algum tipo de padrão, o framework ROS proporciona certa organização devido a estrutura de pacotes, contudo para o desenvolvimento dos nós e definição da quantidade de pacotes e seu propósito ainda é necessário algum padrão. Design Patterns (Padrões de Projeto) são soluções genéricas para o desenvolvimento de um projeto dependendo do problema a solucionar. Em 1994 quatro engenheiros de software escreveram o livro mais popular sobre o assunto "Design Patterns: Elements of Reusable Object-Oriented Software" (GAMMA et.al 1994) este livro definiu 23 padrões para problemas que podem ocorrer durante o desenvolvimento de um projeto. É comum encontrar um padrão dentre os 23 que se encaixe no desenvolvimento de um projeto, sua utilização apresenta ganho de produtividade, legibilidade de código e fácil manutenção de código.

2.11 FRAME

Para criação de um mapa e utilização dele para navegação de um robô é importante compreender como os objetos incluindo o próprio autônomo estão dispostos, no framework ROS que conta com ferramentas para auxiliar neste processo. Um autônomo normalmente é disposto de vários módulos, como garras expansivas, câmera, motores de movimentação e angulação conforme definido em "*Crucial Components of an Autonomous Robot*" por (Walker, 2020). Em um mapa é imprescindível entender que o autônomo será um objeto dentro

deste e terá uma posição relativa ao mapa, assim como os módulos que o compõe. No framework há uma nomenclatura específica para esses, sendo chamados de frames, cada frame sempre terá sua relação com o mapa ao utilizar as bibliotecas específicas do framework, as informações disponibilizadas por um frame normalmente são referentes ao posicionamento destes no mapa, contendo orientação e posição. O mapa é um plano cartesiano isto quer dizer que toda informação fornecida pode ser relacionada por uma relação de eixos x e y, o posicionamento de um robô é equivalente a um ponto de relação destes eixos. Podemos observar na figura 7 a disposição de alguns frames de um robô em relação ao mapa.

Figura 7 – Exemplificação de Frames de um robô.



Fonte: Kohlbrecher, 2014.

2.12 TF (TRANSFORMAÇÕES)

Quando um frame tem a possibilidade de alterar sua posição no plano cartesiano os utilitários do framework precisam saber que tipo de movimento foi realizado para que assim possam atualizar as bases de cálculos e a visualização em simuladores que comumente são utilizados. As transformações são uma árvore de visualização que indicam os relacionamentos dos frames, quando um evento de translação ou rotação é realizado por determinado frame a transformação deve ser publicada, com isto os assinantes podem ser notificados do evento e atualizar suas referências.

2.13 NUVEM DE PONTOS

Nuvem de pontos é a técnica de recriar objetos a partir de uma grande coleção de pontos, estes são identificados a partir da aglomeração dos pontos, normalmente uma nuvem pode ser obtida por captura de sensor a laser, contudo uma imagem também pode ser utilizada para a obtenção. Esta técnica comumente apresenta perda de dados, porém dependendo do tamanho na nuvem e da técnica utiliza não é necessário nenhum tipo de recuperação, isto devido ao fato da nuvem ser tão densa que se torna irrelevante a perca de alguns pontos.

2.14 SENSOR LASER

Sensores a laser são comumente utilizados para obtenção de distância a objetos, esse tipo de sensor emite um feixe de luz em linha reta, quando o feixe é refletido por um objeto o pulso retorna ao sensor. A medição de distância é feita pelo tempo de voo do laser até o seu retorno, normalmente um sensor deste tipo apresentar alta precisão.

2.15 ODOMETRIA

Odometria é uma técnica usada para medir a distância percorrida, a utilização desta técnica se torna importante para o mapeamento pois para obter a localização de um robô é necessário saber qual foi o deslocamento realizado a partir de um ponto x de origem. Pode também ser utilizada para determinar a localização do robô em relação ao ambiente através do acompanhamento dos movimentos das rodas. A técnica de Odometria está sujeita a erros conforme a distância percorrida aumenta, por isso técnicas de recuperação de erro normalmente são empregadas custando um certo processamento.

2.16 VISO2 ROS

A biblioteca Viso2_ROS (LLUIS, 2015) é uma das mais atualizadas para a busca de Odometria com base em algoritmos visuais, pode ser utilizada por meio de câmeras estéreo (onde é utilizado múltiplos monóculos) ou câmeras

monoculares. Este utilitário supera o fator de escala desconhecido pela utilização de apenas um monóculo partindo do princípio de uma transformação fixa sobre o solo em que a câmera está posicionada. Cada ação sobre o plano deve ser publicada para que o algoritmo funcione de maneira correta.

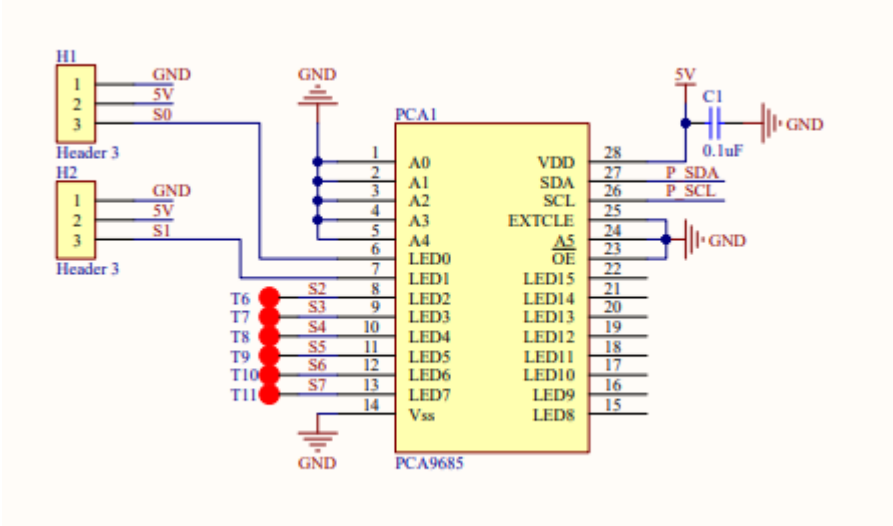
3 METODOLOGIA

3.1 HARDWARE

3.1.1 Conexão Servo Motor

Os servos motores utilizados para angulação da câmera foram conectados utilizando o módulo PCA9685, os 16 canais deste permitem a conexão de múltiplos servos. A placa do AlphaBot2 disponibiliza duas entradas H1 e H2 conforme o esquemático da figura 8, estas contam com pinos para alimentação e aterramento, além disso o pino 3 é disponibilizado para os sinais PWM para o controle dos servos.

Figura 8 – Disposição das conexões do módulo PCA9685.

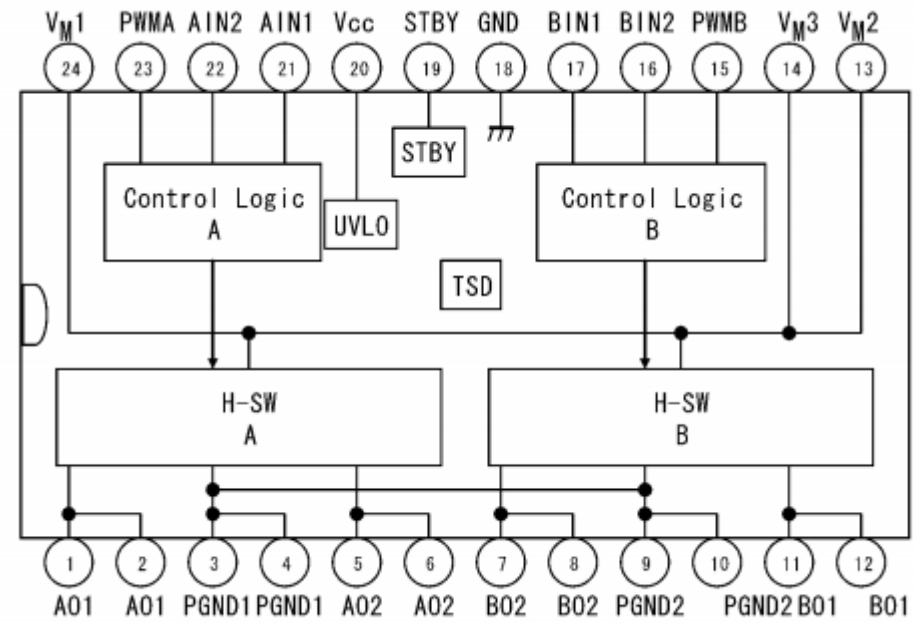


Fonte: Waveshare, 2017.

3.1.2 Conexão Motor Elétrico

Os dois motores de corrente continua utilizados para a movimentação do robô foram controlados a partir do circuito integrado TB6612FNG, este permite controlar até dois motores. O controle é feito por PWM por meio de dois canais de entrada AIN2 E AIN1 conforme a figura 9, neste caso o controle por meio de pulso é realizado para controlar a aceleração dos motores, por ter a presença de dois canais a aceleração pode ser feita de maneira distinta em cada um dos motores, isto possibilita acelerar com intensidades diferentes cada um deles. Neste circuito também encontramos o terminal do tipo VMOT, este é utilizado para aplicar a tensão necessária para o motor funcionar.

Figura 9 – Disposição dos pinos do circuito integrado TB6612FNG.

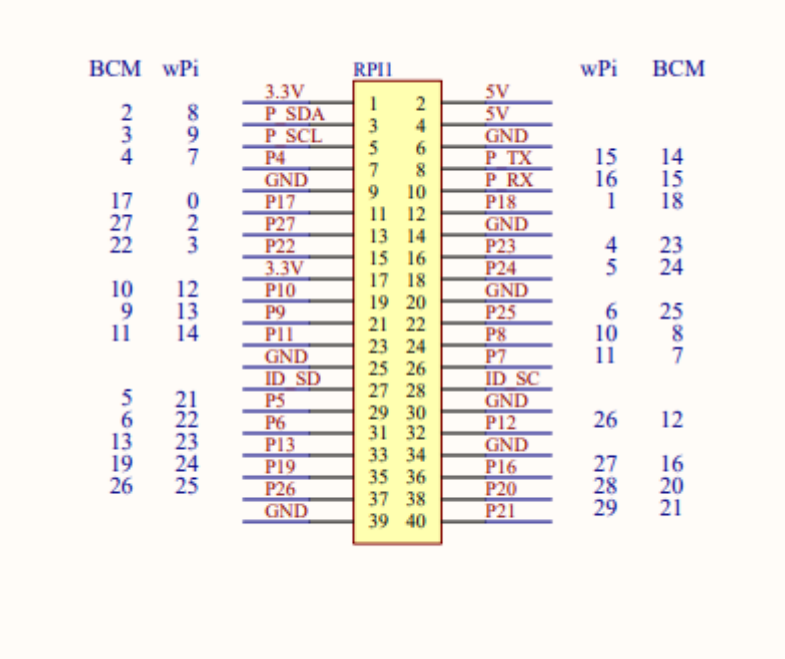


Fonte: Waveshare, 2017.

3.1.3 Conexão Raspberry

A conexão entre a Raspberry e o Alphabot2 é feita através de todos os pinos de propósito geral do micro computador, estes antes disponíveis para utilização do usuário são utilizados para os controles gerais do robô como servo motor, motor elétrico, leds e sensor ultrassônico. A alimentação do robô pode ser feita através das baterias, contudo pela conexão provinda dos pinos de alimentação 1, 2 e 4 como demonstrado na figura 10 este pode ser ligado pela conexão usb do computador.

Figura 10 – Pinos de conexão entre o robô e o computador.

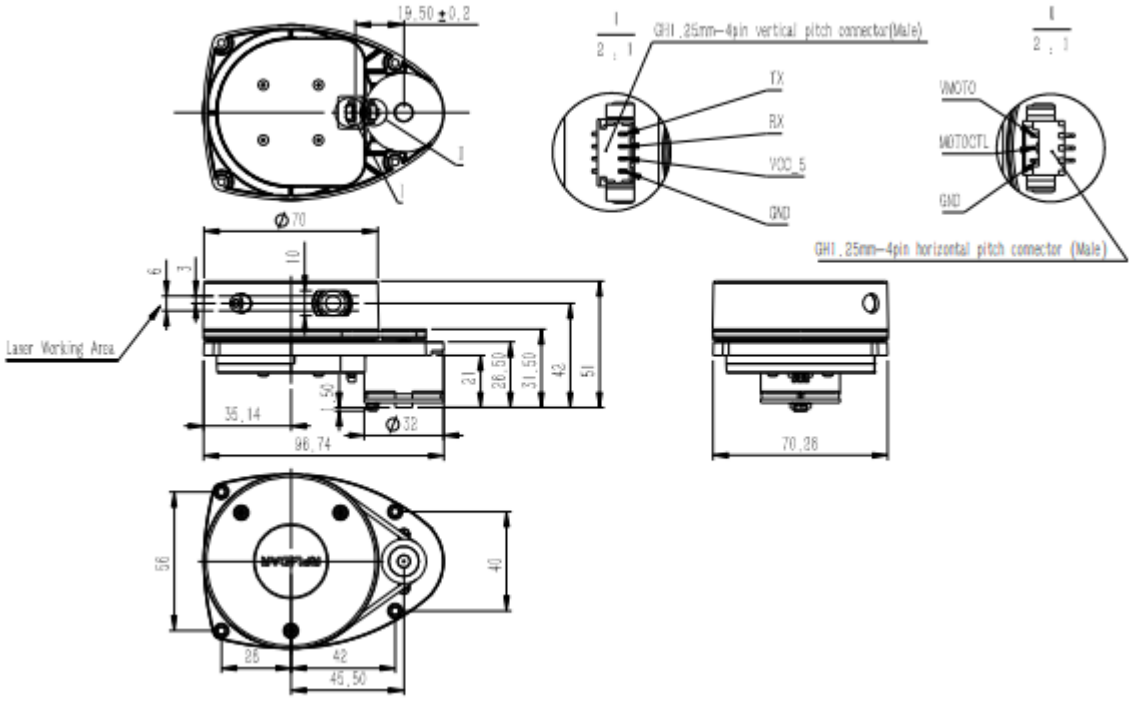


Fonte: Waveshare, 2017.

3.1.4 Conexão RPLidar

O módulo laser utilizado no projeto é um RPLidar, este conta com rotação de motor para varrer o ambiente tem 360°, sua utilização se deve ao fato da existência de bibliotecas do framework próprias para sua utilização. Este é conectado um adaptador usb, as entradas que fazem essa ponte são a alimentação 5V, aterramento, pino de transmissão de dados e recepção. O diagrama pode ser observado na figura 11.

Figura 11 – Diagrama RPLidar.



Fonte: Slamtec, 2016.

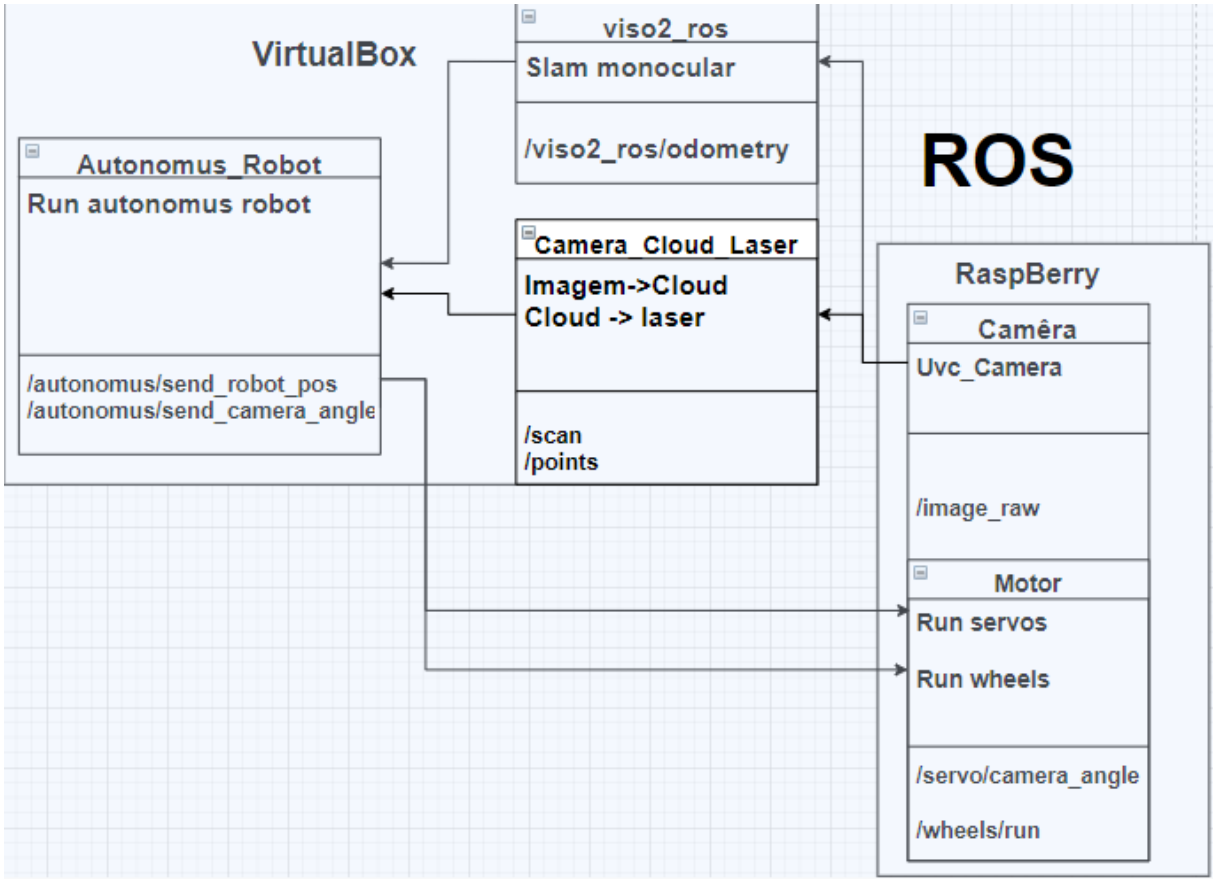
3.2 SOFTWARE

3.2.1 Definição Estrutura Projeto

O padrão de projeto definido foi o *Facade*, este tem como base ajudar em desenvolvimentos onde um conjunto amplo de objetos precisa funcionar eficientemente, as definições utilizadas seguiram o capítulo de “Padrões de projeto Estrutural” do livro “Mergulho nos padrões de projeto” (SHVETS, 2018). A estrutura dos pacotes do framework ajuda a construir neste tipo de padrão, o modelo desenhado teve como objetivo distribuir as responsabilidades a cada uma das

fachadas, chegando até o corpo maior que seria o algoritmo de navegação autônomo:

Figura 12 – Estrutura do projeto.



Fonte – Autor.

3.2.2 Configuração Máquina Virtual

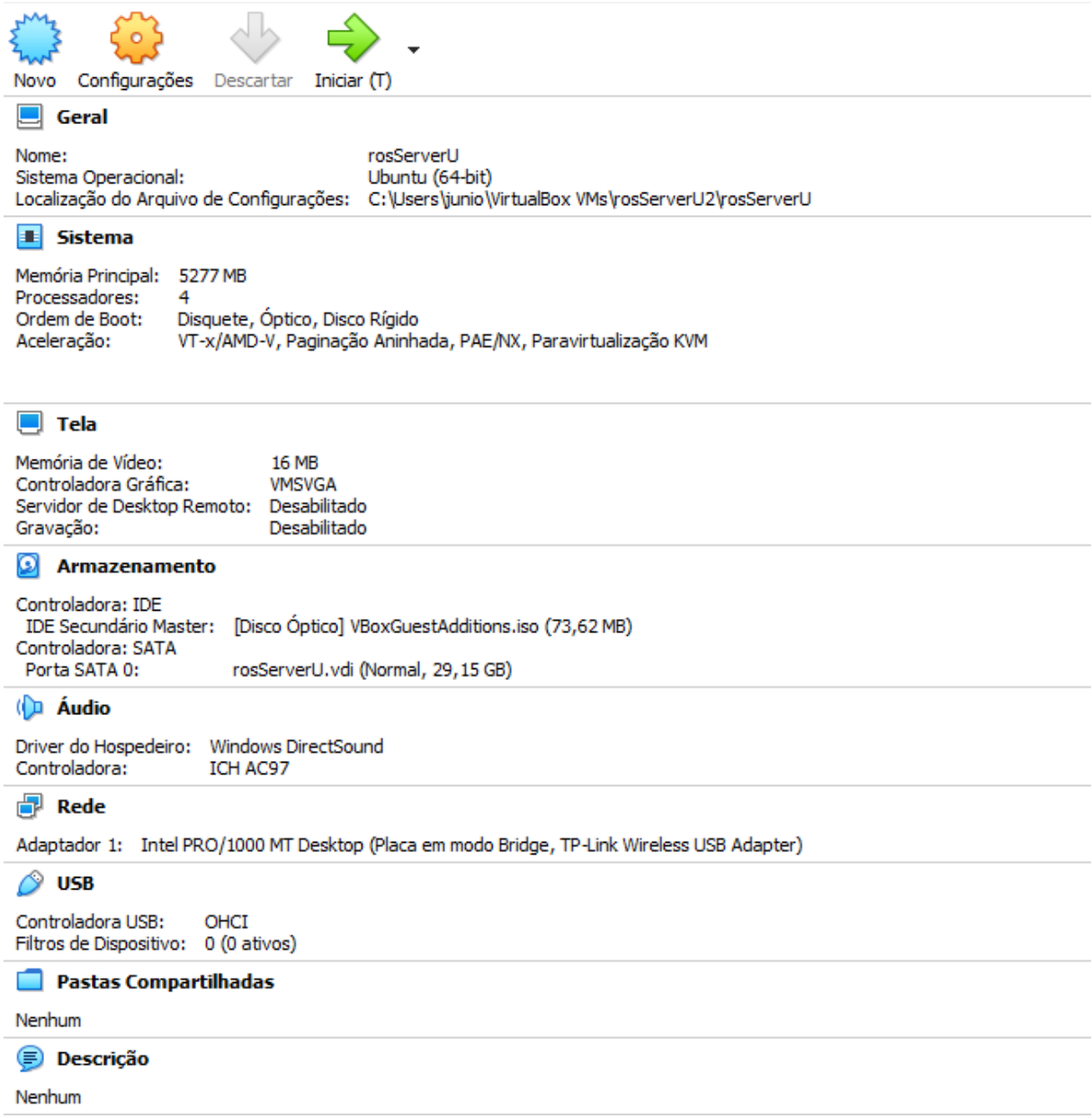
A imagem dá máquina virtual definida foi utilizando sistema operacional Ubuntu, esta versão deve corresponder a utilizada na Raspberry para facilitar o

processo de comunicação, a versão correspondente a imagem Ubiquity é a Bionic Beaver (UBUNTU, 2018).

As configurações realizadas foram selecionar a imagem Ubuntu, definir uma quantidade suficiente para o processamento das operações, que neste caso significou utilizar toda a memória disponível para a máquina virtual (5.2 GB).

Por fim selecionar uma placa de rede em modo Bridge, pois assim a máquina virtual alocaria um IP de rede para imagem possibilitando conexão SSH com a Raspberry, as demais configurações foram definidas pelo próprio software e pode ser visualizado na figura 13.

Figura 13 – Configuração da máquina virtual.


Novo
Configurações
Descartar
Iniciar (T)
Geral
Nome: rosServerU
Sistema Operacional: Ubuntu (64-bit)
Localização do Arquivo de Configurações: C:\Users\junio\VirtualBox VMs\rosServerU2\rosServerU
Sistema
Memória Principal: 5277 MB
Processadores: 4
Ordem de Boot: Disquete, Óptico, Disco Rígido
Aceleração: VT-x/AMD-V, Paginação Aninhada, PAE/NX, Paravirtualização KVM
Tela
Memória de Vídeo: 16 MB
Controladora Gráfica: VMSVGA
Servidor de Desktop Remoto: Desabilitado
Gravação: Desabilitado
Armazenamento
Controladora: IDE
IDE Secundário Master: [Disco Óptico] VBoxGuestAdditions.iso (73,62 MB)
Controladora: SATA
Porta SATA 0: rosServerU.vdi (Normal, 29,15 GB)
Áudio
Driver do Hospedeiro: Windows DirectSound
Controladora: ICH AC97
Rede
Adaptador 1: Intel PRO/1000 MT Desktop (Placa em modo Bridge, TP-Link Wireless USB Adapter)
USB
Controladora USB: OHCI
Filtros de Dispositivo: 0 (0 ativos)
Pastas Compartilhadas
Nenhum
Descrição
Nenhum

Fonte – Autor.

3.2.3 Configuração do Framework ROS

O framework ROS não pode ser obtido por executável, a forma disponibilizada pelos mantedores é utilizando comandos de terminal, são disponíveis algumas versões destes, isto porque algumas contemplam todo ambiente de trabalho e bibliotecas, em quanto outras a versão minimalista para a rede funcionar. Por mais interessante que seja utilizar uma versão com recursos desnecessários a versão instalada foi a completa devido a utilização da máquina virtual.

- **Passo 1:** Adicionar permissão para pacotes da organização ROS:

sudo sh -c 'echo "deb packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

- **Passo 2:** Com a permissão concedida ao sistema adicionar as chaves para obtenção do pacote:

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

- **Passo 3:** O comando update deve ser feito para ser buscado as novas versões:

sudo apt update

- **Passo 4:** Comando para instalação do framework completo:

sudo apt install ros-”version”-desktop-full

3.2.4 Utilização VISO2_ROS

Este pacote é o utilizado para obtenção da Odometria através de imagens retificadas, a retificação é necessária para que a convergência das lentes não afete o resultado de saída do algoritmo. De maneira infortuna este não é disponibilizado por padrão no conjunto do ROS, isto obriga o build manual para obtenção dos arquivos compilados para utilização. Algumas dependências são necessárias, porém todas ficam disponíveis através da instalação do ROS, então este passo não deve ter sido realizado antes da obtenção do framework. O código fonte pode ser obtido através do GitHub viso2_ros (LLUIS, 2015).

Na pasta raiz do projeto buscado do GitHub:

```
> catkin build
```

3.2.5 Conexão Distribuída

A conexão SSH normalmente é um recurso já disponível nos sistemas operacionais Unix, para realizar o acesso bastar ter dois dispositivos em uma mesma rede, tendo este requisito a conexão pode ser feita via terminal pelo seguinte comando:

Para conexão com um outro dispositivo:

```
> ssh {user}@{dispositivo};
```

Também é necessário que ao utilizar o serviço do ROS dois dispositivos tenham o mesmo endereçamento de IP, este framework contém variáveis de ambiente para isto, podem ser configuradas da seguinte forma:

Os dois dispositivos precisam exportar seu IP nesta variável de ambiente:
> $ export ROS_IP = “IPv4 da rede”;

Os dois dispositivos devem exportar um IP com porta para o serviço mestre, neste caso a raspberry utiliza o IP da máquina virtual visto que ela realizara o gerenciamento da rede:
> $ export ROS_MASTER_URI = “http:// IPv4 da rede:PortalLocalHost”;

3.2.6 Obtenção de distanciamento do ambiente

A obtenção de distâncias em relação aos objetos presentes no ambiente é feita a partir da aglomeração da nuvem de pontos. Uma nuvem pode ser obtida através de leitura a laser ou imagem, neste projeto pela utilização de um monóculo a nuvem foi recuperada das imagens provindas da câmera do robô. A obtenção a partir de imagem é obtida através das superfícies visíveis da área observada. Assim como uma nuvem pode ser obtida através de uma leitura a laser o processo reverso pode ser realizado, isso deve ao fato de conseguirmos definir as aglomerações de pontos como um objeto de distância conhecida.

3.2.7 Obtenção do Mapa Ambiente

Um mapa ambiente pode ser obtido através da varredura de um ambiente, os métodos disponíveis pelo framework se utilizam de uma estrutura do conjunto de várias leituras de sensor a laser, por meio deste as linhas do mapa podem ser traçadas. Outro requisito são as transformações, para geração do mapa é necessário saber em que ponto do plano cartesiano o robô se encontra, está informação é necessária para que as linhas sejam traçadas nos lugares corretos. Normalmente o frame utilizado para este propósito é o de Odometria em relação ao frame do Mapa, isto se deve pelo fato do frame de Odometria conter as informações de deslocamento do robô.

4 RESULTADOS E DISCUSSÃO

O primeiro passo realizado foi capturar imagens por meio da câmera do robô, a utilização do framework fornece diversos recursos, o adotado neste projeto foi a utilização do pacote uvc_camera (WALTER, 2015), as imagens capturadas em tempo real tinham as dimensões de 320x240 pixels conforme a figura 14.

Figura 14 – Captura de imagens com Framework.

Fonte – Autor.

Por meio dos comandos de terminal disponíveis para verificação dos estados dos tópicos foi possível validar que as imagens do robô estavam sendo transmitidas pelo mestre da rede por meio da conexão SSH realizada conforme a figura 15.

Figura 15 – Verificação dos Tópicos.

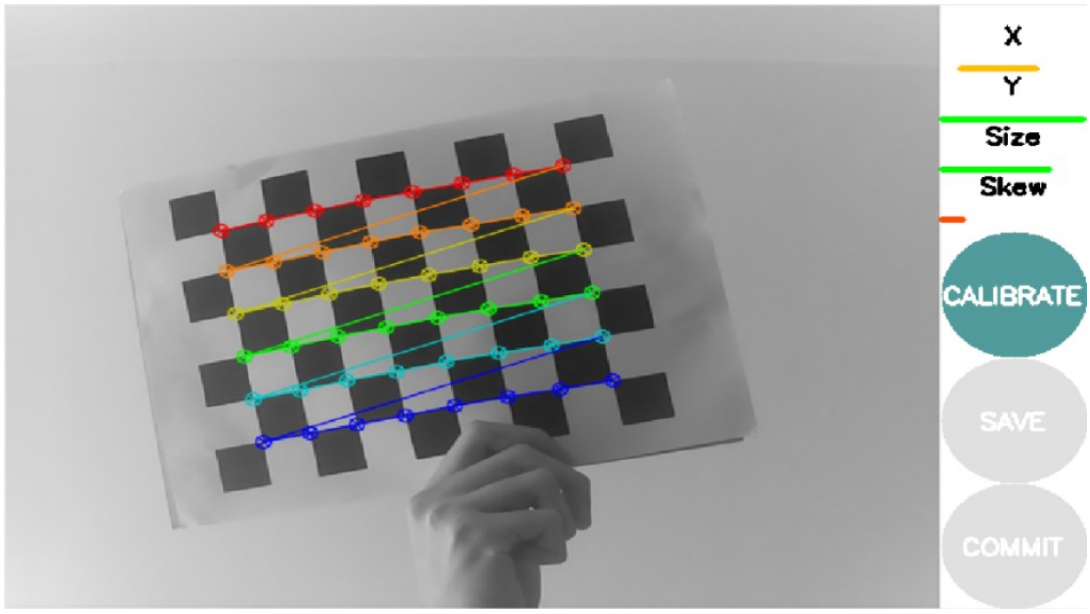
```
root@rosu-VirtualBox:~/catkin_ws/src/sync/src# rostopic list
/TCC/camera_info
/TCC/image_raw
/TCC/image_raw/compressed
/TCC/image_raw/compressed/parameter_descriptions
/TCC/image_raw/compressed/parameter_updates
```

Fonte – Autor.

A partir da captura das imagens foi possível perceber de maneira visual que existia certa discrepância principalmente relativo ao distanciamento dos objetos dá lente da câmera, por este motivo foi realizado a calibração dá câmera. Foi utilizado o pacote camera_calibration (RABAUD, 2015), por meio da captura de imagens de um tabuleiro de damas demonstrado na figura 16 o algoritmo

utilizado pode determinar o coeficiente de distorção das imagens e os armazenar em variáveis num arquivo YAML.

Figura 16 – Calibração da câmera por meio de um tabuleiro de damas.



Fonte – Autor.

Como observado foi necessário a movimentação do tabuleiro nos eixos X e Y, também foi necessário aproximar e distanciar o tabuleiro. Após preencher o percentual necessário para o algoritmo realizar os cálculos, um arquivo foi salvo em disco contendo as variáveis de distorção da câmera conforme mostra a figura 17.

Figura 17 – Arquivo da calibração de câmera.

```
image_width: 640
image_height: 480
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [1009.958391, 0.000000, 381.514611, 0.000000, 1001.762271, 308.466848, 0.000000, 0.000000, 1.000000]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.322086, -0.020206, -0.000168, 0.000276, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [969.467041, 0.000000, 386.307995, 0.000000, 0.000000, 977.001953, 311.044399, 0.000000, 0.000000, 0.000000, 1.000000
0.000000]
```

Fonte – Autor.

O passo seguinte foi a utilização do pacote viso2_ros (LLUIS, 2015) para obtenção da Odometria, porém foram obtidas duas mensagens que impossibilitaram o funcionamento do algoritmo, a primeira conforme a figura 18 indicava que a imagem deveria ser retificada, isto por que o algoritmo consegue determinar o posicionamento a partir do momento em que os pontos das imagens capturadas em tempo real estejam sincronizados, já a segunda conforme a figura 19 indicava que o cabeçalho da imagem e a matriz desta não estavam sincronizadas.

Figura 18 – Mensagem de alerta imagem deve ser retificada.

```
mono_odometer needs rectified input images.
```

Fonte – Autor.

Figura 19 – Mensagem de alerta sincronização.

```
Image messages received:      24
CameraInfo messages received: 93
Synchronized pairs:           0
```

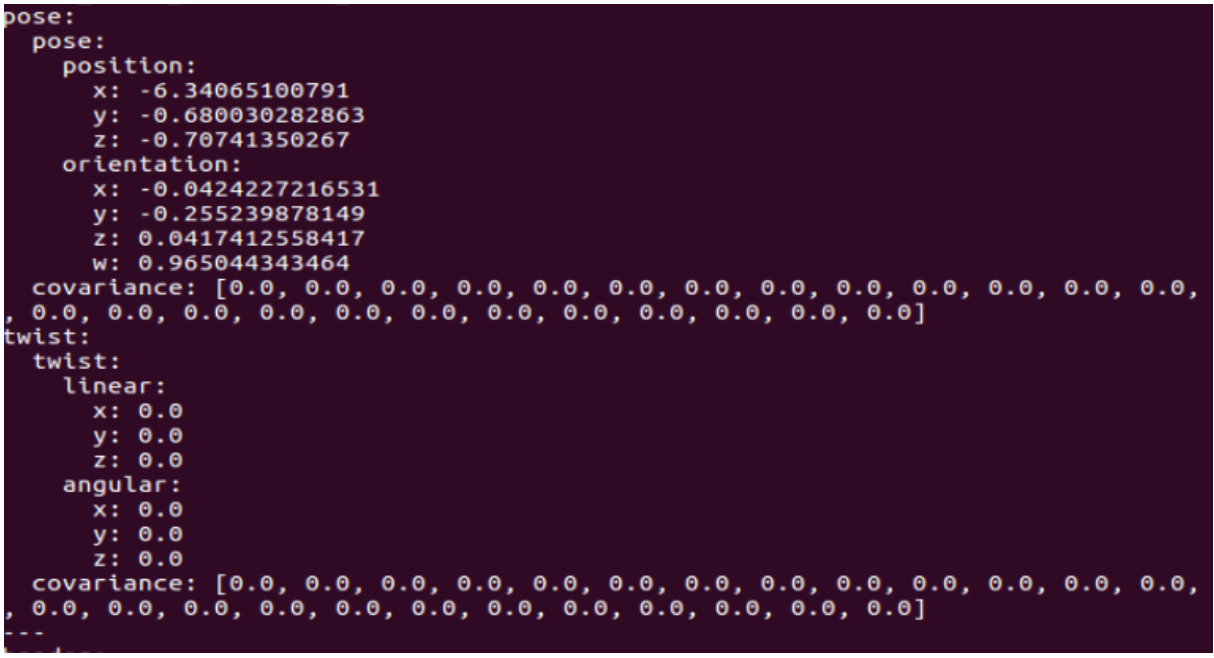
Fonte – Autor.

Para resolução da mensagem sobre a imagem retificada foi utilizado o pacote image_proc (TORRES, 2019) responsável pelo processo de retificação de imagem e publicação em tópico.

No que se refere ao sincronismo foi desenvolvido um código conforme o Anexo A para interceptar todas as mensagens provindas dos tópicos de imagem (`/TCC_cam/image_raw`) e informação da câmera (`/TCC_cam/camera_info`), as mensagens interceptadas foram passadas por um filtro de data e hora, assim permitindo apenas a passagem das imagens sincronizadas com o cabeçalho, também neste trecho do código foi realizada a conversão da imagem de 8 bits para 16 bits, isto porque para gerar uma nuvem de pontos é necessária uma imagem de pelo menos 16 bits.

Os dados obtidos pela biblioteca de Odometria se mostraram incorretos conforme podemos observar na matriz de posição na figura 20, a leitura destes indicavam que o robô estaria abaixo do solo adotando o eixo z como a referência indicativa da altura em relação ao solo, sua orientação em relação ao seu modelo também estava incorreta. Analisando a documentação do pacote foi possível perceber os tópicos descritos comtemplavam especialmente câmeras estéreo, nenhum tipo de tratativa de erro é feito sobre as câmeras monoculares conforme o tópico “3. *Limitations*”(LLUIS, 2015).

Figura 20 – Tópico de odometria sendo publicado.

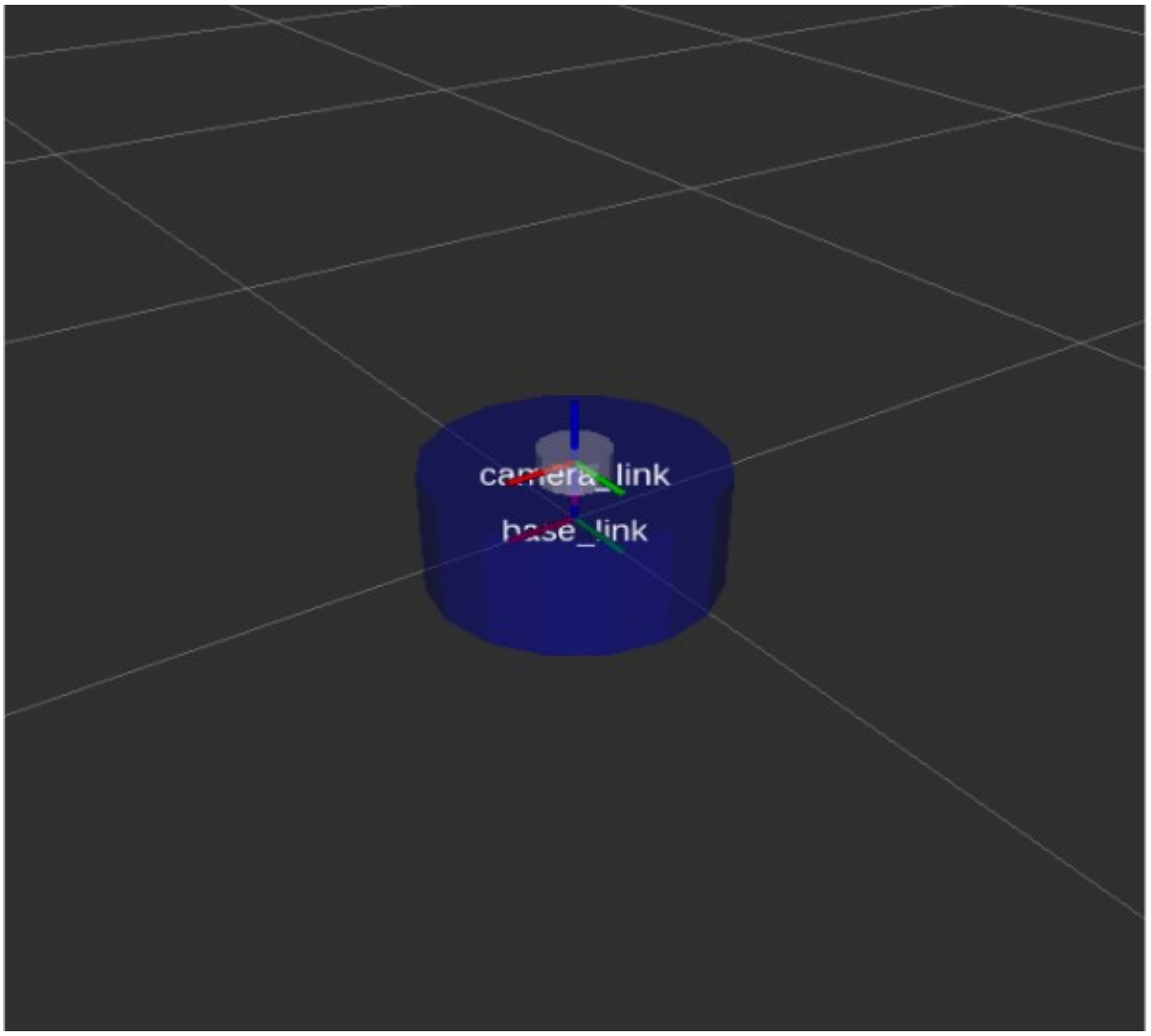


Fonte – Autor.

Utilizando as imagens retificadas pelo filtro foi possível gerar uma nuvem de pontos, para este processo foi utilizado o pacote depth_image_proc (MIHELICH, 2017), a verificação da nuvem de pontos só é possível a partir de uma transformação sobre o frame da câmera (camera_link), por este motivo um modelo do robô utilizado foi desenvolvido conforme o Anexo B seguindo os padrões URDF (HENKEL, 2019).

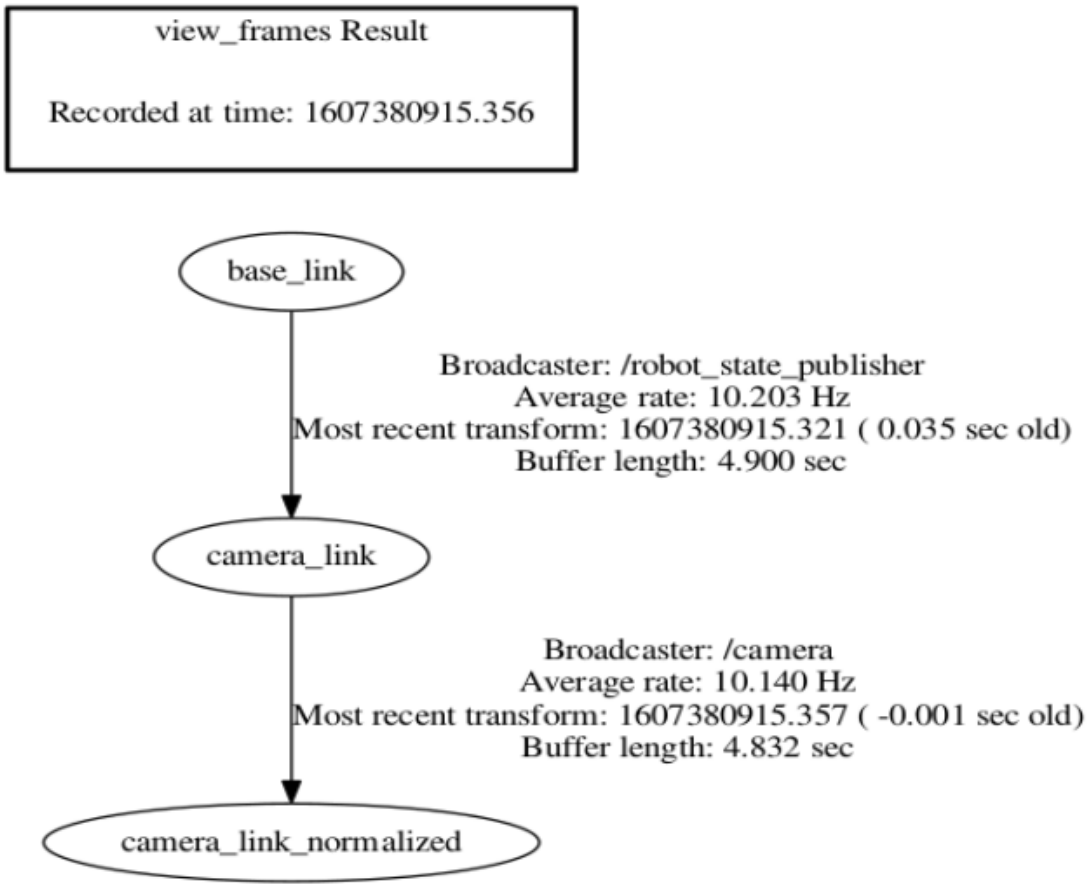
A partir do modelo preparado demonstrado na figura 21 foi possível utilizar o simulador RVIZ (WOODALL, 2018) para visualizar a nuvem de pontos em relação a um plano cartesiano e o robô, contudo a nuvem ficou relacionada sobre o frame relativo ao link da câmera no eixo Z, o desejado seria a nuvem em relação ao eixo X do plano pois este refletiria o ambiente real, para isso foi feito uma transformação no frame relativo (camera_link) para a mudança de eixo (camera_link_normalized) conforme a figura 22, o arquivo contendo está transformação está disponível no Anexo C.

Figura 21 – Visualização modelo do robô.



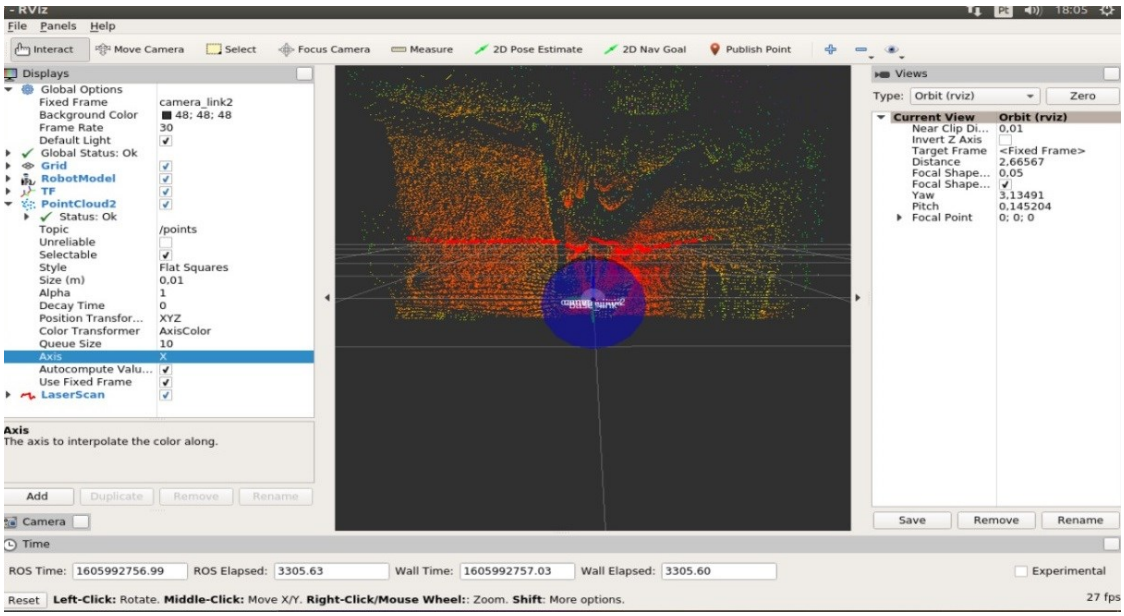
Fonte – Autor.

Figura 22 – Árvore TF do robô.



Fonte – Autor.

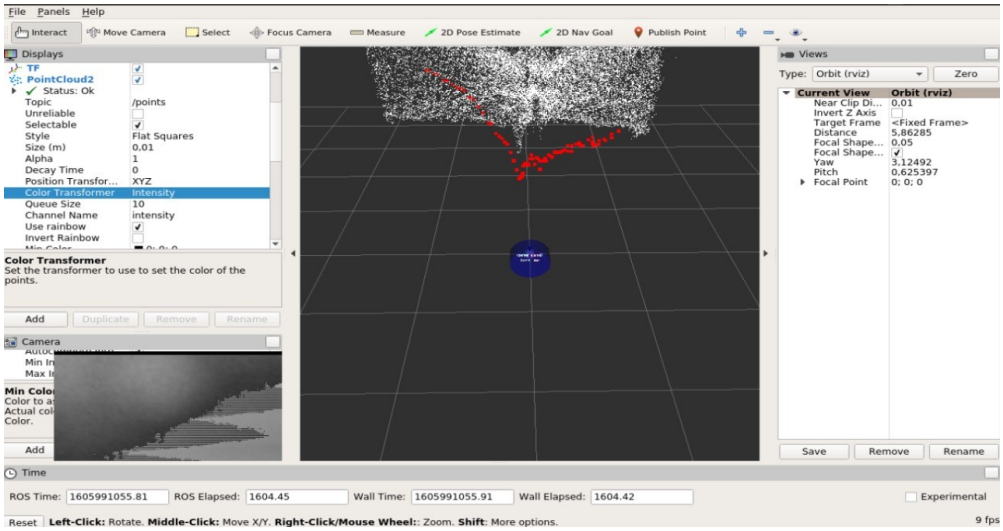
Figura 23 – Visualização nuvem de pontos.



Fonte – Autor.

Como podemos observar na figura 23 a nuvem de pontos pode ser dividida com base em um mapa de calor, os pontos mais próximos têm destaque vermelho, em quanto os mais distantes apresentam destaque em verde, com base nisso é possível converter a nuvem de pontos para dados do tipo sensor_msgs/laser_scan da biblioteca pointcloud_to_laserscan (BOVBEL, 2015), desta forma é possível observar a saída resultante com a nuvem agora em cor branca e os dados do sensor em vermelho conforme mostrado na figura 24.

Figura 24 – Visualização dos dados a partir da nuvem de pontos.

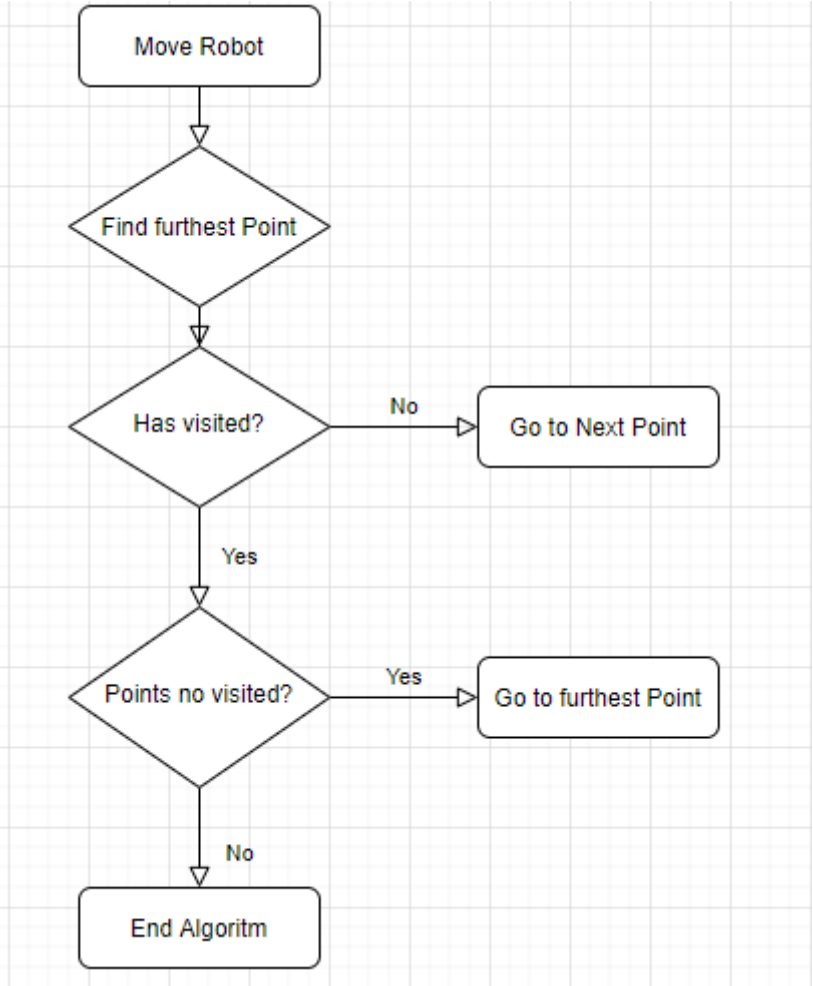


Fonte – Autor.

Com os dados convertidos e sendo possível realizar a leitura da distância do robô para os objetos do ambiente por meio do tópico (/scan) foi possível desenvolver os algoritmos para controle do robô, o primeiro conforme o Anexo D responsável pela angulação da câmera por um servo motor, com isto o laser poderia coletar leituras de pontos distintos, já o segundo conforme o Anexo E ficou responsável pela movimentação do robô.

Por meio do conjunto destes controles e leituras do sensor a laser foi possível desenvolver o algoritmo de navegação autônoma conforme o Anexo F, a ideia central foi a observação dos lados do robô e a sua frente, com base na análise da distância até o próximo objeto era decidido para que ponto o robô deveria seguir. Este tipo de algoritmo poderia ser incrementado com os dados da Odometria, podendo guardar cada uma das posições evitando que em algum momento o robô voltasse até determinado local já visitado conforme o diagrama na figura 25, contudo os dados errôneos não permitiram tal incremento.

Figura 25 – Diagrama em bloco da estrutura de decisão da navegação.



Fonte – Autor.

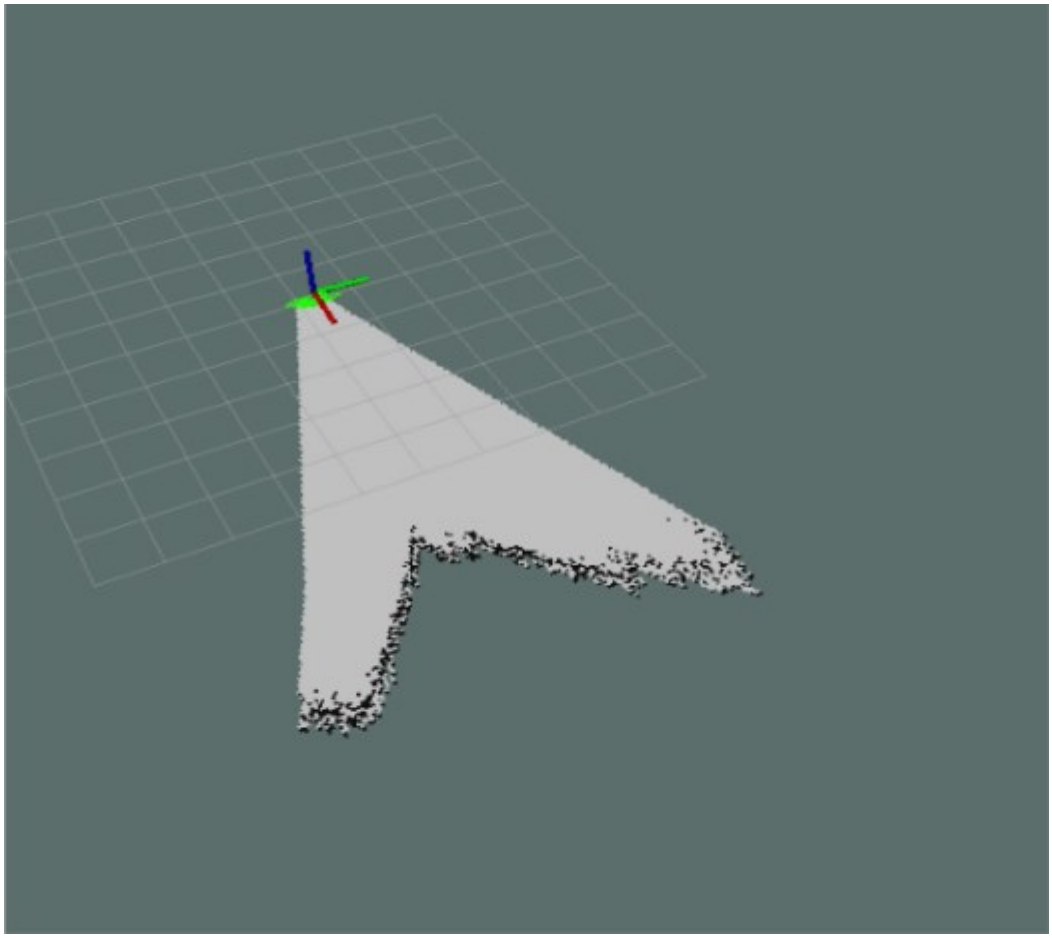
A maior parte dos testes realizados foram em frente a paredes conforme a figura 26, com isto foi possível determinar se o robô conseguiria identificar e determinar corretamente a existência das paredes. Em todos os casos a nuvem de pontos se manteve como na figura 23, porém a distância estimada pela conversão para mensagens sensor_msgs/laser_scan se mostrou imprecisa com as medições reais. Com o robô parado era possível estimar um coeficiente de distorção das mensagens, ao se locomover o dado não podia ser interpretado de nenhuma forma.

Figura 26 – Ambiente de Teste.

Fonte – Autor.

Com relação ao mapeamento as transformações em relação principalmente a Odometria acabaram influenciando na saída final do mapa. A biblioteca utilizada foi a Hector_Slam (Kohlbrecher, 2014), esta acabou se perdendo devido os dados inconsistentes, porém com o robô parado foi possível identificar as linhas de uma parede identificada pelo sensor laser na figura 24, o trecho do mapa recuperado pode ser observado na figura 27.

Figura 27 – Mapeamento monocular.



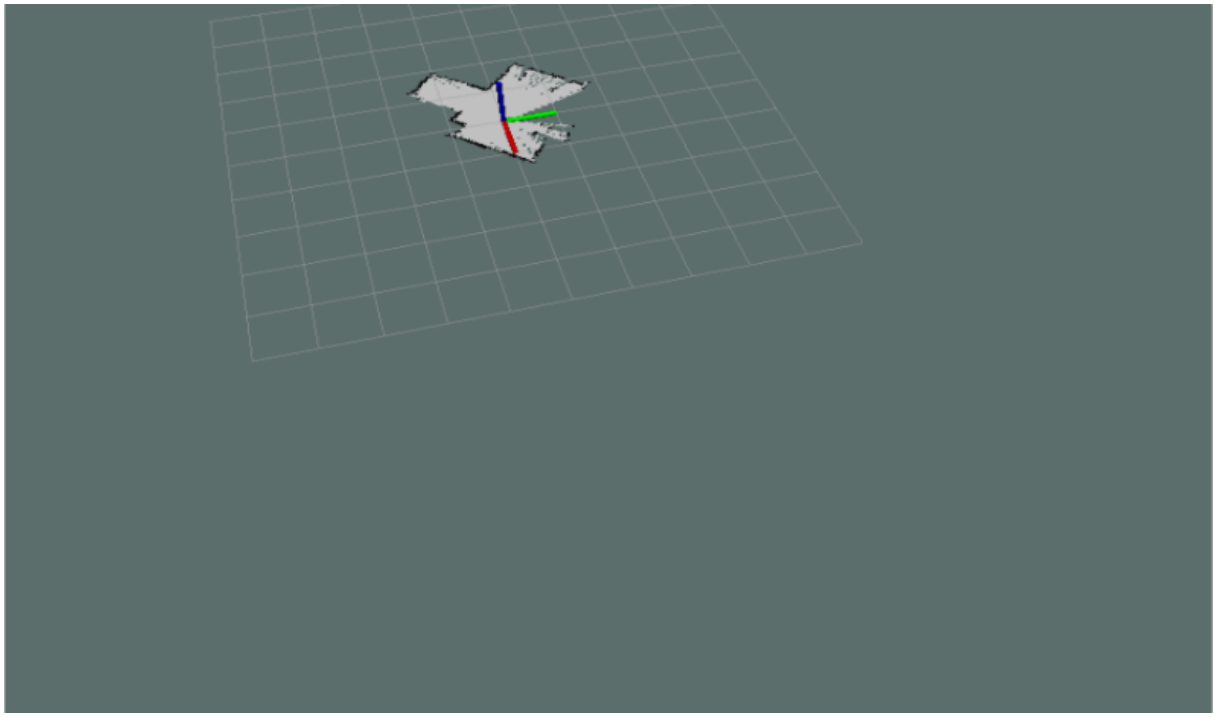
Fonte – Autor.

Os testes realizados com o sensor a laser RPLidar foram feitos de maneira estática sem movimentação com o robô pois a placa não acompanha suporte para este módulo. O ambiente utilizado para o teste se manteve a mesma sala, as leituras realizadas foram precisas, em qualquer posição do ambiente o sensor conseguiu mapear quase toda sala, sendo limitado apenas por algumas quinas. A figura 28 mostra um dos testes, o sensor foi apoiado ao topo do robô. A figura 29 mostra o mapeamento do ambiente realizado pelo sensor ou as linhas escuras representam os objetos detectados, já a figura 30 demonstra as leituras a laser.

Figura 28 – Ambiente teste RPLidar.

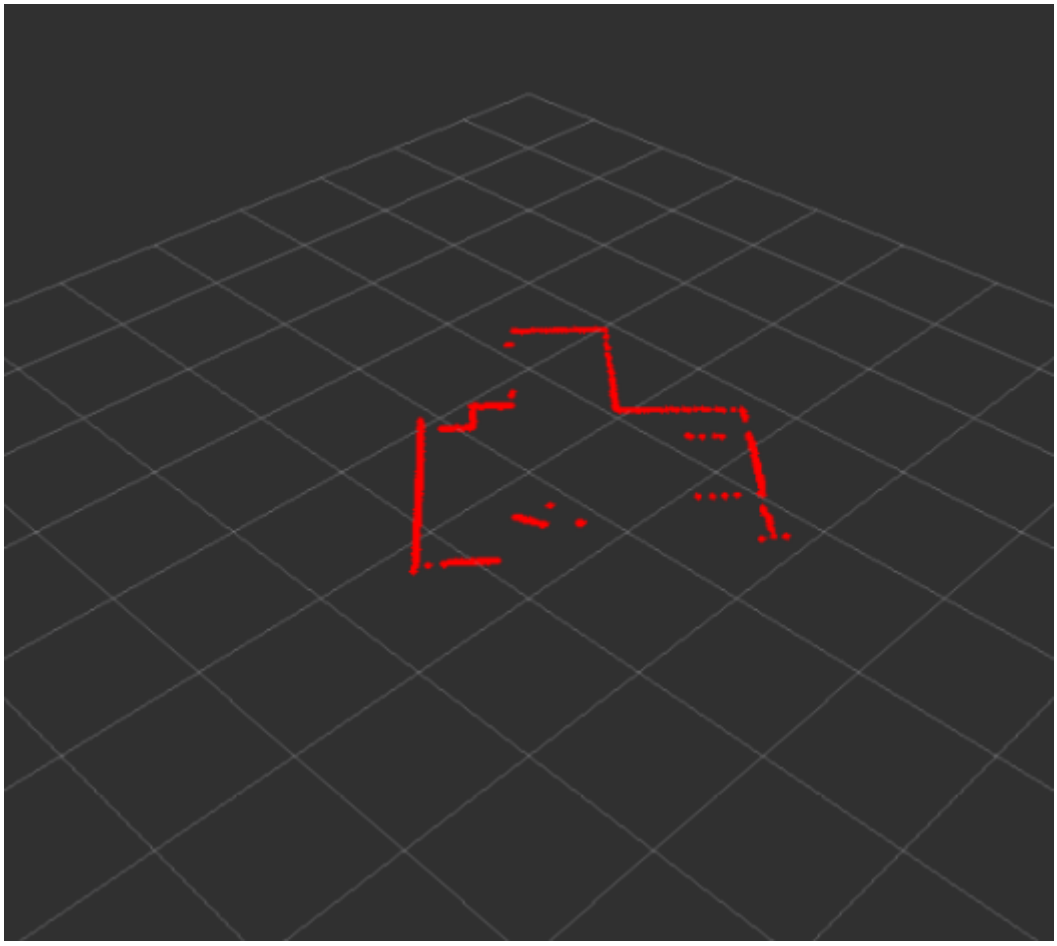
Fonte – Autor.

Figura 29 – Mapeamento RPLidar.



Fonte – Autor.

Figura 30 – Leitura laser RPLidar.



Fonte – Autor.

5 CONCLUSÃO

Todos os objetivos foram executados com sucesso, alguns imprevistos foram encontrados durante o projeto, porém com a utilização de recursos do framework puderam ser superados. O pacote utilizado para Odometria acabou resultando em saídas inutilizáveis, isto não impactou a navegação, porém limitou a possiblidade de algoritmos com melhor otimização e a visualização do mapa do ambiente. No que tange os resultados conforme os demos e as imagens demons-

tradas neste artigo podemos perceber que os dados no geral eram inconsistentes, as medições através do monóculo apresentavam um certo nível de deslocamento, este coeficiente não pode ser calculado devido a ter influência direta do ambiente. Quando com iluminação suficiente as medições se mostraram precisas quando o robô se posicionava em frente a paredes, porém nunca em movimento constante. As medições utilizando um dispositivo nativo laser se mostraram precisas, assim como o mapa apresentado. Podemos concluir que utilizando como base as tecnologias utilizadas neste projeto não é possível realizar a técnica de SLAM de forma efetiva, contudo apesar das várias transformações utilizadas ainda tivemos resultados positivos em determinados pontos do projeto utilizando apenas o monóculo. Em trabalhos futuros seria interessante utilizar algoritmos que apenas com base na entrada da imagem possam realizar o SLAM. Outra possibilidade seria o desenvolvimento deste algoritmo, muito se perderia em relação ao framework, contudo outras bibliotecas poderiam ser utilizadas, com isto o controle dos resultados e tratativa de erros estaria em um nível maior do que encontramos neste projeto.

6 REFERÊNCIAS BIBLIOGRÁFICAS

CAPEK, Karel. **R.U.R (ROSSUM'S UNIVERSAL ROBOTS).** Penguin Books, 2004.

THRUN, S.; BEETZ, M.; BENNEWITZ, M.; BURGARD, W.; CREMERS, A. B.; DELLAERT, F.; FOX, D., HÄHNEL, D.; ROSENBERG, C.; ROY, N.; SCHULTE, J.; SCHULZ, D. **Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva.** Journal of Robotics Research, 2000.

KROMBACH, Nicola; DROESCHEL, Davida; HOUBEN, Sebastian; BEHNKE Sven. **Feature-based Visual Odometry Priorfor Real-time Semi-dense Stereo SLAM.** Robotics and Autonomous Systems, 2018.

BARRET, Daniel; SILVERMAN, Richard; BYRNES, Robert. **SSH The Secure Shell.** O'Reilly, 2001.

GAMMA, Erich; HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. **DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTEED SOFTWARE.** Addison-Wesley Professional, 1994.

SHVETS, Alexander. **Mergulho nos Padrões de Projeto.** Refactoring.Guru, 2018.

GREGERSEN, Erik. **Moore's law**. Disponível em: <https://www.britannica.com/technology/Moores-law>. Último acesso: 28 de março de 2020.

PI_CAMERA_MODULE. **Getting Started with the Camera module**. Disponível em: < https://projects.raspberrypi.org/en/projects/getting-started-with-picamera > Último acesso: 28 de março de 2020.

WAVESHARE. **ALPHABOT2-USER-MANUAL-En**. Disponível em: <https://www.mouser.com/PDFDOCS/ALPHABOT2-USER-MANUAL-EN.PDF> Último acesso: 18 de julho de 2020.

SIQUEIRA, Andressa. **O que é um Raspberry Pi**. Disponível em: <https://medium.com/tht-things-hackers-team/o-que-%C3%A9-um-raspberry-pi-2cd1cd0bdb68> Último acesso: 18 de julho de 2020.

RASPBERRYPI.ORG. **RASPBERRY PI OS (previosly called raspbian)**. Disponível em: <https://www.raspberrypi.org/downloads/raspbian/> Último acesso: 28 de março de 2020.

UBIQUITY_ROBOTICS. **RASPBERRY PI IMAGES.** Disponível em: <https://downloads.ubiquityrobotics.com/pi.html> Último acesso: 28 de março de 2020.

OPENCV. **ABOUT**. Disponível em: <https://opencv.org/about/>. Último acesso: 28 de março de 2020.

SOARES, Daniel. **MOTOR CC.** Disponível em: <http://www.dmcvg.cefetmg.br/wp-content/uploads/sites/66/2017/04/MOTOR_CC.pdf> Último acesso: 28 de março de 2020.

BERTULUCCI, Cristiano Silveira. **Motor CC: Saiba como Funciona e de que forma Especificar.** Disponível em: <https://www.citisystems.com.br/motor-cc/#:~:text=O%20princ%C3%ADpio%20b%C3%A1sico%20de%20funcionamento,giro%20do%20eixo%20do%20motor.> Último acesso: 28 de março de 2020.

ORIENTALMOTOR. **Comparação entre motores de passo: 2 fases e 5 fases.** Disponível em: <https://www.orientalmotor.com.br/tecnologia/comparacao-entre-motores-de-passo-2-fases-e-5-fases.html> Último acesso: 28 de março de 2020.

ROBOTICSBACKEND. **What is a ROS Node?.** Disponível em: <https://roboticsbackend.com/what-is-a-ros-node/> Último acesso: 28 de março de 2020.

ORACLE. Oracle VM VirtualBox. Disponível em: <https://www.oracle.com/br/virtualization/virtualbox/>. Último acesso: 28 de março de 2020.

ROS_KINETIC. **ROS Kinectic kame**. Disponível em: <http://wiki.ros.org/kinetic>. Último acesso: 20 de julho de 2020.

UBUNTU. **Ubuntu Realeases**. Disponível em: <https://releases.ubuntu.com/18.04/>. Último acesso: 20 de julho de 2020.

WALTER, Lucas. **Uvc_Camera**. Disponível em: <http://wiki.ros.org/uvc_camera>. Último acesso: 20 de outubro de 2020.

RABAUD, Vincet. **Image_Proc**. Disponível em: <http://wiki.ros.org/image_proc>. Último acesso: 20 de outubro de 2020.

TORRES, David. **Camera_Calibration**. Disponível em: <http://wiki.ros.org/camera_calibration>. Último acesso: 20 de outubro de 2020.

LLUIS, Pep. **Viso2_ROS**. Disponível em: <http://wiki.ros.org/viso2_ros>. Último acesso: 20 de outubro de 2020.

MIHELICH, Patrick. **Depth_Image_Proc**. Disponível em: <http://docs.ros.org/en/jade/api/depth_image_proc/html/classdepth__image__proc_1_1RegisterNodelet.html>. Último acesso: 20 de outubro de 2020.

HENKEL, Cristian. **Building a Visual Robot Model with URDF from Scratch**. Disponível em: <http://wiki.ros.org/urdf/Tutorials/Building%20a%20Visual%20Robot%20Model%20with%20URDF%20from%20Scratch>. Último acesso: 20 de outubro de 2020.

WOODALL, William. **RVIZ**. Disponível em: <http://wiki.ros.org/rviz>. Último acesso: 20 de outubro de 2020.

BOVBEL, Paul. **Point_Cloud_To_Laser_Scan**. Disponível em: <http://wiki.ros.org/pointcloud_to_laserscan>. Último acesso: 20 de outubro de 2020.

WALKER, Jason. **What Are Autonomous Robots?**. Disponível em: <https://waypointrobotics.com/blog/what-autonomous-robots/>. Último acesso: 20 de outubro de 2020.

KOHLBRECHER, Stefan. Hector_Slam. Disponível em: <http://wiki.ros.org/hector_slam>. Último acesso: 20 de outubro de 2020.

SLAMTEC. RPLIDAR A1. Disponível em: <https://aws.robu.in/wp-content/uploads/2018/10/LD108_SLAMTEC_rplidar_datasheet_A1M8_v2.1_en.pdf >. Último acesso: 20 de outubro de 2020.

7 ANEXOS

ANEXO A - SyncFilterDate

""" Titulo: SyncFilterDate

Autor: Autor do artigo

Data: 2020 """

```
#!/usr/bin/env python

import rospy

import message_filters

from sensor_msgs.msg import Image, CameraInfo

from cv_bridge import CvBridge, CvBridgeError

import cv2

import numpy as np



image_raw_pub = rospy.Publisher('image_raw', Image, queue_size=1)

camera_info_pub = rospy.Publisher('camera_info', CameraInfo, queue_size=1)



global i
```

```
i                                =                                0

bridge                           =                           CvBridge()


def              callback(image_raw,              camera_info):

                                 global                           i

                           i                  +=                  1

          print('got   an   sync   Image   and   CameraInfo',   i)


                stamp              =              rospy.Time.now()


       img_mono8    =    bridge.imgmsg_to_cv2(image_raw,    de-
sired_encoding="mono8")

        img_mono16     =     np.uint16(img_mono8)     *     256


      image_raw   =   bridge.cv2_to_imgmsg(img_mono16,   encod-
ing='mono16')

                 image_raw.height              =              240

                    image_raw.width             =             320

                 image_raw.encoding          =          'mono16'

             image_raw.header.frame_id      =      'camera_link'

                 image_raw.header.stamp          =          stamp

                                 image_raw_pub.publish(image_raw)


             camera_info.header.stamp          =          stamp
```

```
        camera_info.header.frame_id      =      'camera_link'
              camera_info.height             =             240
              camera_info.width              =             320
                          camera_info_pub.publish(camera_info)


rospy.init_node('SyncFilterDate')

image_sub  =  message_filters.Subscriber('/TCC_cam/image_raw',
Image)

info_sub  =  message_filters.Subscriber('/TCC_cam/camera_info',
CameraInfo)

ts  =  message_filters.TimeSynchronizer([image_sub,  info_sub],
10)

ts.registerCallback(callback)

rospy.spin()
```

ANEXO B - AlphabotUrdf

""" Titulo: AlphabotUrdf

Autor: Autor do artigo

Data: 2020 """

```
<?xml                                              version="1.0"?>

<robot                                            name="alphabot">
<material                                            name="blue">
          <color        rgba="0        0        0.8        1"/>
                                                    </material>
                    <material                      name="white">
           <color         rgba="1        1        1        1"/>
                                                    </material>
                    <link                        name="base_link">
                                                        <visual>
                                                      <geometry>
              <cylinder       length="0.2"       radius="0.2"/>
                                                    </geometry>
```

```
                        <material                    name="blue"/>
                                                        </visual>
 </link>
<link                                              name="camera_link">
                                                          <visual>
                                                        <geometry>
              <cylinder      length="0.05"       radius="0.05"/>
                                                        </geometry>
                        <material                    name="white"/>
                                                        </visual>
 </link>
<joint          name="base_to_camera"          type="continuous">
                     <parent                     link="base_link"/>
                    <child                    link="camera_link"/>
             <axis           xyz="0           0           0.010"/>
            <origin           xyz="0           0           0.10"/>
  </joint>
```

ANEXO C – CameraLinkTransformation

""" Titulo: Camera_Link_Transformation

Autor: Autor do artigo

Data: 2020 """

```
<launch>
       <node     pkg="tf"     type="static_transform_publisher"
name="camera"  args="0.0  0  0.0  0.0  -1.57  1.57  /camera_link
/camera_link_normalized                 100"                 />
</launch>
```

ANEXO D – RunServos

""" Titulo: RunServos

Autor: Autor do artigo

Data: 2020 """

```
#!/usr/bin/env                                          python3
import                                                     time
import                                                   rospy
from              std_msgs.msg              import             *
from          adafruit_servokit          import          ServoKit

kit                    =                    ServoKit(channels=16)
```

```
def servosCallback(channel, angle):
    kit.servo[channel].angle = angle

def runServosRoutine(msg):
    print('Posicionando servo', msg.data)
    if msg.data == 0:
        servosCallback(0, 45)
    elif msg.data == 2:
        servosCallback(0, 90)
    elif msg.data == 1:
        servosCallback(0, 135)

def rosNodeServo():
    print('Node Servos Iniciado')
    rospy.init_node('run_servos')
    rospy.Subscriber('slam_manager_servos',Int32, runSer-
vosRoutine)
    rospy.spin()

if __name__=='__main__':
    rosNodeServo()
```

ANEXO E – RunWheels

""" Titulo: RunWheels

Autor: Autor do artigo

Referência: (Ref adicionar alphabot manual)

Data: 2020 """

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import *
import RPi.GPIO as GPIO
import time

class AlphaBot2(object):

    def __init__(self,ain1=12,ain2=13,ena=6,bin1=20,bin2=21,enb=26):
        print('GPIO Rodas configurado com sucesso!')
        self.AIN1 = ain1
        self.AIN2 = ain2
        self.BIN1 = bin1
        self.BIN2 = bin2
```

```
                        self.ENA            =            ena
                        self.ENB            =            enb
                         self.PA             =             20
                         self.PB             =             20

                                        GPIO.setmode(GPIO.BCM)
                                      GPIO.setwarnings(False)
                                   GPIO.setup(self.AIN1,GPIO.OUT)
                                   GPIO.setup(self.AIN2,GPIO.OUT)
                                   GPIO.setup(self.BIN1,GPIO.OUT)
                                   GPIO.setup(self.BIN2,GPIO.OUT)
                                    GPIO.setup(self.ENA,GPIO.OUT)
                                    GPIO.setup(self.ENB,GPIO.OUT)
                  self.PWMA      =      GPIO.PWM(self.ENA,500)
                  self.PWMB      =      GPIO.PWM(self.ENB,500)
                                      self.PWMA.start(self.PA)
                                      self.PWMB.start(self.PB)

                        def                    forward(self):
                           self.PWMA.ChangeDutyCycle(self.PA)
                           self.PWMB.ChangeDutyCycle(self.PB)
                              GPIO.output(self.AIN1,GPIO.LOW)
                             GPIO.output(self.AIN2,GPIO.HIGH)
                              GPIO.output(self.BIN1,GPIO.LOW)
            GPIO.output(self.BIN2,GPIO.HIGH)

  def                                          stop(self):
                              self.PWMA.ChangeDutyCycle(0)
                              self.PWMB.ChangeDutyCycle(0)
                            GPIO.output(self.AIN1,GPIO.LOW)
                            GPIO.output(self.AIN2,GPIO.LOW)
                            GPIO.output(self.BIN1,GPIO.LOW)
                            GPIO.output(self.BIN2,GPIO.LOW)

                                  def         backward(self):
                           self.PWMA.ChangeDutyCycle(self.PA)
                           self.PWMB.ChangeDutyCycle(self.PB)
                             GPIO.output(self.AIN1,GPIO.HIGH)
                              GPIO.output(self.AIN2,GPIO.LOW)
                             GPIO.output(self.BIN1,GPIO.HIGH)
                              GPIO.output(self.BIN2,GPIO.LOW)


                                     def            left(self):
                                self.PWMA.ChangeDutyCycle(30)
                                self.PWMB.ChangeDutyCycle(30)
                              GPIO.output(self.AIN1,GPIO.HIGH)
```

```
                                GPIO.output(self.AIN2,GPIO.LOW)
                                GPIO.output(self.BIN1,GPIO.LOW)
                               GPIO.output(self.BIN2,GPIO.HIGH)


                            def                      right(self):
                                  self.PWMA.ChangeDutyCycle(30)
                                  self.PWMB.ChangeDutyCycle(30)
                                GPIO.output(self.AIN1,GPIO.LOW)
                               GPIO.output(self.AIN2,GPIO.HIGH)
                               GPIO.output(self.BIN1,GPIO.HIGH)
                                GPIO.output(self.BIN2,GPIO.LOW)

                        def                 setPWMA(self,value):
                           self.PA            =           value
                  self.PWMA.ChangeDutyCycle(self.PA)

     def                                    setPWMB(self,value):
                          self.PB            =           value
                            self.PWMB.ChangeDutyCycle(self.PB)

              def       setMotor(self,       left,       right):
                  if((right   >=   0)   and   (right   <=   100)):
                               GPIO.output(self.AIN1,GPIO.HIGH)
                                GPIO.output(self.AIN2,GPIO.LOW)
                               self.PWMA.ChangeDutyCycle(right)
                  elif((right   <   0)   and   (right   >=   -100)):
                                GPIO.output(self.AIN1,GPIO.LOW)
                               GPIO.output(self.AIN2,GPIO.HIGH)
                         self.PWMA.ChangeDutyCycle(0   -   right)
                  if((left   >=   0)   and   (left   <=   100)):
                               GPIO.output(self.BIN1,GPIO.HIGH)
                                GPIO.output(self.BIN2,GPIO.LOW)
                                self.PWMB.ChangeDutyCycle(left)
                  elif((left   <   0)   and   (left   >=   -100)):
                                GPIO.output(self.BIN1,GPIO.LOW)
                               GPIO.output(self.BIN2,GPIO.HIGH)
                        self.PWMB.ChangeDutyCycle(0   -   left)

global                                                      Ab
Ab                          =                       AlphaBot2()
Ab.stop()

def                                    runWheelsRoutine(input):
                                global                          Ab
                                                print(input.data)
                                                          try:
```

```
                        if        input.data        ==        1:
                                                       Ab.stop()
                       elif        input.data        ==        2:
                                                     Ab.forward()
                       elif        input.data        ==        3:
                                                    Ab.backward()
                       elif        input.data        ==        4:
                                                       Ab.left()
                       elif        input.data        ==        5:
                                                      Ab.right()
                        except                KeyboardInterrupt:
                                                    GPIO.cleanup()

def                                                rosNodeWheels():
                print('Node          Wheels          Iniciado')
                                   rospy.init_node('run_wheels')
            rospy.Subscriber('slam_manager_wheels',      Int32,
runWheelsRoutine)
                                                     rospy.spin()

if                                          __name__=='__main__':
        rosNodeWheels()
```

ANEXO D – SlamManager

""" Titulo: SlamManager

Autor: Autor do artigo

Data: 2020 """

```
#!/usr/bin/env                                          python3
import                                                    rospy
from            std_msgs.msg            import            Int32
from           nav_msgs.msg           import           Odometry
from         sensor_msgs.msg          import          LaserScan
from          tf          import          TransformBroadcaster
from               rospy                import                Time

global                                                lastLeftMsg
global                                               lastRightMsg
global                                              lastCenterMsg
lastLeftMsg                          =                        0.0
lastRightMsg                          =                       0.0
lastCenterMsg                          =                      0.0
odometryArray                          =                      0.0
```

```
global checkingPosition
checkingPosition = 0
global isFirst
isFirst = True

pubServos = rospy.Publisher('slam_manager_servos', Int32,
queue_size=1)
pubWheels = rospy.Publisher('slam_manager_wheels', Int32,
queue_size=1)

def angleServos(anglePos):
    pubServos.publish(anglePos)

def checkArea():
    global checkingPosition
    angleServos(checkingPosition)
    checkingPosition = checkingPosition + 1

def moveRobot(pos):
    pubWheels.publish(pos)

def move_robot_model_x():
    translation = (0.2, 0.0, 0.0)
    rotation = (0.0, 0.0, 0.0, 1.0)
    b = TransformBroadcaster()
    b.sendTransform(translation, rotation, Time.now(),
'base_link', '/map')
    b.sendTransform(translation, rotation, Time.now(), 'cam-
era_link', '/map')
    b.sendTransform(translation, rotation, Time.now(), 'cam-
era_link_normalized', '/map')

def move_robot_model_rotate():
    translation = (0.0, 0.0, 0.0)
    rotation = (0.0, 3.0, 0.0, 1.0)
    b = TransformBroadcaster()
    b.sendTransform(translation, rotation, Time.now(),
'base_link', '/map')
    b.sendTransform(translation, rotation, Time.now(), 'cam-
era_link', '/map')
    b.sendTransform(translation, rotation, Time.now(), 'cam-
era_link_normalized', '/map')

def moveDecision():
    global lastLeftMsg
    global lastRightMsg
    global lastCenterMsg
```

```
    if lastCenterMsg > max(lastLeftMsg,lastRightMsg):
        print('Indo para Frente')
        moveRobot(2)
    elif lastLeftMsg > max(lastCenterMsg,lastRightMsg):
        print('Girando para Esquerda')
        moveRobot(4)
        rospy.sleep(1.)
        print('Indo para Frente')
        moveRobot(2)
    elif lastRightMsg > max(lastCenterMsg,lastLeftMsg):
        print('Girando para Direita')
        moveRobot(5)
        rospy.sleep(1.)
        print('Indo para Frente')
        moveRobot(2)
    else:
        print('Indo para Frente Else',lastCenterMsg, lastLeftMsg, lastRightMsg)
        moveRobot(2)
        rospy.sleep(1.)
        pubWheels.publish(1)

def assignMessage(msg):
    global lastLeftMsg
    global lastRightMsg
    global lastCenterMsg
    if checkingPosition == 0:
        lastCenterMsg = msg.ranges[180]
    elif checkingPosition == 1:
        lastRightMsg = msg.ranges[180]
    elif checkingPosition == 2:
        lastLeftMsg = msg.ranges[180]
    print('LastCenterMsg', lastCenterMsg)
    print('LastRightMsg', lastRightMsg)
    print('LastLeftMsg', lastLeftMsg)


def slam_manager(msg):
    global isFirst
    if isFirst:
        print('Angulando robo para posicao inicial')
        angleServos(2)
        isFirst = False
    else:
        global checkingPosition
        print('Laser Center', msg.ranges[180])
        assignMessage(msg)
```

```
                if checkingPosition != 3:
                                    checkArea()
                                            else:
                                    moveDecision()
                        checkingPosition = 0
                                    angleServos(2)
        print('-----------------------------------------------')
                                    rospy.sleep(5.)

if __name__== '__main__':
            print('starting slam manager')
        print('-----------------------------------------------')
                            rospy.init_node('slam_manager')
    sub = rospy.Subscriber('/scan', LaserScan, slam_manager)
    rospy.spin()
```