Mapeamento de ambiente 2D através de sensor laser utilizando um robô diferencial móvel

Victor Tadashi Pereira [1] e Roberto Santos Inoue [2]

***Resumo*— Localização e mapeamento simultâneos se trata de uma técnica aplicada em robótica que se utiliza de sensores para adquirir conhecimento do ambiente e da trajetória realizada por um robô móvel. Esta técnica também é conhecida pela sigla *SLAM (Simultaneos Localization and Mapping)* e é comumente usada em robótica para várias aplicações. Uma das técnicas de *SLAM* usadas neste projeto de graduação é desenvolvido pela equipe Hector da T.U. Darmstadt que provou ser solução aplicada a apenas um sensor laser de linha garantindo uma solução rápida e eficiente com baixo ciclos de processamento de dados [1]. Neste artigo foram acoplados um laser Hokuyo UST-10LX, um computador Odroid-XU4 e uma câmera Odroid USB em um robô móvel afim de realizar um mapeamento 2D se utilizando de pacotes encontrados no *ROS (Robot Operating System)* tais como *Hector SLAM*, Urg Node e *RVIZ*. Com isso foi montado um tutorial afim de ajudar trabalhos futuros na hora de implementar sensores laser se utilizando de pacotes do *ROS*.**

Palavras-chave: Hector SLAM, Hokuyo, LIDAR, Odroid, 2D Mapping, ROS.

I. Introdução

O robô móvel inteligente é um sistema que integra tecnologias como a de comunicação, energia, microeletrônica e mecânica. Recentemente, o rápido desenvolvimento de robôs móveis atraiu uma atenção generalizada. Além disso, os robôs móveis também são amplamente utilizados no dia a dia das pessoas, como limpeza de casas, serviços médicos, serviços de alimentação, transporte militar inteligente, entretenimento, entre outros. A premissa de que robôs móveis trabalham de forma autônoma nessas áreas é que os robôs devem ser capazes de gerar mapas no ambiente atual. O *SLAM* que implementa o robô deve conhecer a posição e a localização do robô no ambiente atual. Ou seja, o robô explora todos os cantos do ambiente e determina sua posição no ambiente interno e a orientação do corpo de acordo com os sinais do ambiente. Portanto, *SLAM* é uma questão chave na área de robôs móveis. O conteúdo principal deste artigo é realizar o algoritmo do *Hector SLAM*, ou seja, o robô carrega o laser para escanear o ambiente interno e realizar a auto-localização além de construir mapas internos em um ambiente [2]. O algoritmo *Hector SLAM* utiliza a alta frequência de amostragem do *LIDAR (Light Detection And Ranging)* bidimensional moderno (por exemplo, o Hokuyo UST-10LX tem uma frequência de varredura de 40 Hz) e fornece uma estimativa de posição 2D na taxa de varredura do sensor. Embora o sistema não forneça detecção explícita de fechamento de loop, ele é preciso o suficiente para muitos cenários do mundo real. O *SLAM* de correspondência de varredura é um algoritmo que estima a posição e rotação do robô entre duas varreduras com base na correspondência de varredura do vizinho mais próximo. A rede do feixe de laser é otimizada usando o mapa obtido para representar o ponto do laser no mapa, e a probabilidade de ocupação da grade é estimada. O método Gaussiano-Newton é usado para resolver o problema de correspondência de varredura, e o conjunto de pontos de laser é mapeado para a transformação de corpo rígido (x, y, teta) do mapa existente. Se um *LIDAR* 2D com uma alta frequência de amostragem for usado, o *Hector SLAM* tem um efeito de mapeamento significativo em um pequeno ambiente interno. Como o algoritmo não precisa coletar dados do odômetro, é muito adequado para alguns robôs com computação limitada [2]–[6].

Este projeto tem como objetivo acoplar uma câmera, um Hokuyo UST-10LX (laser óptico) e um Odroid-XU4 (computador) à um SkateBot (robô móvel) para executar um mapeamento 2D de uma determinada área utilizando os pacotes disponíveis no *ROS* tais como *RVIZ* e *Hector SLAM* que serão detalhados a seguir.

O artigo está organizado em 5 seções, contando com esta introdução. A Seção 2 mostra todos os materiais e métodos utilizados neste projeto, com a escolha dos componentes, suas especificações e seus funcionamentos, já na Seção 3 é discutida a parte de implementação de toda a parte de integração do sensor, câmera USB e SkateBot com os pacotes do *ROS*. Na Seção 4 apresenta os resultados obtidos na implementação e execução do projeto e, por fim, na Seção 5 são apresentadas as conclusões obtidas do trabalho.

II. Materiais e Métodos

Os materiais utilizados para a realização deste projeto e suas respectivas especificações e informações de uso podem ser encontrados nos tópicos abaixo.

A. ROS

ROS sistema operacional de robôs, é um projeto *open source*, ou seja, de código aberto no qual permite o desenvolvimento e a aplicação de diversos componentes. Esses componentes ou *frameworks* destinado a aplicações robóticas fornecem uma série de bibliotecas e ferramentas que auxiliam na construção de aplicações robóticas [7]–[9]. Além

[1] Autor é aluno do Departamento de Engenharia Elétrica, Universidade Federal de São Carlos, UFSCar, São Carlos, Brasil victortadashi123@gmail.com.

[2] Orientador é professor do Departamento de Computação, Universidade Federal de São Carlos, UFSCar, São Carlos, Brasil rsinoue@ufscar.br

disso o fato de existir diversos projetos que se utilizam do *ROS* como sua principal ferramenta de desenvolvimento de projetos aplicados a robótica o torna ainda mais atraente para o desenvolvimento do mesmo. O *ROS* utiliza nós [8] e tópicos [9] sendo os nós aqueles que performam algum tipo de atividade computacional através de um programa em C++, Python ou Plugin, e os tópicos são barramentos nomeados pelos quais os nós trocam mensagens. Cada tópico possui uma tipagem o que permite que apenas nós com uma tipagem igual possam receber mensagens desse tópico em questão, além disso os tópicos possuem uma semântica anônima de ler e editar, ou seja, os nós não sabem com quem estão se comunicando além disso um tópico pode possuir diversos editores e leitores porém um nó só pode editar ou ler, ou seja, não pode ler e editar um mesmo tópico. O uso de nós implicam em vários benefícios para o sistema em geral, pois como eles estão separados em pequenas atividades computacionais isso faz com que o sistema não quebre por inteiro caso algum nó falhe.

B. Skate Bot

Skate Bot se trata de um robô móvel de código aberto com rodas de tração diferencial sendo um motor para cada roda, além disso o robô utiliza um chassi de alumínio, uma placa Arduíno Mega e um computador Odroid com uma distribuição Linux de base Debian como demonstra a Figura 1. O Skate Bot possui um controle cinemático para movimentação do mesmo com base em comandos de velocidade linear e angular o que possibilita a locomoção do robô até um dado destino. O controle é feito por intermédio de um controlador embarcado na placa Arduíno Mega e se utiliza da linguagem C++, o modo de tração diferencial possui a vantagem de poder manobra-lo em locais com pouco espaço o que é interessante para o mapeamento 2D [10].

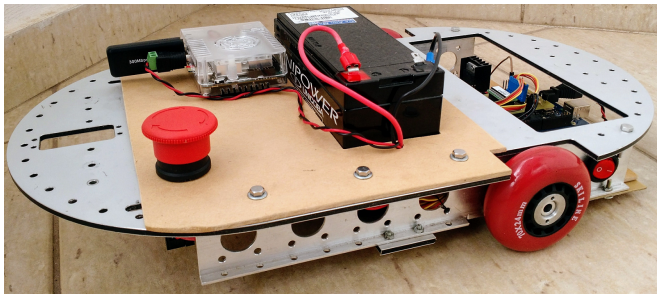


Figura 1. Skate Bot.

C. Odroid-XU4

Na Figura 2 temos um Odroid-XU4 que se trata de um pequeno dispositivo de computação que oferece um suporte de código aberto, a placa possibilita a execução de várias versões de Linux e Android. Ao implementar as interfaces USB 3.0, eMMC 5.0 e Gigabit Ethernet, o Odroid-XU4 oferece velocidade de transferência de dados, um recurso que é cada vez mais necessário para suportar o poder de processamento avançado em dispositivos ARM [11].

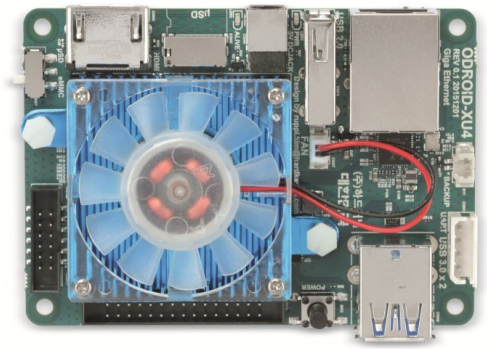


Figura 2. Odroid-XU4.

D. Hector SLAM

O *Hector SLAM* combina um sistema *SLAM* 2D baseado em uma varredura robusta. Os autores focaram na estimativa de movimentação do robô em tempo real, fazendo uso da alta taxa de atualização e baixo ruído dos *LIDARs*. A estimativa de posição 2D é baseada na otimização do alinhamento dos pontos finais do feixe com o mapa obtido até agora. Os pontos finais são projetados no mapa real e as probabilidades de ocupação são estimadas. A correspondência de varredura é resolvida usando uma equação Gaussiana-Newton, que encontra a transformação rígida que melhor se adapta aos feixes de laser com o mapa. Além disso, uma representação de mapa de resolução múltipla é usada, para evitar ficar preso em mínimos locais [12].

E. RVIZ

RVIZ é uma ferramenta visualização 3D e 2D que permite a depuração de aplicações robóticas e possibilita a visualização do que o robô está vendo, pensando e fazendo. Existem dois meios de inserir dados no *RVIZ*, primeiramente ele entende informações de sensor e estado, como varredura a laser, nuvens de pontos, câmeras e quadros de coordenadas. Ele possui configurações personalizadas que permitem o usuário escolher como ele prefere ver a informação selecionada. A segunda forma são marcações visuais que permitem enviar primitivas como cubos, erros e linhas com a coloração desejada. A combinação de sensores de dados e marcações visuais customizadas tornam o *RVIZ* uma ferramenta poderosa para desenvolver as capacidades do robô e pesquisa [13].

F. Hokuyo UST-10LX

O mapeamento foi feito utilizando os parâmetros técnicos do Sensor Hokuyo UST-10LX Scanning Laser Rangefinder (UUST103) que pode ser visto na Figura 3, o Hokuyo UST-10LX é um dispositivo pequeno, preciso e de alta velocidade para detecção de obstáculos e localização de robôs

autônomos e sistemas automatizados de manuseio de materiais. Este modelo utiliza interface Ethernet para comunicação e pode obter dados de medição em amplo campo de visão até uma distância de 10 metros com resolução milimétrica. Devido ao seu baixo consumo de energia, este scanner pode ser usado em plataformas operadas por bateria. Este sensor usa uma fonte de laser para escanear o campo de visão de 270 graus. As posições dos objetos no intervalo são calculadas com o ângulo do passo e a distância. O sensor envia esses dados por meio do canal de comunicação.

Figura 3. Hokuyo UST-10LX.

A Hokuyo, empresa especializada em fabricação de sensores ópticos e dispositivos de automação com ampla aplicação em engenharia [14]. Os valores técnicos do sensor podem ser vistos na Tabela I. Tais valores foram utilizados para a modelagem do sensor de maneira fiel ao real.

Descrição	Valor
Nome do Produto	Telêmetro de varredura laser
Modelo	UST-10LX
Tensão de alimentação	12VDC/24VDC
Corrente de alimentação	150mA ou menos
Fonte de luz	Laser semicondutor
Faixa de detecção	0.06m até 10m
Max. distância de detecção	30m
Precisão	$\pm 40mm(*1)$
Precisão repetida	$\sigma < 30mm(*1)$
Ângulo de varredura	270°
Velocidade de varredura	25ms (2400rpm)
Resolução angular	0.25°
Tempo de inicialização	Dentro de 10 segundos
Entrada	Corrente 4mA em ON
Saída	30VDC 50mA MAX
Interface	Ethernet 100BASE-TX
Temperatura e umidade ambiente	-10°C até +50°C
Resistência à vibração (operando)	55 até 150Hz
Estrutura Protetora	IP65
Peso	130g (Sem cabo)
Dimensões (L $\times P \times A$)	50 $\times$50 $\times$ 70mm (apenas sensor)

Tabela I
ESPECIFICAÇÕES TÉCNICAS DO UST-10LX.

G. Urg Node

O Urg_Node se trata de um nó que irá publicar os dados da varredura do laser Hokuyo UST-10LX para o tópico /scan10 e este tópico por sua vez será subscrito por outros nós para usar estes dados para visualização ou controle.

H. Odroid Câmera USB

Esta câmera HD USB que pode ser vista na Figura 4 oferece várias opções para projetos baseados em segurança com sua placa Odroid. A câmera possui uma resolução HD de 720p, USB 2.0 de alta velocidade e interface *plug-n-play* além de possuir uma taxa de quadros de até 30 quadros por segundo.

Figura 4. Odroid USB Câmera.

III. IMPLEMENTAÇÃO

A seguir será apresentado a maneira na qual o projeto foi feito, ou seja, sua implementação, tais como instalações dos pacotes e mudanças feitas nos mesmos.

A. Computador e o computador embarcado

Para o inicio do trabalho foram utilizado nos dois computadores versões de Ubuntu iguais para que não aconteça conflitos de software, no qual foi utilizado Ubuntu 18.04 e o *ROS* utilizado versão *Melodic* pois na versão Kinect (16.04) apresentou alguns erros de processamento de video para o *LIDAR*. Esses problemas se deram pois a versão 16.04 do Ubuntu apresenta uma incompatibilidade com o OpenGL fazendo com que um dos Nós (hector_trajectory_server) do pacote *Hector SLAM* apresentasse erro e com isso a trajetória do robô acabava se locomovendo demais fazendo o projeto ficar inviável.

Depois das instalação do *ROS Melodic* é necessário instalar o Catkin, o mesmo serve para executar diversos pacotes independente ao mesmo tempo, funcionando como um *workspace*.

B. Instalação do pacote skate_bot

O pacote skate_bot foi desenvolvido pelo aluno Vitor Izumino como trabalho final de curso, no qual consistia fazer um robô móvel diferencial, para a implementação do robô é necessário fazer alguns preparos. Os comandos a seguir podem precisar de alterações dependendo da localização de seus pacotes. Primeiramente foi copiada a pasta do pacote skate_bot para dentro da pasta /src que por sua vez está

localizada dentro da pasta /catkin_ws. Em seguida foi dado o comando catkin_make e copiado o arquivo skate_bot.rules para dentro da pasta /etc/udev/rules.d conforme os comandos.

```
cd /skate_bot

cp /skate_bot ~/catkin_ws/src/

cd ~/catkin_ws

catkin_make

cd

sudo cp skate_bot.rules /etc/udev/rules
    .d
```

O arquivo skate_bot.rules serve para determinar alguns *inputs* e *outputs* para o funcionamento do controlador via Arduíno.

C. SSH

O SSH é um protocolo de rede para fazer a comunicação dos servidores que você deseja. Então no projeto foi feito a comunicação do computador deixando o computador como ”master” e o computador embarcado (Odroid) como o ”slave”. Os comandos abaixo devem ser utilizados no PC Master.

```
export ROS_IP="IP_PC_MASTER"

export ROS_MASTER_URI="http://IP_PC_MASTER
    :11311"

echo 'export ROS_IP="IP_PC_MASTER"' >> ~/.
    bashrc

echo 'export ROS_MASTER_URI="http://
    IP_PC_MASTER:11311"' >> ~/.bashrc
```

O IP_PC_MASTER é encontrado no comando Ifconfig mostrando o IP da conexao Wlan do Master. As linhas de comando anteriores alteram o arquivo bashrc, para que não seja necessário executar-los em toda nova aba do terminal.

No PC Slave foram definidos os seguintes comandos.

```
export ROS_IP="IP_ODROID"

export ROS_MASTER_URI="http://IP_PC_MASTER
    :11311"

echo 'export ROS_IP="IP_ODROID"' >> ~/.
    bashrc

echo 'export ROS_MASTER_URI="http://
    IP_PC_MASTER:11311"' >> ~/.bashrc
```

Utilizando o comando ifconfig para encontrar o IP do slave. Aonde fica ROS_IP vai ser utilizado IP_ODROID do slave e ROS_MASTER_URI o IP_PC_MASTER.

D. Controle de Xbox One

Para fazer a instalação do controle de Xbox One de inicio utiliza o comando no computador Master:

```
sudo apt-get install ros-melodic-teleop-
    twist-joy
```

Depois conectar o controle de Xbox One no computador em seguida utilizar o comando:

```
cd /dev/input

ls
```

Utilizando comando ls para aparecer a lista de todos os arquivos e pastas contidos na pasta atual conforme a Figura 5.

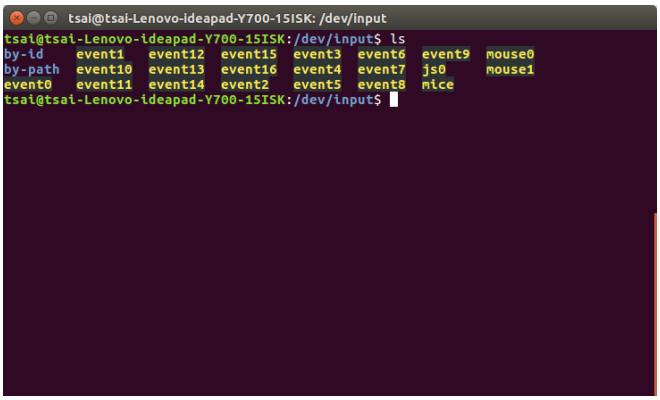


Figura 5. Tela demonstrando o input do controle.

Todos os controles possuem *input* no formato jsX (aonde X é o número do input), como observado na Figura 5 o controle está com *input* de js0, caso possua mais de um *input* no formato jsX o usuário deve retirar o controle, executar o comando ls novamente e verificar qual *input* desaparece e aparece conforme o controle é colocado e retirado. Com isso podemos alterar o valor do *input* no pacote do teleop_twist_joy utilizando os seguintes comandos:

```
roscd teleop_twist_joy/launch/

sudo nano teleop.launch
```

Após a utilização destes comandos, alterar as linhas de código do *launch* do pacote teleop_twist_joy.

As linhas que precisam ser alteradas são as seguintes:

```
<arg name="joy_config" default="xbox" />

<arg name="joy_dev" default="/dev/input/
    js0" />
```

Adequar a linha do joy_config para seu tipo de controle, verificando se há script para seu controle, caso não tenha fazer um script para seu controle.

Alterar a linha de comando para o input (joy_dev) conforme a Figura 6 para que pacote consiga receber dados do controle.

Após fazer as alterações o controle deve estar pronto para funcionar, lembrando que o *roscore* deve estar ativado em alguma aba do terminal para funcionar.

Figura 6. Alteração do input no launch do pacote do teleop_twist_joy.

```
rosrun teleop_twist_joy teleop_node
```

```
roslaunch teleop_twist_joy teleop.launch
```

Para verificar o funcionamento utilizar o comando *Rostopic* para visualizar os valores do tópico escolhido.

```
Rostopic echo /joy
```

Os valores booleanos da Figura 7 apresentam as entradas do controle, o valor 1 quando o botão está pressionado e valor 0 quando o contrário.

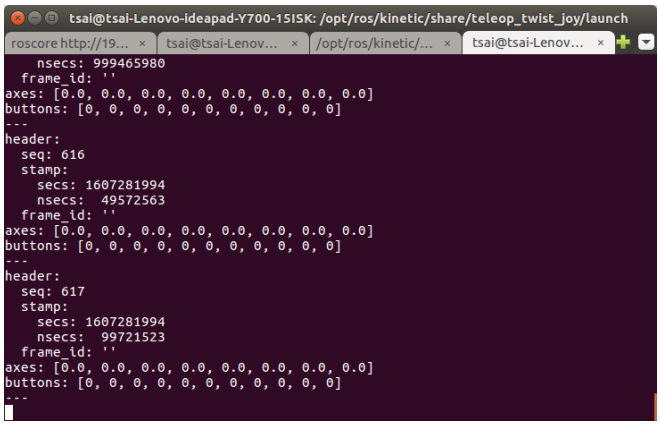


Figura 7. Funcionamento do botão do teleop_twist_joy.

Após o funcionamento do teleop_twist_joy, utilizamos o comando:

```
rqt_graph
```

Com o intuito de visualizar as ligações e tipo de saída de dados no pacote do teleop_twist_joy, no qual é utilizado para fazer o robô andar emitindo o valores do *input* para /cmd_vel mostrado na Figura 8.

E. Câmera USB

Para implementação da câmera USB, utilizar o seguinte comando:

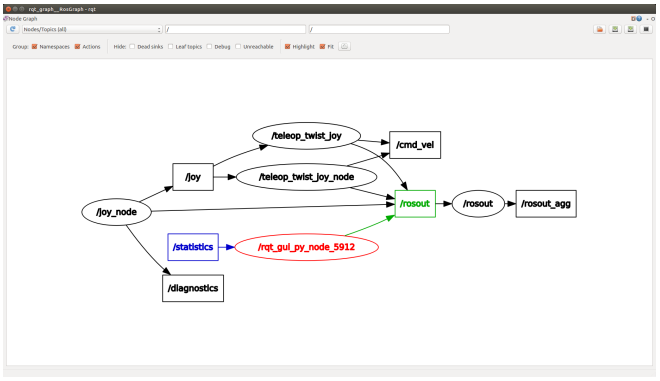


Figura 8. Rqt_graph teleop_twist_joy.

```
sudo apt-get install ros-melodic-usb-cam
```

Igualmente aos outros pacotes identificar o *input* da câmera, no qual o *input* tem formato de videoX (onde X é um número inteiro variável), alterar no *launch* do pacote da câmera o parâmetro *value*.

```
<param name="video_device" value="/dev/
    video1" />
```

O *launch* se encontra no arquivo usb_cam-test.launch que está localizado na pasta usb_cam/launch, exemplo de como localizar a pasta e arquivo em questão:

```
roscd usb_cam/launch
sudo nano usb_cam-test.launch
```

Após alterar o pacote da câmera utilizar o seguinte comando e adicionar o tópico */image_raw* no Rviz para visualizar a imagem da câmera conforme a Figura 9.

```
rviz rviz
```

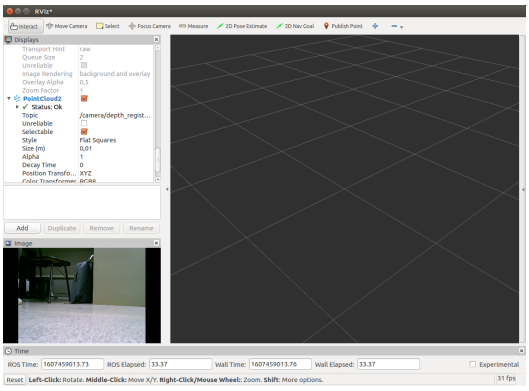


Figura 9. Camera USB no *RVIZ* no canto esquerdo inferior.

*F. Laser (*LIDAR*)*

A implementação do laser com o modelo do Hokuyo UST-10LX precisa ser feito algumas alterações nas opções de internet, pois o Hokuyo UST-10LX utiliza conexão via cabo ethernet com isso foi preciso deixar as opções de ajustes conforme mostrado nas Figuras 10 e 11, com a conexão via

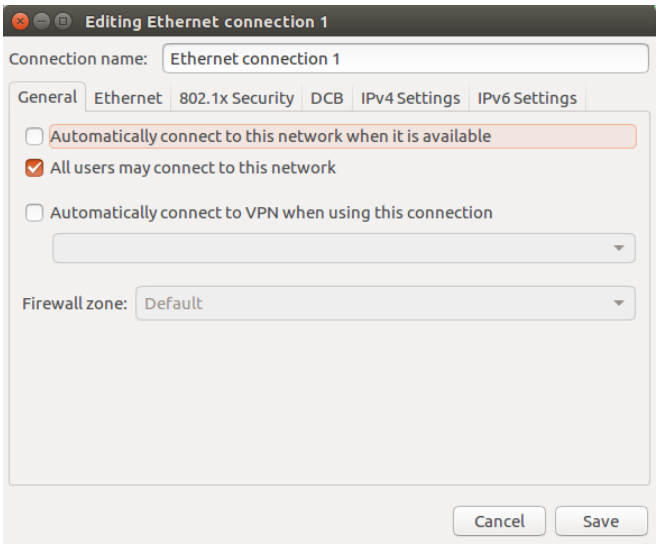


Figura 10. Edição da Conexão Ethernet.

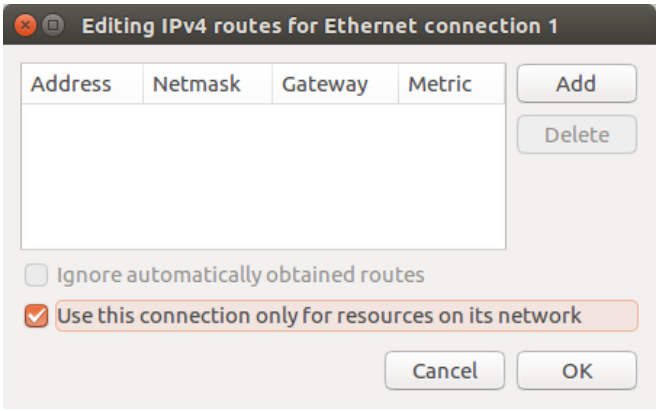


Figura 11. Edição das Rotas IPV4.

SSH necessita fazer esses ajustes para que funcione o laser e o Wi-Fi simultaneamente.

Após as alterações é necessário a instalação do pacote do *urg_node*, no qual se refere ao laser.

```
sudo apt-get install ros-melodic-urg-node
```

Após a instalação do pacote é necessário verificar o IP do seu laser no manual no caso o laser utilizado possui o IP 192.168.1.15. Com isso fazer uma nova configuração da ethernet precisa ser um IP com o mesmo caminho porém com canal diferente, por isso vamos configurar a ethernet como mostrado a Figura 12, alterando a configuração no IPV4 address para 192.168.1.10, mask para 255.255.255.0 e gateway para 192.168.1.255.

Após fazer a configuração é preciso rodar o seguinte código para funcionamento do laser:

```
rosrun urg_node urg_node _ip_address
    :="192.168.1.15" __name:=urg_node10
    scan:=scan10
```

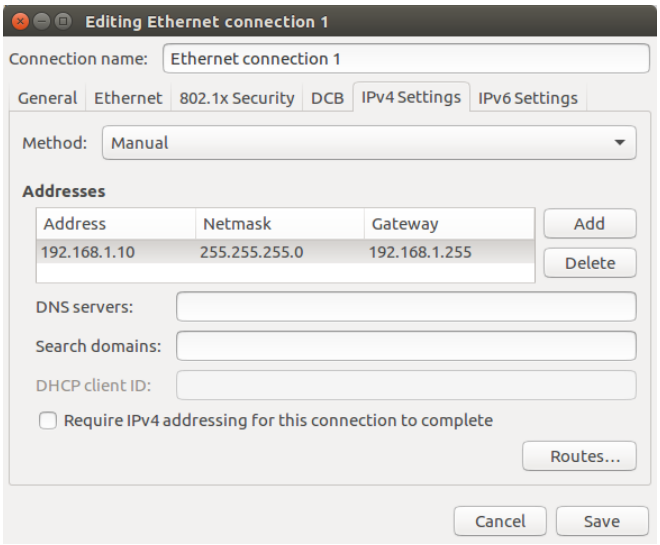


Figura 12. Configuração da ethernet para o laser

Depois é necessário abrir o *RVIZ* e trocar o *Fixed Frame* para laser e adicionar o tópico *laser_scan* e o resultado mostrado na Figura 13.

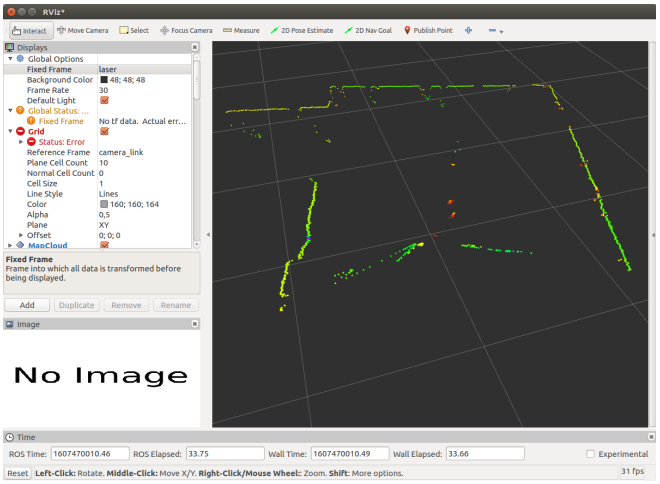


Figura 13. Funcionamento do mapeamento do laser

A sala na qual o mapeamento está sendo executado é mostrada na Figura 14.

Para a instalação do pacote *Hector SLAM* é necessário o seguinte comando:

```
sudo apt-get install ros-melodic-hector-
    slam
```

Em seguida é necessário alterar algumas linhas do arquivo *mapping_default.launch* localizado na pasta *hector_mapping* do pacote *Hector SLAM*.

```
<arg name="
    tf_map_scanmatch_transform_frame_name"
     default="scanmatcher_frame"/>
<arg name="base_frame" default="laser"/>
```

Figura 14. Demonstrativo da sala utilizada

```
<arg name="odom_frame" default="laser"/>
<arg name="pub_map_odom_transform" default
    ="true"/>
<arg name="scan_subscriber_queue_size"
    default="5"/>
<arg name="scan_topic" default="scan10"/>
<arg name="map_size" default="2048"/>
```

Após isso foi criado um arquivo e nomeado neste caso como hokuyo.launch (código disponível no Apêndice I) para se adequar aos parâmetros do laser utilizado. Enfim foi executado o *Hector SLAM* utilizando o arquivo previamente criado com a seguinte linha de código.

```
roslaunch hector_slam_launch hokuyo.launch
```

IV. Resultados e Discussão

O experimento do algoritmo *Hector SLAM* foi feito no mesmo ambiente da Figura 14, este ambiente possuía diferentes estruturas incluindo cadeiras, mesas, armários, paredes e ferramentas. Devido ao posicionamento de cada uma das estruturas a complexidade do ambiente aumenta. Como experimento foi feito uma mapeamento utilizando o Skate_bot para mapear o ambiente previamente citado e o resultado do mapeamento pode ser visto na Figura 15. Notou-se uma baixa taxa de quadros da câmera devido ao baixo poder de processamento do Odroid-XU4 e erros no algoritmo do *SLAM* quando o robô se locomovia verticalmente, ou seja, não é recomendado utilizar essa configuração para mapear ambientes acidentados ou com robôs VANT (Veículo aéreo não tripulado) como drones. Notou-se também que a qualidade do mapeamento 2D varia conforme a velocidade com que o SkateBot se locomove como pode ser visto na Figura 16, isso também se deve ao baixo processamento do Odroid-XU4, logo para projetos futuros é aconselhável se utilizar de um computador com maior poder de processamento do que o utilizado neste projeto. Apesar das desvantagens citadas acima foram obtidos bons resultados uma vez que mesmo no caso aonde a velocidade do robô móvel foi aumentada o mapeamento ainda assim se encontra aceitável em relação aos resultados obtidos para diferentes velocidades em [2] como pode ser visto na Figura 17.

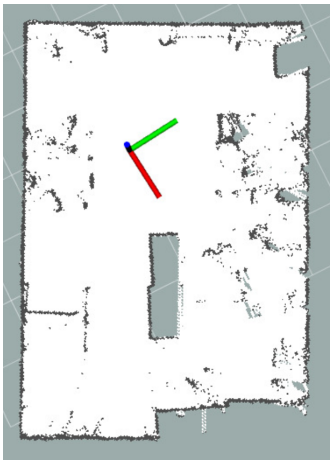


Figura 15. Mapeamento 2D utilizando *Hector SLAM*.

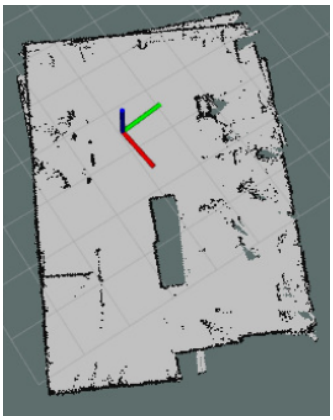


Figura 16. Mapeamento 2D com maior velocidade no robô.

V. Conclusão

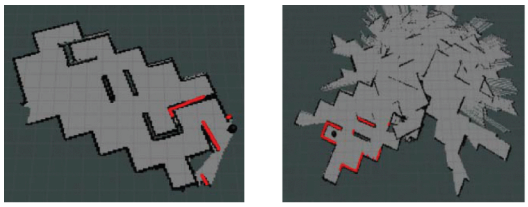
Ao longo deste projeto foram desenvolvidas medições se utilizando de apenas um sensor de linha, um robô de tração diferencial, uma câmera USB e um computador. Utilizando o pacote do *Hector SLAM* para inferir os dados de mapeamento e localização. Neste projeto tivemos um bom resultado isso pode ser notado uma vez que o mesmo não apresenta paredes duplicadas e demarca os diversos objetos encontrados no laboratório tais como armários, mesas e fios onde foi realizado o mapeamento, o bom resultado se deve a qualidade do laser utilizado, mesmo se utilizando de um mapeamento sem odometria e um computador embarcado com baixo poder de processamento. O projeto acabou servindo como um tutorial com intuito de ajudar na futura implementação de sensores laser para mapeamento utilizando-se do *Hector SLAM*.

Apêndice I
Hokuyo.launch

Esse launch foi feito para se adequadar aos parâmetros do Hokuyo UST-10LX e foi utilizado ao invés do tutorial.launch [15].

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

 <launch>
  <param name="pub_map_odom_transform" value="
      true"/>
```



(a) lower speed (b) higher speed

Figura 17. Mapeamento 2D utilizando o *Hector SLAM* para velocidades abaixo e acima de 0.2 m/s, imagem gerada por Xuexi et al [2].

```
<param name="map_frame" value="map"/>
<param name="base_frame" value="base_frame
    "/>
<param name="odom_frame" value="odom"/>

<node pkg="tf" type="
    static_transform_publisher" name="
    map_2_odom"
args="0 0 0 0 0 0 /map /odom 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="odom_2_base_footprint" args="0 0 0 0 0
     0 /odom /base_footprint 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="base_footprint_2_base_link" args="0 0
    0 0 0 0 /base_footprint
/base_link 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="base_link_2_base_stabilized_link" args
    ="0 0 0 0 0 0
/base_link /base_stabilized 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="base_stablized_2_base_frame" args="0 0
     0 0 0 0
/base_stabilized /base_frame 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="base_frame_2_laser_link" args="0 0 0 0
     0 0
/base_frame /laser 100"/>

<node pkg="tf" type="
    static_transform_publisher"
name="base_2_nav_link" args="0 0 0 0 0 0 /
    base_frame /nav 100"/>

<node pkg="rviz" type="rviz" name="rviz"
args="-d $(find hector_slam_launch)/rviz_cfg
    /mapping_demo.rviz"/>

<include file="$(find hector_mapping)/launch
    /mapping_default.launch"/>
<include file="$(find hector_geotiff)/launch
    /geotiff_mapper.launch"/>

</launch>
```

REFERENCES

[1] F. Q. Bordon and D. D. Goulart, "Sistema de mapeamento robótico para corpos móveis com redundância sensorial," 2020.

[2] Z. Xuexi, L. Guokun, F. Genping, X. Dongliang, and L. Shiliu, "Slam algorithm analysis of mobile robot based on lidar," July 2019.

[3] W. Wei, B. Shirinzadeh, S. Esakkiappan, M. Ghafarian, and A. Al-Jodah, "Orientation correction for hector slam at starting stage," in *2019 7th International Conference on Robot Intelligence Technology and Applications (RiTA)*, 2019.

[4] N. Yu and B. Zhang, "An improved hector slam algorithm based on information fusion for mobile robot," in *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 2018.

[5] W. WEI, B. Shirinzadeh, M. Ghafarian, S. Esakkiappan, and T. Shen, "Hector slam with icp trajectory matching," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 2020.

[6] J. Yang, C. Wang, Q. Zhang, B. Chang, F. Wang, X. Wang, and M. Wu, "Modeling of laneway environment and locating method of roadheader based on self-coupling and hector slam," in *2020 5th International Conference on Electromechanical Control Technology and Transportation (ICECTT)*, 2020.

[7] "About ros," 2020, acessado: 2020-11-15. [Online]. Available: http://www.ros.org/about-ros

[8] "About ros nodes," 2020, acessado: 2020-11-15. [Online]. Available: http://wiki.ros.org/Nodes

[9] "About ros topics," 2020, acessado: 2020-11-15. [Online]. Available: http://wiki.ros.org/Topics

[10] V. I. Sgrignoli, "Prototipagem de um robô de tração diferencial," *Projeto de Conclusão de Curso - Programa de Graduação em Engenharia Elétrica, Universidade Federal de São Carlos, São Carlos*, pp. 14–58, 2017.

[11] "About odroid xu4," 2020, acessado: 2020-11-15. [Online]. Available: https://www.odroid.co.uk/hardkernel-odroid-xu4/odroid-xu4

[12] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," November 2011.

[13] "About rviz," 2020, acessado: 2020-11-15. [Online]. Available: http://wiki.ros.org/rviz

[14] "Ust-10lx," 2016, acessado: 2020-11-15. [Online]. Available: https://www.hokuyo-usa.com/products/scanning-laser-rangefinders/ust-10lx

[15] "Launch hokuyo-hector-slam," 2020, acessado: 2020-12-23. [Online]. Available: https://github.com/xpharry/hokuyo-hector-slam/blob/master/launch/hector_hokuyo.launch