Implementações de sensores utilizando ROS para um robô móvel com rodas.

Andre Miyagusuku Tsai [1] e Roberto Santos Inoue [2]

***Resumo*— Com a expansão da indústria 4.0, atividades logísticas com robôs móveis e AGVs (*Automated Guided Vehicles*) são cada vez mais indispensáveis nas indústrias. Para fazer a navegação nas logísticas da indústria, há desafios a serem solucionados em robótica móvel para os problemas de localização, mapeamento de ambientes, e desvio de obstáculos. O mais básico deles é a integração de diversos sensores, atuadores e algoritmos, que são utilizados em um robô móvel através de um computador embarcado . Nesse sentido, este trabalho fornece um tutorial básico para integração de diversos tipos de sensores, como IMU (*Inertial Measurement Unit*), GPS (*Global Positioning System*), câmera, laser LiDAR (*Light Detection And Raging*) e sensor de movimento Kinect do Xbox 360, implementados com um computador embarcado para recepção e processamentos de dados utilizando o *framework* ROS (*Robot Operating System*). Foram realizados testes com alguns pacotes ROS, como teleop_twist _joy, myahrs_drive, skate_bot, nmea_navsat_driver, usb_cam, urg_node, freenect, rtabmap_ros, de forma que cada um destes pacotes foi escolhido para uso com cada um dos sensores. Foram realizados testes em ambiente físico mostrando o funcionamento dos sensores de forma conjunta integrado ao robô.**

Palavras-Chave - Sensores, ROS, Robô Móvel, Integrações, Linux.

I. Introdução

O termo ”Indústria 4.0” geralmente se refere a um conjunto de habilitação de tecnologias na área de automação industrial com os seguintes objetivos: melhorar a saúde dos trabalhadores, aumentar produtividade e qualidade da produção das indústrias [1]. Outro termo bastante comum na indústria 4.0 é o CPS (*Cyber Physical System*) refere-se a todos os sistemas físicos que podem interagir com outros sistemas físicos e operadores humanos por meio da troca de dados via *software*. Este é o pressuposto para a descentralização e cooperação entre diferentes sistemas dentro de um ambiente em nuvem, que está diretamente conectado à ideia de Indústria 4.0. Formalmente, podemos definir um CPS como uma rede [2] que realiza a integração entre os elementos físicos (como sensores e atuadores) e dados computacionais [3], [4].

Com a expansão da Indústria 4.0, o uso dos AGVs começou a desempenhar um papel importante nas empresas que realizam gestão de tarefas internas e atividades logísticas [5]. A ideia de usar robôs que podem ser controlados remotamente ao invés do uso de um humano é um dos principais alvos do CPS. A fim de controlar e navegar os AGVs com CPSs, localização e mapeamento de ambiente, a posição dos obstáculos móveis e fixos, e a alocação ótima de tarefas deve ser superada. Contudo, uma infraestrutura de comunicação e um mecanismo de gerenciamento em tempo real comumente necessarios para lidar com todas essas dificuldades e para permitir que vários AGVs funcionem em harmonia [6].

Nos últimos anos, com o desenvolvimento da tecnologia de robôs, o problema de estimativa de dados tornou-se cada vez mais importante. A localização atual geralmente é alcançada por mesclar as informações internas e externas, coletadas através dos sensores do robô [7]. Sensores internos geralmente incluem giroscópio de fibra ótica, IMU e codificador [8], enquanto que medições externas geralmente são realizadas por GPS e laser.

Um robô é um mecanismo multifuncional reprogramável projetado para mover materiais, ferramentas, ou dispositivos especializados por meio de movimentos variados, conforme definição dada pelo *Robot Institute of America* em 1979 [9].

Robôs móveis são capazes de locomover-se no ambiente em que estão inseridos. Fisicamente, um robô móvel pode ser decomposto em um mecanismo para fazer o robô locomover-se pelo ambiente, em um ou mais computadores para controlar o robô e uma coleção de sensores com os quais o robô obtém informação do ambiente [10].

Na robótica movél um *software* muito utilizado é o ROS. Por ele ser um *framework* flexível, fornece serviços padrões de sistema operacional, tais como abstração de hardware, controle de dispositivos de baixo nível, a implementação de funcionalidades comumente usadas, passagem de mensagens entre processos e gerenciamento de pacotes. É uma coleção de ferramentas, bibliotecas e convenções que visam simplificar a tarefa de criar comportamento robusto e complexo do robô em uma ampla variedade de plataformas robóticas. Futuramente vai ser explicado mais sobre os *frameworks* utilizados do ROS [11].

Portanto, objetivo desse projeto consiste em utilizar o ROS como o *software* de comunicação entre o robô e os sensores, mostrando a integração desde a instalação dos pacotes ROS até a visualização do funcionamento dos sensores, que consistem na IMU, câmera, Laser LiDAR, Kinetic Xbox360. Esses tipos de sensores escolhidos são alguns dos sensores mais comuns utilizados para navegação e mapeamentos.

O artigo está organizado em cinco seções, contando com esta introdução. A Seção 2 mostra todos os materiais e métodos utilizados, com a escolha dos componentes e seu funcionamento, na Seção 3 é realizada a discussão de toda a parte de integração dos sensores com os pacotes do ROS. A Seção 4 traz os resultados obtidos da implementação e, por fim, na Seção 5 são apresentadas as conclusões obtidas do

[1] Andre Miyagusuku Tsai é aluno do Departamento de Engenharia Elétrica, Universidade Federal de São Carlos, UFSCar, São Carlos, Brasil andretsai@estudante.ufscar.br

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

trabalho.

II. Materiais e métodos

O projeto consiste da implementação de diversos sensores, a importância de cada sensor está explicitada nas linhas abaixo a fim de esclarecer quais suas importâncias no momento atual, correlacionado à robótica móvel e à era da indústria digital em questão. A era atual de indústria 4.0, digitalização e decisões provenientes de análise de dados, compõe uma parcela de extrema importância no desenvolvimento de aplicações voltada para robótica em geral. Sensores inteligentes são um mecanismo para a aproximação e flexibilização da interpretação de dados e assim permitir uma tomada de decisão ou um processamento mais simples e eficiente [12].

A. ROS

O sistema operacional de robôs, é uma coleção de *frameworks* para aplicações robóticas. Esse sistema operacional é um projeto de código aberto que suporta o desenvolvimento e aplicação de diversos componentes. Tal sistema fornece mecanismos de abstração de *hardware*, *drivers*, bibliotecas, ferramentas de visualização, pacotes de gerenciamento de ferramentas, etc. Atualmente, diversos dispositivos e sensores desenvolvem bibliotecas para o ROS a fim de facilitar a programação de aplicações, como é discutido em [13] e [14].

Por ser um ferramenta didática e possuir uma certa rastreabilidade em todas as ferramentas, funções e bibliotecas disponíveis para o uso geral, o ROS mostra-se um recurso valioso para a aplicação de robótica e desenvolvimento de algoritmos voltados para o mesmo. Somado a isso, existem na literatura diversos projetos que utilizam o ROS como ferramenta principal para o desenvolvimentos de projetos aplicados à robótica (vide referências utilizadas: [14], [15], [16], [17] , [11]).

Os *frameworks* utilizados nesse projetos são: Nós, Tópicos, Pacotes, Rosbag, Rviz que vão ser explicados nos tópicos futuros nos quais são baseados no trabalho [18].

1) Nós: Os nós são compostos de programas ativos e executáveis usados por cada máquina, e esses nós podem enviar informações à rede para um tópico específico (*publisher node*) ou podem ler dados que vêm de um tópico específico (*subscriber node*). Neste projeto, o uso de nós foi importante, por exemplo, para fazer a interface de sensores, para navegar e mapear o terreno, para fazer a comunicação entre vários dispositivos embarcados de robôs móveis e também para realizar a experimentação virtual dos algoritmos criados.

2) Tópicos: Os tópicos são responsáveis por armazenar as mensagens que circulam através do núcleo ROS (mestre) para permitir que elas sejam lidas por um nó assinante ou alteradas por um nó editor. Os tópicos podem ser entendidos como variáveis globais da rede, que podem ser acessadas por todos os elementos conectados. Usando o terminal, você pode obter diferentes informações sobre as mensagens trocadas na rede por meio de comandos como *rostopic list* (que mostra todos os tópicos do sistema), *rostopic pub* (que envia dados para um tópico específico) ou *rostopic echo* (que visualiza dados de um tópico específico).

3) Pacotes: Os Pacotes ROS são compostos por um número arbitrário de nós que permite aos usuários terem uma estrutura mais organizada e são usados para dar à rede diferentes módulos com funcionalidade total. Embora a estrutura do ROS seja complexa de entender, ela garante que todas as unidades do sistema se comunicarão de maneira consistente e padronizada para evitar qualquer conflito de informações e dados. A Fig. 1 obtida com o comando `rqt_graph` mostra um exemplo simples de como a comunicação ocorre entre os vários nós do sistema dentro do exemplo padrão do ROS denominado “turtlesim”, sendo o nó `"/turtlesim"` o nó mestre. Com este comando, é possível visualizar quais nós estão ativos e como eles interagem entre si na rede.

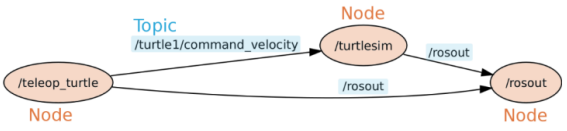

Fig. 1. ROS gráfico computacional do exemplo turtle-sim. Fonte: [19]

4) Bags: *Bag* é um formato de arquivo em ROS para armazenar dados de mensagens ROS. As *Bags*, assim chamadas por causa de sua extensão .bag têm um papel importante no ROS e uma variedade de ferramentas foi desenvolvida para permitir que armazene, processe, analise e visualize-as [20].

5) Rviz: RVIZ é um visualizador tridimensional usado para visualizar robôs, os ambientes em que eles trabalham e dados de sensores. É uma ferramenta altamente configurável, contando com diversos tipos de visualizações e *plugins* [21].

B. Robô

Nesse projeto foi utilizado o robô desenvolvido no trabalho de conclusão de curso [22], e ele consiste em um robô móvel com rodas (RMR) de tração diferencial. O seu controlador consistirá simplesmente da movimentação do RMR com base em comandos de velocidade linear e angular, guiando o robô de uma posição inicial a um dado destino. Ele será realizado com base no controle cinemático. O controle do RMR será feito por intermédio de um controlador embarcado em uma placa Arduino Mega, utilizando-se da linguagem C++ [22].

O RMR desenvolvido utiliza um modo de tração diferencial, essa configuração consiste em um sistema robótico de duas rodas ativas, com um motor em cada uma delas. Esse tipo de tração foi escolhido devido a vantagens quanto a sua mobilidade, uma vez que manobrá-lo é uma tarefa simples e que requer pouco espaço, além de sua simplicidade estrutural e baixo custo de montagem [23], [22]. Ele é mostrado na Fig. 2.

No trabalho [22] também foi desenvolvido um pacote de ROS, chamado skate_bot que permite fazer a leitura da odometria do robô e também da velocidade angular e linear recebida por outro pacote, através do tópico cmd_vel que faz o robô realizar ações. O tópico cmd_vel provém do pacote teleop_twist_joy, que consiste no pacote referente

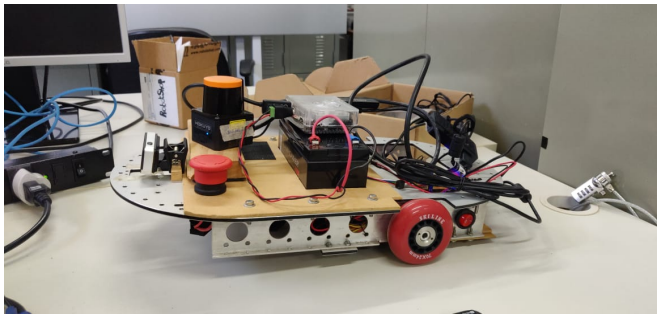

Fig. 2. Robô RMR desenvolvido por [22] com integração de sensores.

ao controle de Xbox One, permitindo que ao utilizá-lo, o controle determine o movimento do robô.

C. GPS

O sistema de posicionamento global GPS, é um sistema de navegação via satélite que possui um aparelho receptor móvel que detecta sua posição, assim como o horário, sob condições atmosféricas, seja qual for o lugar da Terra desde que o receptor se encontre no campo de visão de três ou mais satélites GPS (quanto mais satélites, maior a precisão da medição). Esse componente é muito utilizado em AGVs para navegações *outdoor* para obter sua localização. No entanto, o GPS é afetado pela localização do satélite de modo que, dependendo da posição e distância, pode apresentar baixa precisão e atraso na recepção. A localização GPS é usada principalmente ao ar livre e devem ser evitados para localização *indoor* [24].

O modelo do GPS implementado é o Adafruit Ultimate GPS Breakout demonstrado na Fig.3.

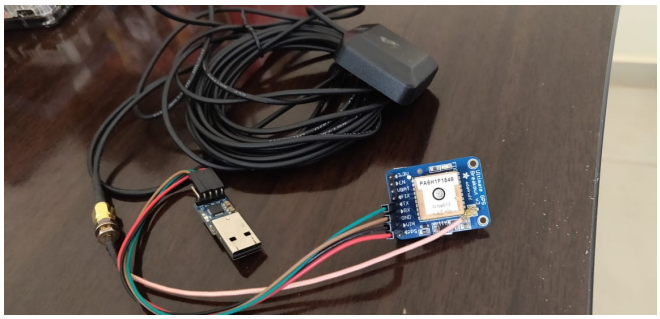

Fig. 3. GPS da Adafruit Ultimate GPS Breakout implementado no trabalho.

D. IMU

A IMU é um equipamento que possui 3 tipos de sensores principais, os quais seria acelerômetro, giroscópio e magnetômetro, sendo que a IMU pode definir se o robô está em movimento, inclinado ou rotacionando, este componente é utilizado em carros, robôs e drones, entre outros projetos [18]. A IMU utilizada nesse projeto é a myAHRS+ withrobot demonstrada na Fig. 4.

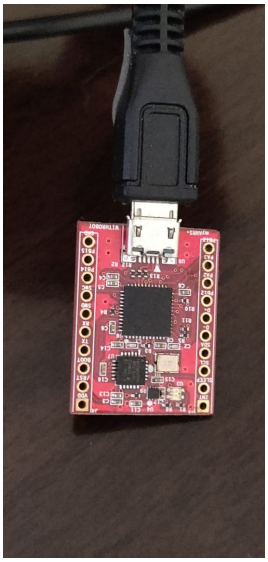

Fig. 4. IMU utilizado no projeto.

E. CÂMERA USB

A câmera de conexão USB para o projeto serviu para ter o campo de visão do robô via SSH, sendo utilizado para ver o obstáculos quais o robô vai se deparar. A câmera utilizada foi a câmera usb do ODROID mostrado na Fig.5.

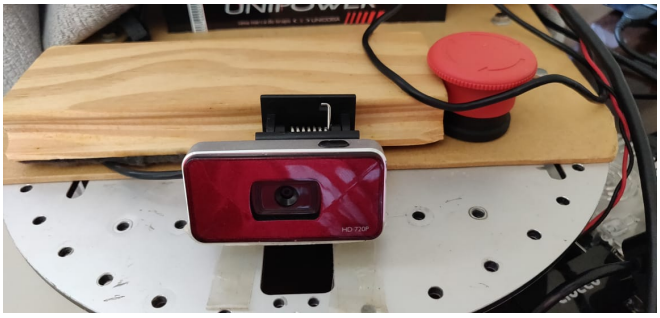

Fig. 5. Câmera ODROID acoplada no robô.

F. Controle Xbox One

Foi utilizado um controle para realizar a movimentação do robô, sendo essa movimentação feita via SSH. E o controle utlizado é um controle de Xbox One, como mostrado na Fig.6.

Fig. 6. Controle Xbox One utilizado para o controle de movimento do robô.

G. Laser LiDAR

No trabalho foi escolhido o uso do sensor *Scanning Lasecrever Range Finder Smart* - URG mini - UST-10LX (UUST003). Atualmente, com os computadores capazes de executar cada vez mais cálculos por segundo, possibilitou-se que sensores a laser LiDAR, que possuem capacidades de medição extremamente precisas, bem como também uma alta taxa de aquisição de dados, facilitaram com que os algoritmos de SLAM (*Simultaneous Localization And Mapping*) se tornassem cada vez mais precisos em suas estimativas. [25]. O modelo utilizado é da marca Hokuyo e é mostrado na Fig.7.

Fig. 7. LiDAR Hokuyo UST-10LX usado no projeto.

H. KINECT

A primeira versão do sensor, desenvolvido pela Microsoft, era um sensor de profundidade de baixo custo para imagens rápidas e de alta qualidade. Seu lançamento causou um grande impacto na robótica, com diversas aplicações utilizando ele como base do projeto. Entretanto, a tecnologia do sensor utilizada na primeira versão era sensível a determinadas condições de incidência solar, tornando o sensor limitado a algumas aplicações internas. Essa versão do sensor se baseia na tecnologia de tempo de voo (*Time of Flight* - ToF), o que permite ao sensor uma maior resolução e um maior campo de visão [26].

O algoritmo utilizado para fazer o SLAM do kinect foi o RTAB-Map (mapeamento baseado em aparência em tempo real, do inglês: *Real-Time Appearance-Based Mapping*), utiliza-se uma biblioteca em C++ multi-plataforma e um pacote para ROS. O algoritmo tem as seguintes premissas, nas quais foram baseadas nos textos [27], [28]:

• Processamento Online: a saída do algoritmo SLAM deve ser limitado ao máximo atraso entre amostras do sensor em questão.

• Odometria robusta e com baixo desvio: embora o laço fechado seja capaz de corrigir a maior parte dos desvios de odometria, nos cenários reais o robô dificilmente é capaz de se localizar corretamente no mapa, seja porque está explorando novos ambientes ou porque há falta de detalhes do cenário. O algoritmo utiliza das medições dos sensores para inferir a odometria corrigindo possíveis desvios de medição.

• Localização robusta: a abordagem SLAM deve ser capaz de reconhecer quanto revisita um local passado (para o laço fechado de detecção) para corrigir o mapa.

• Geração prática de mapa e exploração: as principais abordagens de navegação são baseadas em grades de ocupação, logo, é benéfico o desenvolvimento de abordagem de SLAM que pode fornecer uma grade de ocupação 3D ou 2D pronta para isso, facilitando a integração.

O Equipamento utilizado é demonstrado na Fig.8.

Fig. 8. Kinect Xbox360 implementado no projeto.

I. Computador e o sistema embarcado

No experimento foram utilizados dois computadores, um para processar a informação e outro para controlar o robô. O computador central é um notebook simples, que serve para processar os dados e receber os sinais dos sensores, também é o computador que controla o robô e armazena os dados dos sensores. O outro computador, ODROID, serve para fazer as subscrições dos tópicos dos sensores que possui as informações que seguirá certamente o caminho global ou caminho local. A Fig.9 representa ODROID utilizado.

III. Implementação dos sensores

A. Computador e o sistema embarcado

No trabalho foram utilizadas nos dois computadores versões de Ubuntu iguais evitando, assim, conflitos de *software*, no qual foi utilizada a versão Ubuntu 18.04. O ROS foi utilizado na versão melodic pois na versão kinect apresentou alguns erros de processamento de vídeo e de transmissão de dados do LiDAR.

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

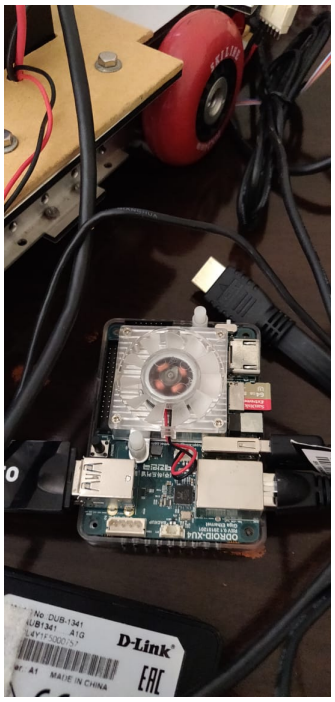

Fig. 9. ODROID XU-4

B. Instalação do pacote skate_bot

O pacote *skate_bot* foi desenvolvido no trabalho final de curso do Vitor Izumino [22], o qual consistiu em realizar o controle para um robô móvel. Para a implementação do pacote ROS skate_bot é necessário utilizar os códigos mostrados abaixo:

```
cd

cp /skate_bot ~/catkin_ws/src/

cd ~/catkin_ws

catkin_make

cd

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

O arquivo skate_bot.rules serve para determinar alguns *input* e *output* para que o funcionamento do controlador via arduino.

C. SSH

O SSH é um protocolo de rede para fazer a comunicação entre servidores. Então, no projeto foi feita a comunicação do computador central com o computador embarcado, realizando também os protocolos que configuram eles como "Mestre" e o computador(ODROID) como "escravo".

O comando mostrado a seguir deve ser utilizado no computador central (*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 definido como IP da conexão *wlan* do computador central, ele é encontrado utilizando o comando *Ifconfig*. Esse comando deve ser modificado pelo valor de IP encontrado com comando *ifconfig*. Para a utilização do sistema como mestre e escravo, esses comandos devem ser incluídos no arquivo bashrc, de forma que não é necessária a repetição do código todas as vezes em que se queira abrir uma nova aba.

No escravo 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
```

Da mesma forma, é utilizado o comando *ifconfig* para encontrar o IP do dispositivo escravo. Os valores de IP substituem então comandos do código acima, de forma que no lugar de "IP_ODROID" é feita a substituição para o IP da máquina correspondente ao escravo e no lugar de "IP_PC_MASTER" é substituído para o valor de IP da máquina correspondente ao dispositivo mestre.

D. Controle de Xbox One

Para a instalação do *joystick* de Xbox One inicialmente utiliza-se o comando que instala o pacote teleop_twist_joy no computador central [29]:

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

Após a instalação, é necessário conectar o *joystick* de Xbox One no computador, e em seguida utilizar o comando:

```
cd /dev/input

ls
```

Utilizando o comando ls é informado para o usuário o valor do *input* referente ao dispositivo, como observado na

Fig.10.

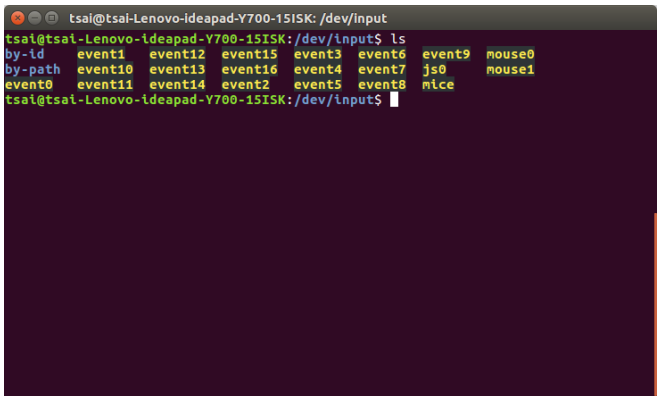

Fig. 10. Tela demonstrando o *input* encotrado js0.

Todos os *joystick*s possuem *inputs* no formato jsX (sendo X é o número do *joystick*). Como observado na Fig. 10, *joystick* utilizado está configurado com *input* js0. Com isso, altera-se o valor do *input* no pacote do teleop_twist_joy utilizando os comandos demonstrados a seguir:

```
roscd teleop_twist_joy/launch/

sudo nano teleop.launch
```

Após a utilização desses comandos, é feita a modificação do *launch* do pacote teleop_twist_joy, conforme mostra a Fig.11.

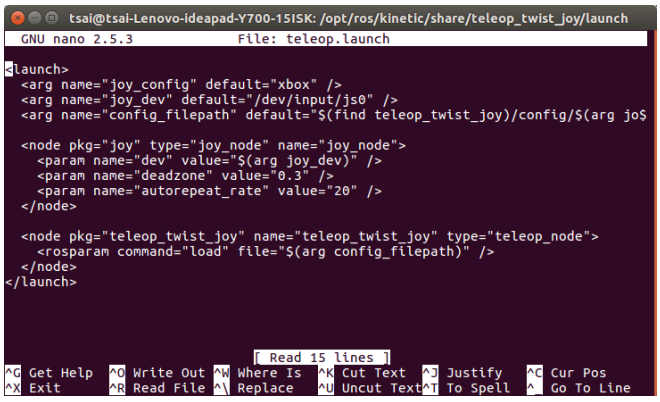

Fig. 11. Alteração do *input* do *launch* do pacote do teleop_twist_joy. na linha "Joy_config".

As linhas do arquivo do *launch* do pacote do teleop_twist_joy que precisam ser alteradas para que o*joystick* consiga ter a configuração e funcionamento adequados, são:

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

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

Adequando para seu tipo de controle, caso haja *script* padrão no pacote utilizado, caso contrário é necessário o desenvolvimento de um novo *script* com os comandos para seu *joystick*. Assim, as linhas de comando vistas anteriormente devem ser alteradas para que o pacote consiga receber os dados do controle.

Após fazer as alterações, o *joystick* está pronto para funcionar. É importante mencionar que o roscore precisa estar ativado para funcionamento do controle. São utilizados os seguintes comandos:

```
rosrun teleop_twist_joy teleop_node

roslaunch teleop_twist_joy teleop.launch
```

Em seguida, para teste do pacote do controle:

```
Rostopic echo /joy
```

Esse comando demonstra os estados de todos os botões do controle, como mostrado na Fig.12. Para esse *joystick* conforme é pressionado um botão, seu estado apresenta o valor 1, e valor 0 quando o contrário.

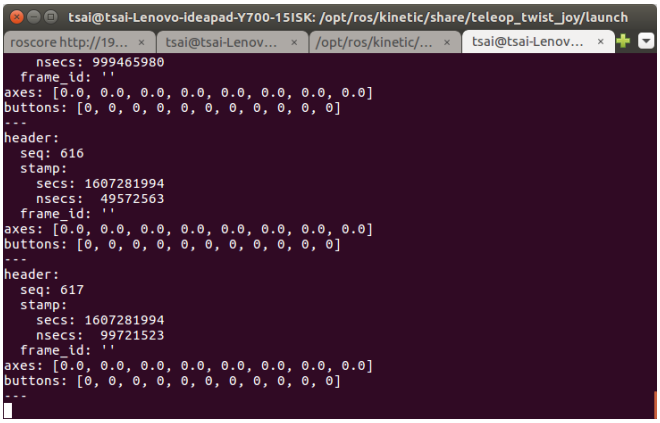

Fig. 12. Demonstrativo do funcionamento do botão do teleop_twist_joy.

Após o funcionamento do teleop_twist_joy, para a demonstração das comunicações entre os dispositivos do sistema, utilizamos o comando abaixo:

```
rqt_graph
```

Conforme a Fig.13, são mostradas as ligações realizadas do pacote teleop_twist_joy, ele indica o caminho percorrido pelos dados enviados pelo controle para realizar a movimentação do robô.

E. Sensor IMU

Para o início da implementação da IMU são utilizados os comandos do pacote myahrs-driver [30]:

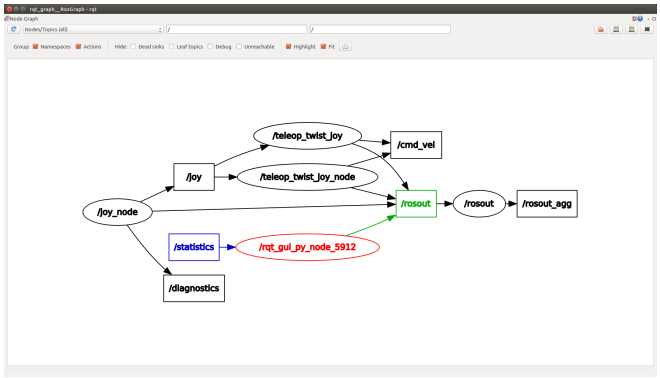

Fig. 13. Rqt_graph teleop_twist_joy.

```
sudo apt-get install ros-melodic-myahrs-driver
```

A partir da instalação do pacote myahrs-driver, inserir via porta USB o sensor IMU, assim como foi feito no controle joy para identificar o *input* correspondente do sensor IMU, utilizando os seguintes comando:

```
cd /dev

ls
```

Observe que o *input* deste senor tem padrão como tty-ACMx (sendo X é um número inteiro do *input*), que pode ser verificado na Fig.14. O dispositivo utilizado no projeto foi identificando como ttyACM0.

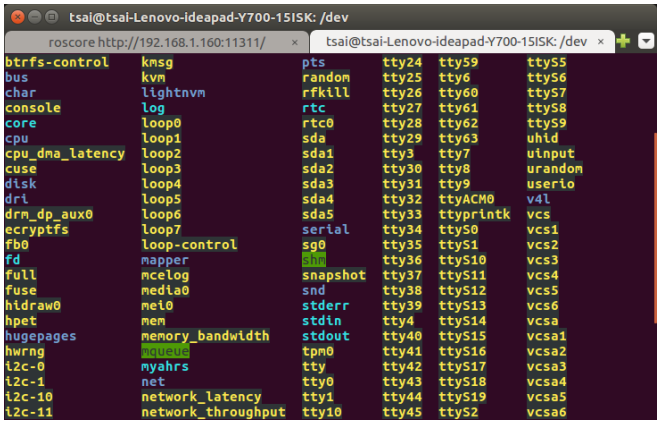

Fig. 14. *Input* do IMU demonstrado, ttyACM0.

Após essa definição, as linhas do código deve ser modificada de acordo com o observado, como mostrado abaixo.

```
rosrun myahrs_driver ...
myahrs_driver _port:=/dev/ttyACM0
```

Após isso, mostramos os dados do pacote através do comando:

```
rostopic echo /imu/data
```

O resultado desse comando é demonstrado conforme na Fig.15.

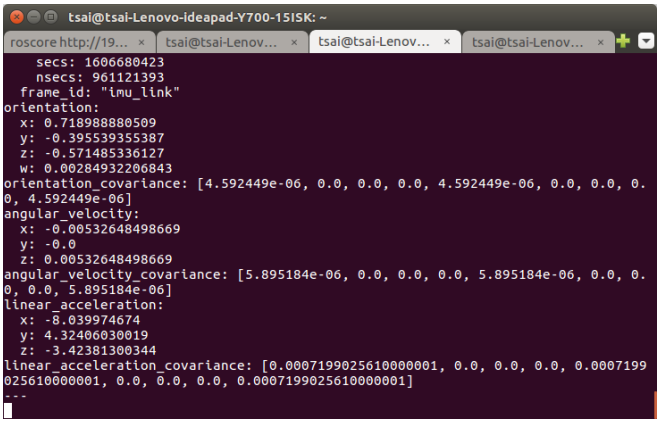

Fig. 15. Dados coletados da IMU mostrando os valores dos eixos de rotação.

A Fig.16 demonstra a ligações do pacote através do comando rqt_graph.

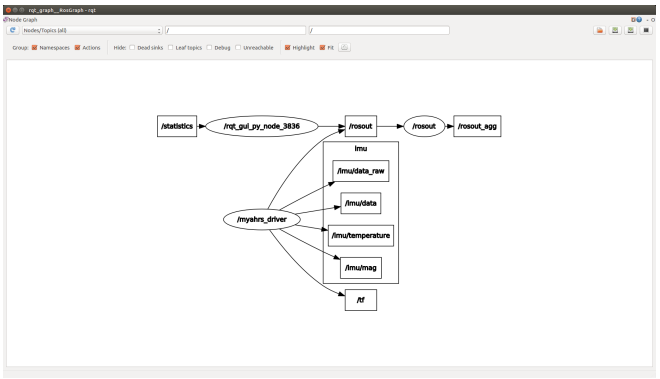

Fig. 16. Rqt_graph das conexões da IMU.

F. Sensor GPS

São utilizados os seguintes comandos, que instalam os pacotes nmea-navsat-driver [31] e gpsd-clients [32]:

```
sudo apt-get ...
install ros-melodic-nmea-navsat-driver

sudo apt-get install gpsd-clients
```

Após a instalação dos pacotes, são utilizados os *scripts* desenvolvidos em [22], que devem ser realocados na pasta do pacote nmea-navsat-driver, através do comando a seguir:

```
roscd nmea_navsat_driver/scripts/
```

É necessário também transformar os arquivos mtk3339_config.py e gpsdConfigMTK.sh para o tipo executável, para isso utilizou-se os seguintes códigos:

```
sudo chmod +x mtk3339_config.py
sudo chmod +x gpsdConfigMTK.sh
```

Após transformar em executável, os arquivos estão pronto para funcionamento. Assim como visto nos dispositivos apresentados anteriormente, é necessário verificar o *input* referente ao GPS, o qual é apresentado no formato cp210x.

O pacote é inicializado pelo comando:

```
rosrun nmea_navsat_driver gpsdConfigMTK.sh
```

O resultado do comando com o funcionamento do GPS, com todas as informações entregues por ele, pode ser observado na Fig. 17.

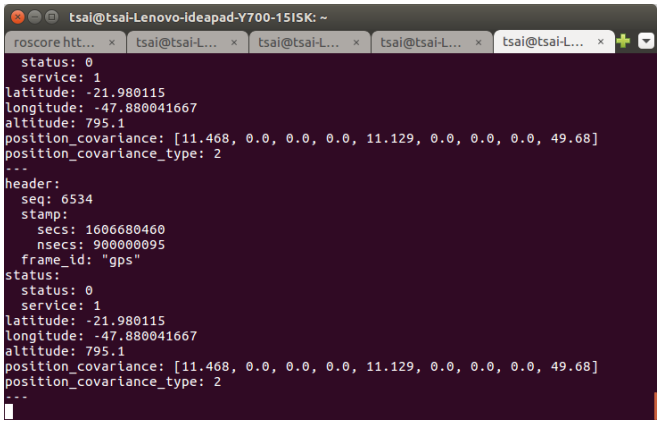

Fig. 17. Demonstrativo do funcionamento do GPS com as informações coletadas.

Novamente, na Fig. 18 são mostradas as ligações do pacote utilizando o comando rqt_graph.

G. Camera USB

Para implementação da câmera USB é utilizado o comando abaixo, o qual instala o pacote usb-cam [33]:

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

Igualmente aos outros pacotes, é necessário identificar o *input* da câmera, no qual tem o formato de videoX (sendo X é um número inteiro variável). É alterado no *launch* do pacote da câmera o parâmetro conforme mostrado a seguir:

```
<param name="video_device" ...
```

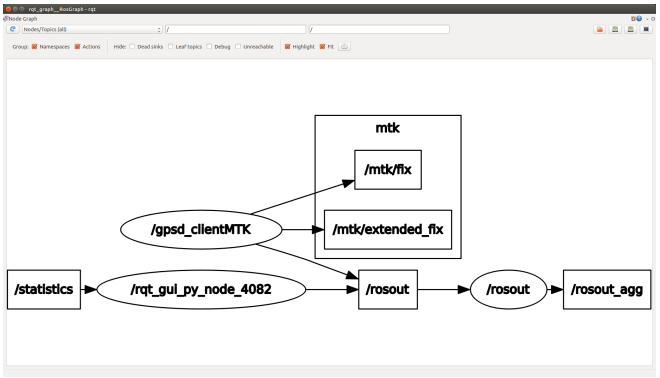

Fig. 18. Gráfico das conexões do GPS.

```
value="/dev/video1" />
```

Esses parâmetros são encontrados no arquivo usb_cam-test.launch que está localizado na pasta usb_cam/launch, abaixo um exemplo de como localizar a pasta em questão.

```
roscd usb_cam/launch

sudo nano usb_cam-test.launch
```

Após alteração do pacote da câmera, a sua visualização é realizada utilizando o comando:

```
roslaunch usb_cam usb_cam-test.launch

rviz rviz
```

A visualização da imagem da câmera é mostrada na Fig.19.

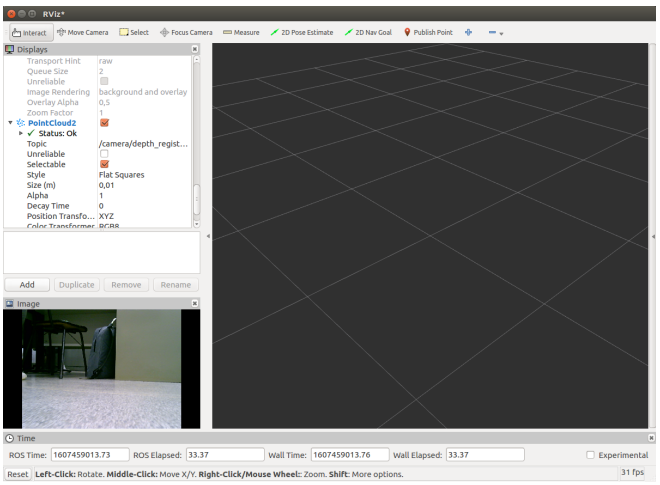

Fig. 19. Câmera USB sendo visualizada no Rviz.

H. Laser (LIDAR)

A implementação do Laser precisa ser feito algumas alterações na rede de internet, pois o modelo Hokuyo utiliza conexão via cabo ethernet. Com isso, foi preciso deixar as opções de ajustes conforme mostrado nas figuras 20 e 21. Com a conexão via SSH, é necessário fazer esses ajustes possibilitando o funcionamento do laser e do Wi-Fi de forma simultânea.

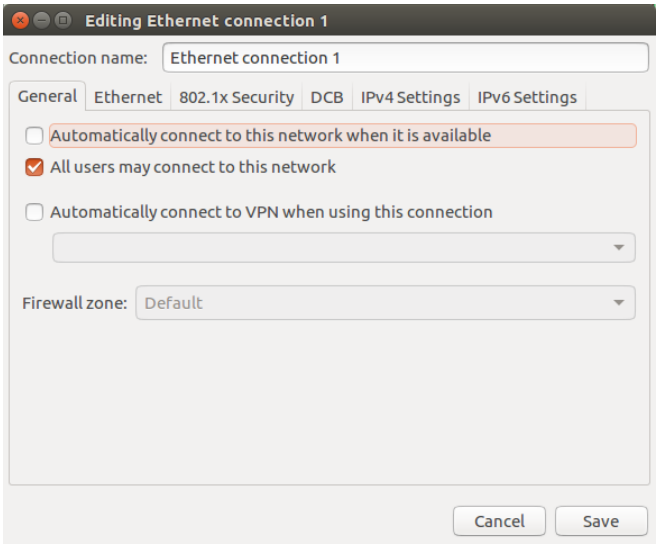

Fig. 20. Edição da Conexão Ethernet deixando não selecionado o opção automática de conexão.

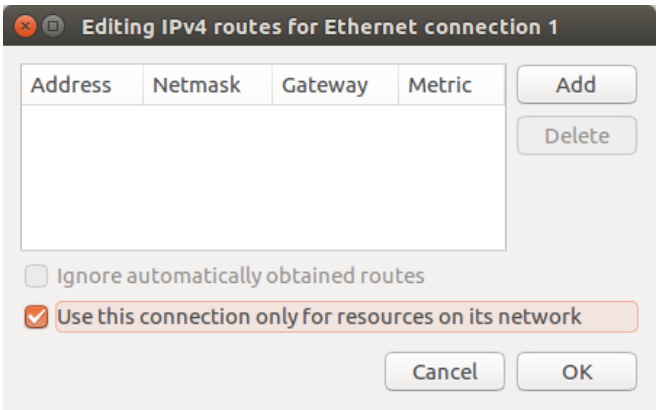

Fig. 21. Edição das Rotas IPV4 selecionando o uso conexão apenas para o recurso de rede

Após essa alteração, é necessária a instalação do pacote do urg_node, para a utilização do laser [34].

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

Com a instalação do pacote é necessário verificar o IP do seu laser no manual do fabricante, no caso deste projeto, o laser utilizado possui o IP 192.168.1.15. Com isso, é feita uma nova configuração da ethernet para o IP com o mesmo caminho porém com canal diferente, por isso configura-se a ethernet como mostrado na Fig. 22, alterando a configuração no IPV4 address para 192.168.1.10, *mask* para 255.255.255.0 e *gateway* para 192.168.1.255.

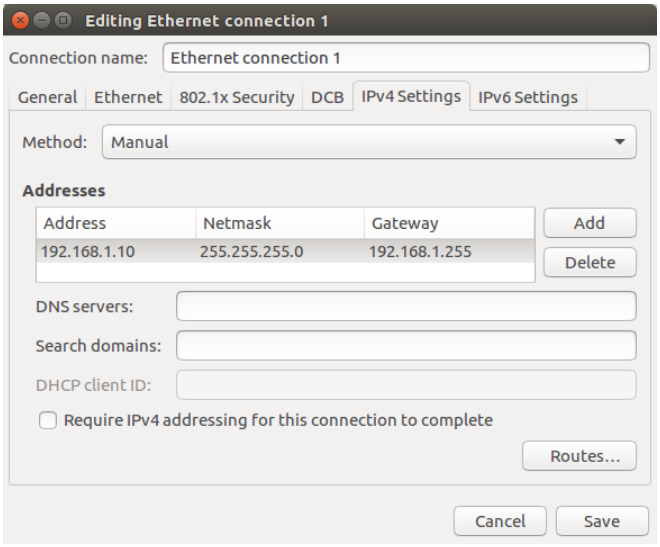

Fig. 22. Configuração adequada na conexão ethernet para o laser.

Após essa configuração, é preciso executar o seguinte código para funcionamento do laser:

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

Depois é necessário utilizar o Rviz trocando o *fixed frame* para laser e adicionar o tópico laser_scan, como mostrado na Fig. 23.

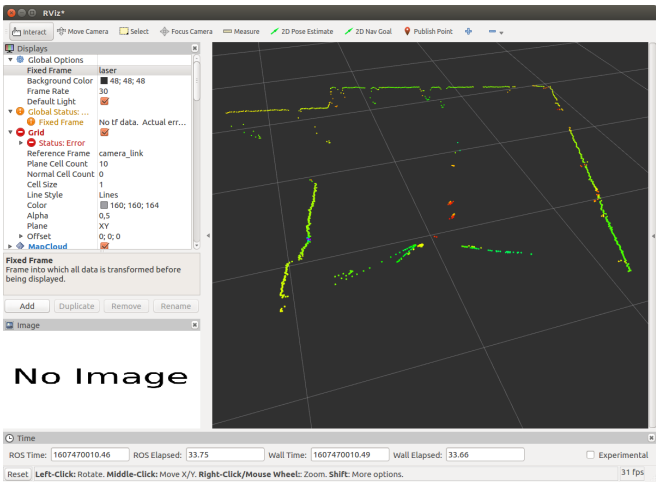

Fig. 23. Funcionamento do mapeamento do laser.

A sala na qual o mapeamento está sendo executado é mostrada na Fig. 24.

Fig. 24. Demonstrativo da sala utilizada para funcionamento do laser.

I. *Kinect Xbox 360*

A implementação do kinect Xbox 360 precisa instalar dois pacotes de ROS, freenect-launch [35] e rtabmap [36].

```
sudo apt-get install ...
ros-melodic-freenect-launch

sudo apt-get install ros-melodic-rtabmap
```

O pacote freenect faz a parte de configuração dos parâmetros do Kinect Xbox 360 enquanto que o pacote rtabmap serve para fazer SLAM 3D, mostrando assim o funcionamento do laser do kinetic. Utilizando o comandos a seguir, podemos ver seu funcionamento.

```
roslaunch freenect_launch ...
freenect.launch depth_registration:=true

roslaunch rtabmap_ros rtabmap.launch
```

É observado o funcionamento do laser na própria interface gráfica que o rtabmap oferece na Fig. 25.

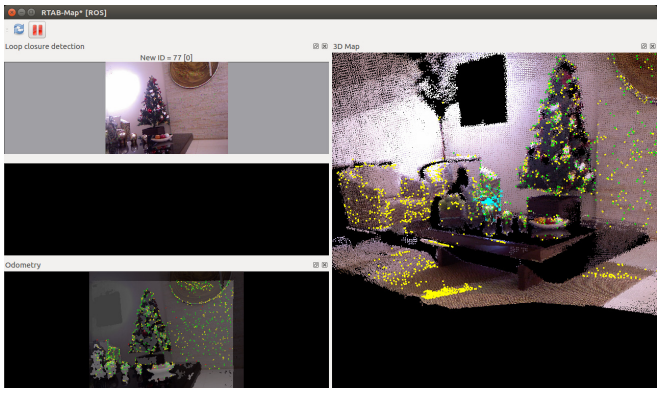

Fig. 25. Mapeamento 3D do kinect Xbox 360 com pacote rtabmap.

A sala analisada é demonstrada na Fig.26.

IV. Resultados e Discussão

O experimento consistia em fazer as implementações de diversos sensores em um robô móvel demonstrada na Fig.27

Fig. 26. Sala analisada do Kinect Xbox 360.

e Fig.28. Ele se mostrou funcional verificou-se todos os dados via *rostopic echo* e para facilitar as análises dos dados, foram feitos os ROS BAGs, os quais já foram citados anteriormente servindo para armazenamento de dados das subscrições nas quais se pretendem fazer análises demonstrado na Fig.29. Nota-se que o ODROID-XU4 possui algumas dificuldades, como problemas de não ter como executar o mapeamento com uso do Kinect Xbox 360, pois devido as baixas taxas de quadros por segundo se torna inviável, mesmo via comunicação escravo, os dados da subscrições ficam incorretos. Verificou-se também que com a utilização do GPS selecionado em um ambiente indoor não foi possível obter seus dados, os dados recebidos ficam descritos como ”not a number” em virtude disso, necessitando de um ambiente externo para o seu correto funcionamento.

V. Conclusão

Nesse estudo, foram feitas integrações dos sensores IMU, Câmera USB, GPS, laser LiDAR, Kinect Xbox 360 com a plataforma ROS, mostrando o funcionamento dos pacotes myahrs-driver, nmea_navsat_driver, usb_cam, teleop_twist_joy, skate_bot, urg_node, freenect, rtabmap, obtendo um resultado satisfatório obtendo os dados desejados. E com isso mostra-se que com estudos e entendimento melhor sobre sistema operacional Linux e dos frameworks do ROS, a integração dos sensores se mostram mais automáticos repetindo as primeiras etapas para fazer a implementação. Objetivo inicial desse trabalho é fazer um artigo que facilita os futuros trabalhos que utilizarão os sensores implementados, em virtude disso buscou-se a forma mais simples e completo para se entender e fazer as implementações. Ao final do trabalho se demonstrou-se quase um tutorial como fazer a implementação dos sensores, podendo fazer as implementações independentes, escolhendo apenas os sensores quais trabalhos futuros vão utilizar.

Agradecimentos

A meus pais, que me incentivaram e deram oportunidade aos estudos.

Ao professor Roberto dos Santos Inoue, pela oportunidade de por me orientar e supervisionar neste trabalho

Aos amigos que conheci durante esses anos de graduação e que estiveram presentes nos altos e baixos por todos esses anos.

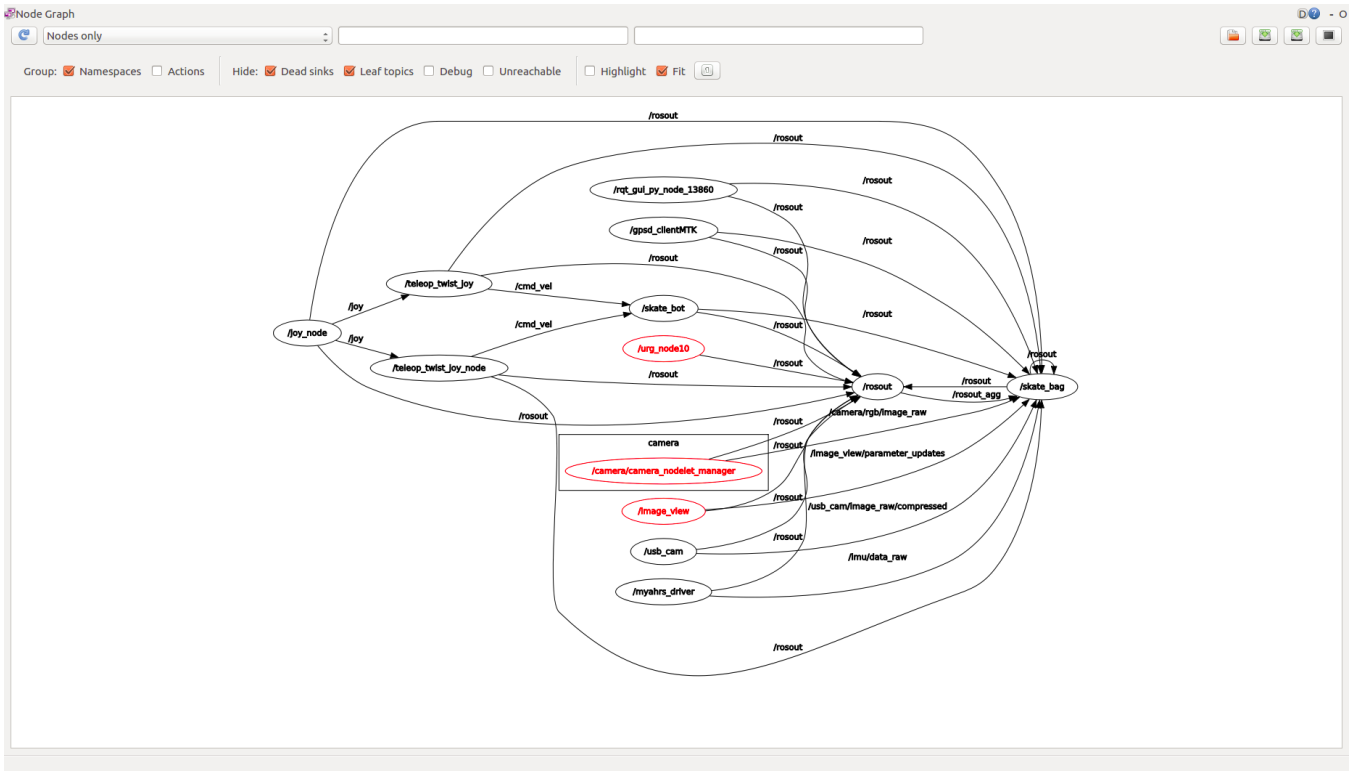

Fig. 27. Rqt_graph parte 1 - Sem nodes do kinect.

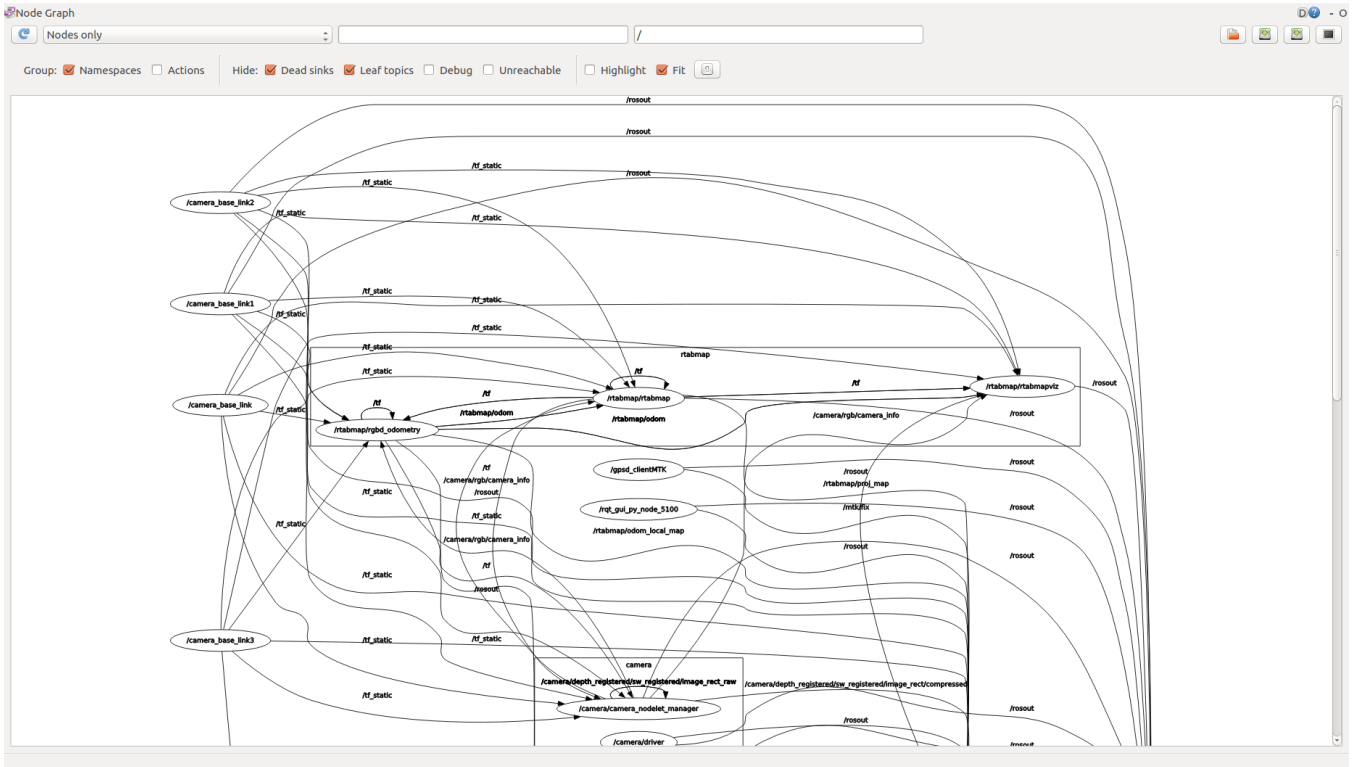

Fig. 28. Rqt_graph parte 2 - Alguns Nodes do kinect.

REFERENCES

[1] Lasi, Heiner, Fettke, Peter, Kemper, Hans-Georg, Feld, Thomas, Hoffmann, and Michael, "Industry 4.0," *Business Information Systems Engineering: The International Journal of wirtschaftsinformatik,6,issue 4*, pp. 239–242, 2014.

[2] E. A. Lee, "Cyber physical systems: Design challenges," *EECS De-*

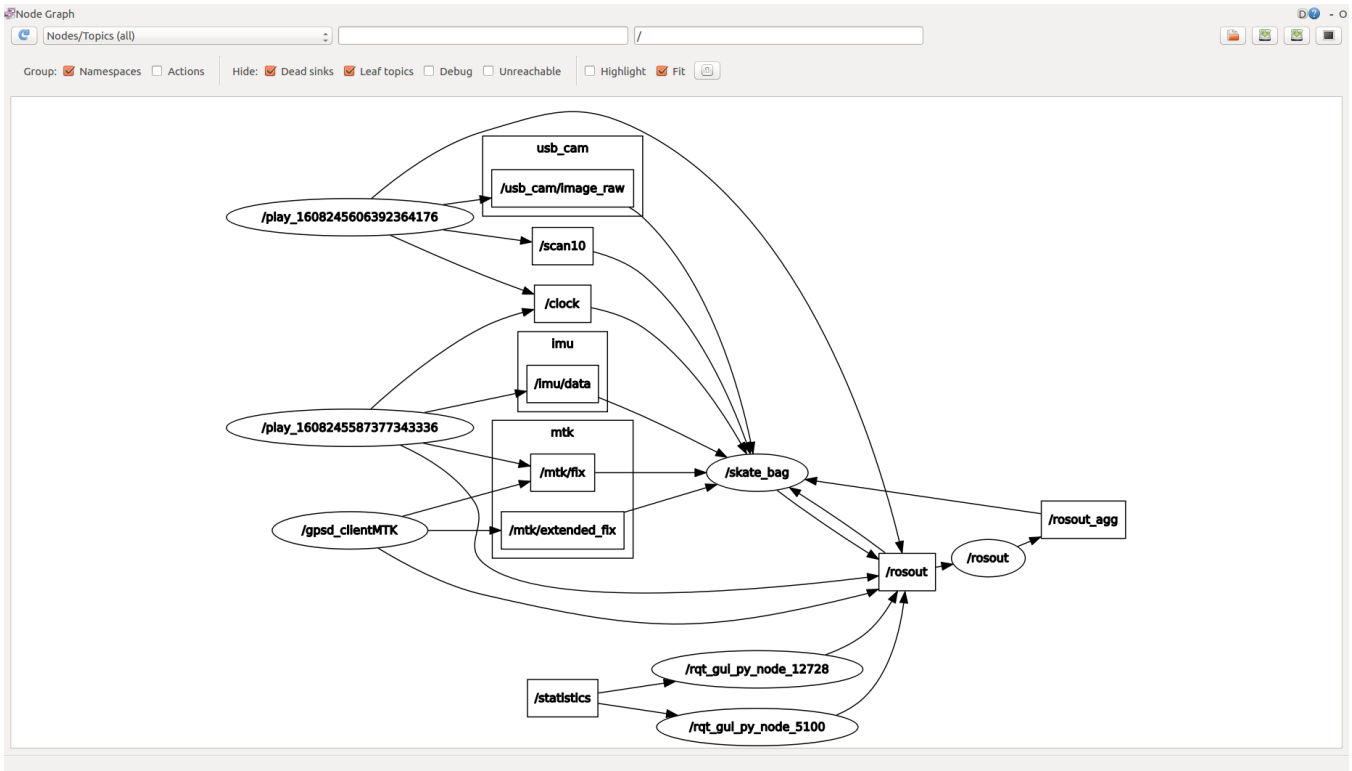

Fig. 29. Rqt_graph do BAG.

partment, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-8, 2008.

[3] C. Alippi, "Intelligence for embedded systems: A methodological approach," *Springer Publishing Company, Incorporated*, 2014.

[4] A. M. F. Ferracuti, A. Freddi and M. Prist, "An integrated simulation module for cyber-physical automation systems," *Sensors*, 2016.

[5] M. Z. A. Fellan, C. Schellenberger and H. D. Schotten, "Enabling communication technologies for automated unmanned vehicles in industry 4.0," *9th Int. Conf.Inf. Commun. Technol. Converg. ICT Converg. Powered bySmart Intell. ICTC 2018*, pp. 171–176, 2018.

[6] F. Okumuş and A. F. Kocamaz, "Cloud based indoor navigation for ros-enabled automated guided vehicles," *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 2019.

[7] S. Horvath and H. Neuner, "System identification of a robot arm with extended kalman filter and artificial neural networks," *Journal of Applied Geodesy, vol. 13, no. 2*, pp. 135–150, 2019.

[8] G. S. Rao, D. Siripurapu, and L. Bagadi, "Elevation and position uncertainty based kf model for position accuracy improvement," *Procedia Computer Science, vol. 143*, pp. 914–920, 2018.

[9] S. THRUN, "Toward a framework for human-robot interaction," *Human-Computer Interaction*, pp. 9–24, 2004.

[10] H. SECCHI, "Una introducción a los robots móviles." *Universidade Nacional de San Juan - UNSJ*, pp. 80–82, 2008.

[11] "About ros," 2020, acessado: 2020-12-18. [Online]. Available: http://www.ros.org/about-ros

[12] S. ANDO, "Intelligent three-dimensional world sensor with eyes and ears." *Handbook of Sensors and Actuators. [S.l.]: Elsevier,*, pp. 117–152, 1996.

[13] T. BAILEY and DURRANT-WHYTE.H, "Simultaneous localization and mapping (slam)." *part i. IEEE Robotics Automation Magazine, v. 13, n. 3*, pp. 108–117, 2006.

[14] M. QUIGLEY, "Ros: an open-source robot operating system." *ICRA workshop on open source software, [S.l.]*, p. 5, 2009.

[15] S. KOHLBRECHER, "A flexible and scalable slam system with full 3d motion estimation." *In: IEEE. Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR), [S.l.]*, 2011.

[16] M. G. OCANDO, "Autonomous 2d slam and 3d mapping of an environment using a single 2d lidar and ros." *Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR).*, pp. 1–6, 2017.

[17] M. KöSEOğLU and O. ÇELIK, O. M.and PEKTAş, "Design of an autonomous mobile robot based on ros." *International Artificial Intelligence and Data Processing Symposium (IDAP). [S.l.: s.n.]*, pp. 1–5, 2017.

[18] R. G. da Silva, "Obstacle detection and avoidance for a mobile robot." *Federal University of São Carlos– UFSCar Center for Exact Sciences and Technology– CCET.*, 2020.

[19] "Understanding topics," 2020, acessado: 2020-12-18. [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Tutorials(2f)UnderstandingTopics.html

[20] "Bags," 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/Bags

[21] "Rviz," 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/rviz

[22] 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*, 2017.

[23] A. GIL and D. Romaguera, "Isrobot: Robot móvel de tracção diferencial." *Universitat Politècnica de Catalunya*, 2007.

[24] W. Jin, J. Qi, and N. Wu, "State estimation of unmanned vehicles in gps restricted environment," *2019 Chinese Automation Congress (CAC)*, 2019.

[25] P. AGRAWAL, "Pce-slam: A real-time simultaneous localization and mapping using lidar data," *2017 IEEE Intelligent Vehicles Symposium (IV). [S.l.: s.n.]*, pp. 1752–1757, 2017.

[26] P. Fankhauser, "Kinect v2 for mobile robot navigation: Evaluation and modeling," *2015 International Conference on Advanced Robotics (ICAR). [S.l.: s.n.]*, pp. 388–394, 2015.

[27] M. LABBE and F. MICHAUD, "Appearance-based loop closure detection for online large-scale and long-term operation." *IEEE Transactions on Robotics, IEEE, v. 29, n. 3.*, pp. 734–745, 2013.

[28] F. Q. Bordon, D. D. Goulart, and M. H. Terra, "Sistema de mapeamento robótico para corpos móveis com redundância sensorial." *Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos*, 2020.

[29] "Teleop_twist_joy," 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/teleop_twist_joy

[30] “Myahrs_driver,” 2020, acessado: 2020-12-18. [Online]. Available: https://github.com/robotpilot/myahrs_driver
[31] “Nmea_navsat_driver,” 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/nmea_navsat_driver
[32] “Gpsd_client,” 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/gpsd_client
[33] “Usb_cam,” 2020, acessado: 2020-12-18. [Online]. Available: https://wiki.ros.org/usb_cam
[34] “Urg_node,” 2020, acessado: 2020-12-18. [Online]. Available: https://wiki.ros.org/urg_node
[35] “Freenect_launch,” 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/freenect_launch
[36] “Rtabmap_ros,” 2020, acessado: 2020-12-18. [Online]. Available: http://wiki.ros.org/rtabmap_ros