

## 4 VALIDAÇÃO (Capítulo extraído do TCC - Ferramenta para aprendizagem de sistemas Ciber Físicos: Uma aplicação inserida no conceito de Gêmeo Digital na Robótica Móvel)

Este capítulo visa validar a proposta DTMR, verificando os resultados obtidos pertinentes ao desenvolvimento deste projeto, embasado nos conceitos de Indústria 4.0, CPS, robótica móvel e DT.

### 4.1 VALIDAÇÃO DO DTMR TUPY

Esta seção consiste em realizar a validação da aplicação do DTMR no ambiente ciber-físico. Dessa forma, será possível verificar questões pertinentes ao controle da movimentação, as imagens geradas pelo Kinect no *desktop*, os sensores ultrassônicos, bem como, conceber o DT, através da plataforma virtual Coppelia-Sim.

#### 4.1.1 Ambiente ROS

Finalizado as etapas de integração (*hardware* e *software*), inicia-se o ambiente do *framework* ROS. Primeiramente, é definido que o *desktop* será o ROS MASTER.

##### 4.1.1.1 Roscore

Este comando é utilizado para criar uma conexão entre os nós, definindo um ROS MASTER. Sem iniciá-lo, fica impossível estabelecer uma conexão ROS, sendo que só pode ser executado em uma das estações via terminal. Logo, digita-se **roscore** no terminal para iniciar a interface (Figura 108).

Figura 108: Iniciando Roscore no Terminal *desktop*.

```

roscore http://desktop:11311/
roscore http://desktop:11311/ 80x24
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://desktop:39267/
ros_comm version 1.14.13

SUMMARY
=====

PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES

auto-starting new master
process[roscout-1]: started with pid [4981]
ROS_MASTER_URI=http://desktop:11311/

setting /run_id to 3ca8676e-6866-11ed-867c-a86badd9a93
process[roscout-1]: started with pid [4995]
started core service [/roscout]

```

Fonte: O Autor (2022).

#### 4.1.1.2 Arduino

Após iniciar o **roscore**, deve-se realizar o login no robô Tupy via **SSH**. Inicia-se o procedimento para configuração da roserial-arduino. Primeiramente, deve-se habilitar a interface USB, geralmente denominada **ACM\***. Essa configuração deve ser realizada, pois, devido à segurança do Linux, deve-se habilitar essa porta para estabelecer a comunicação com dispositivos externos. Então, os comandos da Figura 109 foram inseridos no terminal para realizar essa parametrização.

Figura 109: Configurando Rosserial-Arduino no Terminal.

```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x24
jean@desktop:~$ ssh tupy@192.168.15.22
tupy@192.168.15.22's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1033-raspi2 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

496 packages can be updated.
363 updates are security updates.

Last login: Tue Nov 15 21:04:27 2022 from 192.168.15.21
tupy@tupy-robot:~$ ls -l /dev/ttyACM*
crw-rw-rw- 1 root dialout 166, 0 nov 15 21:04 /dev/ttyACM0
tupy@tupy-robot:~$ sudo usermod -a -G dialout tupy
[sudo] password for tupy:
tupy@tupy-robot:~$ sudo chmod a+rw /dev/ttyACM0
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

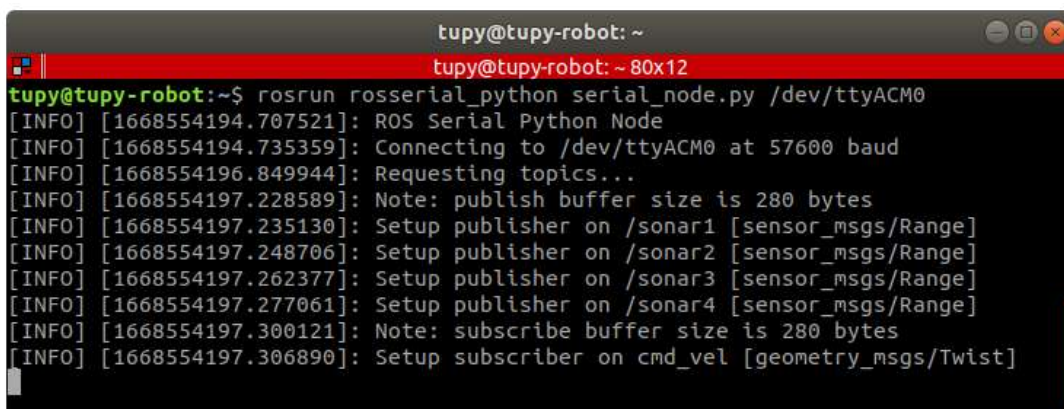
Após a conexão estabelecida do *desktop* com o robô Tupy, basta executar o comando:

- **export ROS\_MASTER\_URI=http://192.168.15.21:11311**
- **export ROS\_IP=192.168.15.22**
- **roslaunch roserial\_python serial\_node.py /dev/ttyACM0**

Como boas práticas de integração com o ROS, é sempre recomendado que seja exportado em qual dispositivo o ROS MASTER está sendo executado e o *Internet Protocol* – Protocolo de Internet (IP) do dispositivo atual. Esses códigos devem ser utilizados toda vez que alguma aplicação relacionada ao ROS for iniciada. No caso, o MASTER é o *desktop*, já o ROS IP pertence ao robô Tupy.

Em seguida, é possível notar que o comando **roslaunch** cria o **ROS Serial Python NODE**. Dessa forma, o ROS executa o *script* da conexão serial do Arduino na porta **tttyACM0**, publicando os Tópicos dos sonares (1, 2, 3 e 4) e inscrevendo as mensagens do **cmd\_vel**, conforme Figura 110.

Figura 110: Iniciando Rosserial-Arduino no Terminal.

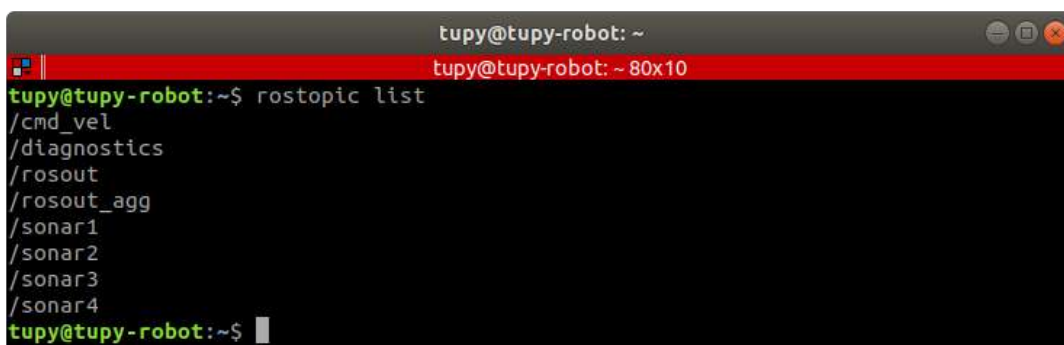


```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x12
tupy@tupy-robot:~$ roslaunch roserial_python serial_node.py /dev/ttyACM0
[INFO] [1668554194.707521]: ROS Serial Python Node
[INFO] [1668554194.735359]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1668554196.849944]: Requesting topics...
[INFO] [1668554197.228589]: Note: publish buffer size is 280 bytes
[INFO] [1668554197.235130]: Setup publisher on /sonar1 [sensor_msgs/Range]
[INFO] [1668554197.248706]: Setup publisher on /sonar2 [sensor_msgs/Range]
[INFO] [1668554197.262377]: Setup publisher on /sonar3 [sensor_msgs/Range]
[INFO] [1668554197.277061]: Setup publisher on /sonar4 [sensor_msgs/Range]
[INFO] [1668554197.300121]: Note: subscribe buffer size is 280 bytes
[INFO] [1668554197.306890]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
```

Fonte: O Autor (2022).

Para verificar se o ROS está identificando os Tópicos ativos, o comando **rostopic list** pode ser digitado para verificar essa informação, conforme Figura 108.

Figura 111: Rostopic List.



```
tupy@tupy-robot: ~
tupy@tupy-robot: ~ 80x10
tupy@tupy-robot:~$ rostopic list
/cmd_vel
/diagnostics
/rosout
/rosout_agg
/sonar1
/sonar2
/sonar3
/sonar4
tupy@tupy-robot:~$
```

Fonte: O Autor (2022).

A partir de então, utilizado a função **echo**, pode-se “ouvir” as informações que estão em cada tópico. Então, digitam-se os seguintes comandos em terminais separados:

- **rostopic echo /sonar1**
- **rostopic echo /sonar2**
- **rostopic echo /sonar3**
- **rostopic echo /sonar4**

Com isso, tem-se a visualização dos Tópicos (Figura 112) das distâncias lidas pelos sensores ultrassônicos, conforme programação do Arduino.

Figura 112: Echo 4 Sonares.

The image shows four terminal windows, each displaying the output of a different sonar sensor. Each window has a title bar indicating the user is 'tupy' on a 'tupy-robot' machine. The data is organized into two groups of four lines each, separated by a dashed line. Each group starts with a 'header:' line, followed by 'seq:', 'stamp:', 'secs:', 'nsecs:', and 'frame\_id:'. The 'radiation\_type:' is 0, 'field\_of\_view:' is 0.259, 'min\_range:' is 0.0, and 'max\_range:' is 2.0. The 'range:' value varies between the sensors and the two groups. The first group of data is for 'sonar1', 'sonar2', 'sonar3', and 'sonar4' respectively. The second group is for 'sonar1', 'sonar2', 'sonar3', and 'sonar4' respectively.

```

tupy@tupy-robot: ~
tupy@tupy-robot: ~ 20x30
tupy@tupy-robot: ~ 20x30
tupy@tupy-robot: ~ 20x30

---
header:
  seq: 3659
  stamp:
    secs: 1668554685
    nsecs: 781625892
  frame_id: "/sonar1"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 154.0
---
header:
  seq: 3660
  stamp:
    secs: 1668554685
    nsecs: 846625892
  frame_id: "/sonar1"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 154.0
---

---
header:
  seq: 569
  stamp:
    secs: 1668554685
    nsecs: 781625892
  frame_id: "/sonar2"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 149.0
---
header:
  seq: 570
  stamp:
    secs: 1668554685
    nsecs: 846625892
  frame_id: "/sonar2"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 151.0
---

---
header:
  seq: 467
  stamp:
    secs: 1668554685
    nsecs: 781625892
  frame_id: "/sonar3"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 57.0
---
header:
  seq: 468
  stamp:
    secs: 1668554685
    nsecs: 846625892
  frame_id: "/sonar3"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 57.0
---

---
header:
  seq: 277
  stamp:
    secs: 1668554685
    nsecs: 714625892
  frame_id: "/sonar4"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 117.0
---
header:
  seq: 278
  stamp:
    secs: 1668554685
    nsecs: 781625892
  frame_id: "/sonar4"
"
radiation_type: 0
field_of_view: 0.259
999990463
min_range: 0.0
max_range: 2.0
range: 124.0
---

```

Fonte: O Autor (2022).

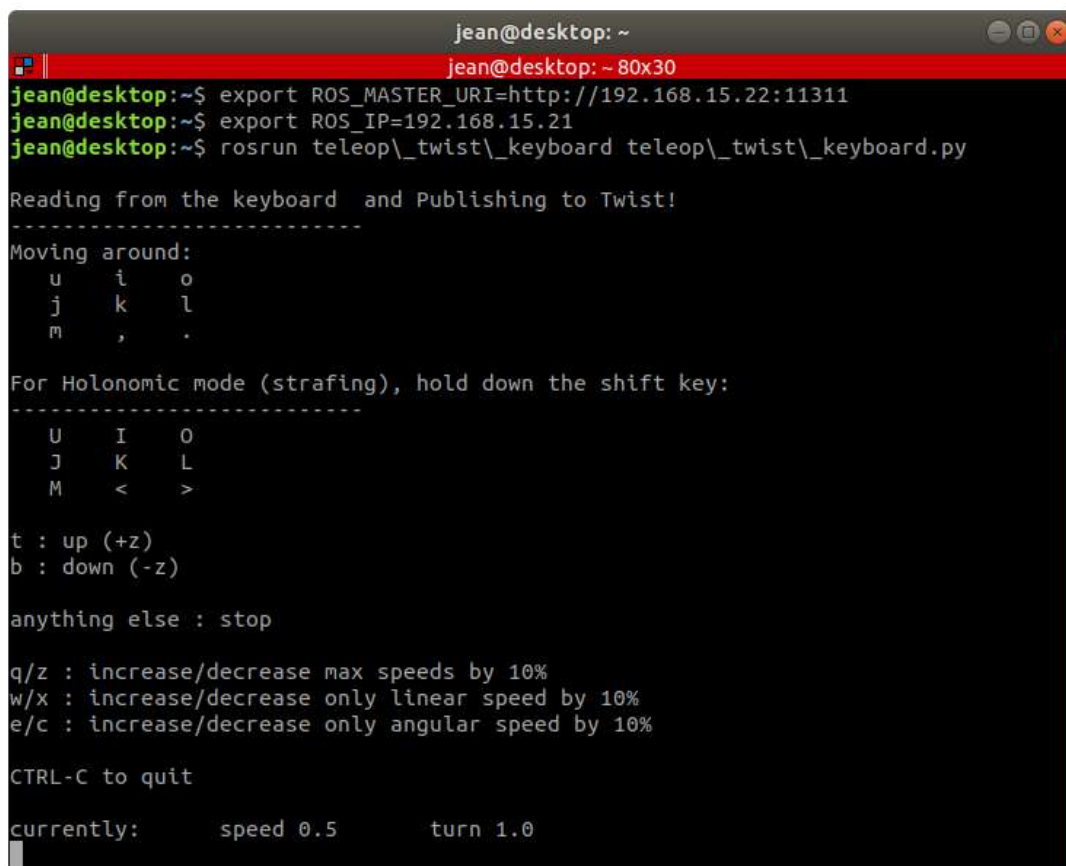
#### 4.1.1.3 Teleoperação

Aqui será implementado o controle do robô físico e virtual simultaneamente com o uso do **cmd\_vel**. Para iniciar a interface deve digitar-se as seguintes linhas de comando no terminal:

- **export ROS\_MASTER\_URI=http://192.168.15.21:11311**
- **export ROS\_IP=192.168.15.22**
- **roslaunch teleop\_twist\_keyboard teleop\_twist\_keyboard.py**

Como as interfaces de inscrição deste foi iniciada anteriormente, o pacote então publica as mensagens vetoriais conforme a Figura 113.

Figura 113: Iniciando Teleoperação.



```

jean@desktop: ~
jean@desktop: ~ 80x30
jean@desktop:~$ export ROS_MASTER_URI=http://192.168.15.22:11311
jean@desktop:~$ export ROS_IP=192.168.15.21
jean@desktop:~$ roslaunch teleop_twist_keyboard teleop_twist_keyboard.py

Reading from the keyboard and Publishing to Twist!
-----
Moving around:
   u   i   o
   j   k   l
   m   ,   .

For Holonomic mode (strafing), hold down the shift key:
-----
   U   I   O
   J   K   L
   M   <   >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

CTRL-C to quit

currently:      speed 0.5      turn 1.0

```

Fonte: O Autor (2022).

A partir de então, conforme a captura das teclas, o robô é movimentado com velocidades linear e angular definidas.

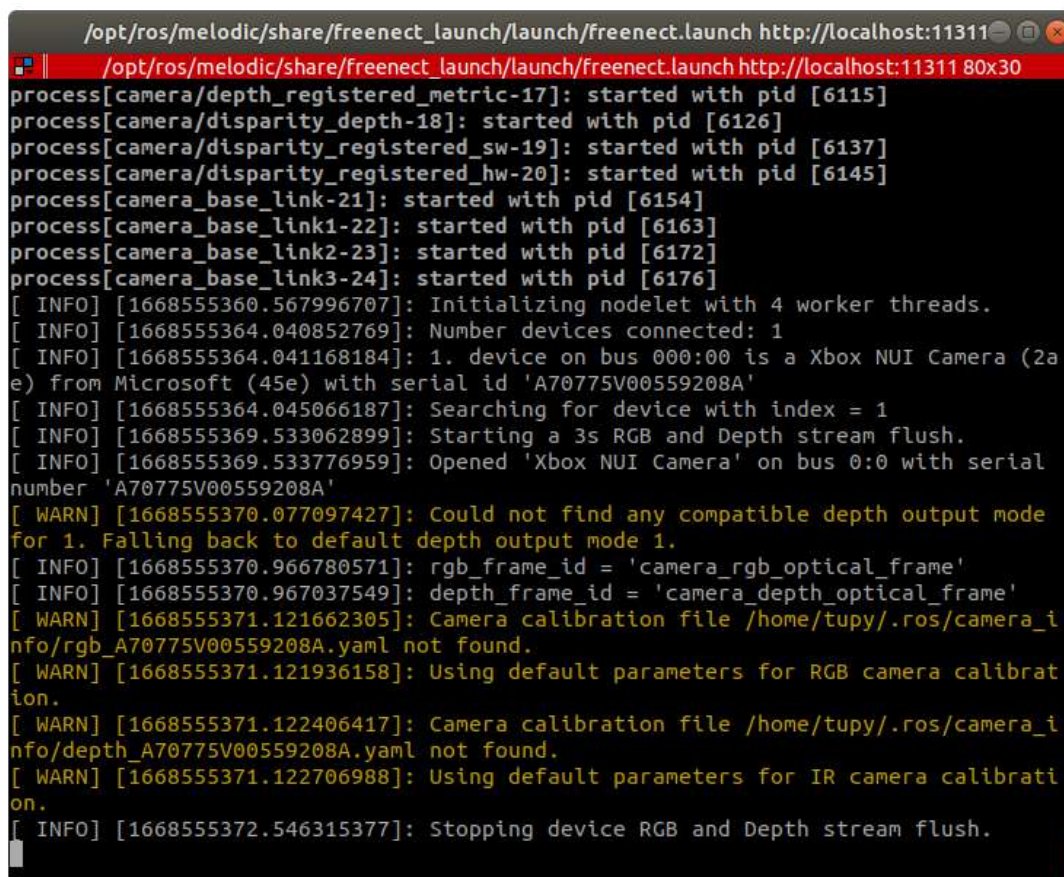
#### 4.1.1.4 Kinect e OpenCV

Após a compilação dos drivers **freenect**, inicia-se com o **roslaunch** a publicação dos Tópicos da câmera do Kinect, conforme os seguintes comandos:

- **ROS\_MASTER\_URI=http://192.168.15.21:11311**
- **ROS\_IP=192.168.15.22**
- **roslaunch freenect\_launch freenect.launch**



Figura 114: Iniciando Pacote Freenect.



```

/opt/ros/melodic/share/freenect_launch/launch/freenect.launch http://localhost:11311
/opt/ros/melodic/share/freenect_launch/launch/freenect.launch http://localhost:11311 80x30
process[camera/depth_registered_metric-17]: started with pid [6115]
process[camera/disparity_depth-18]: started with pid [6126]
process[camera/disparity_registered_sw-19]: started with pid [6137]
process[camera/disparity_registered_hw-20]: started with pid [6145]
process[camera_base_link-21]: started with pid [6154]
process[camera_base_link1-22]: started with pid [6163]
process[camera_base_link2-23]: started with pid [6172]
process[camera_base_link3-24]: started with pid [6176]
[ INFO] [1668555360.567996707]: Initializing nodelet with 4 worker threads.
[ INFO] [1668555364.040852769]: Number devices connected: 1
[ INFO] [1668555364.041168184]: 1. device on bus 000:00 is a Xbox NUI Camera (2a
e) from Microsoft (45e) with serial id 'A70775V00559208A'
[ INFO] [1668555364.045066187]: Searching for device with index = 1
[ INFO] [1668555369.533062899]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1668555369.533776959]: Opened 'Xbox NUI Camera' on bus 0:0 with serial
number 'A70775V00559208A'
[ WARN] [1668555370.077097427]: Could not find any compatible depth output mode
for 1. Falling back to default depth output mode 1.
[ INFO] [1668555370.966780571]: rgb_frame_id = 'camera_rgb_optical_frame'
[ INFO] [1668555370.967037549]: depth_frame_id = 'camera_depth_optical_frame'
[ WARN] [1668555371.121662305]: Camera calibration file /home/tupy/.ros/camera_i
nfo/rgb_A70775V00559208A.yaml not found.
[ WARN] [1668555371.121936158]: Using default parameters for RGB camera calibrat
ion.
[ WARN] [1668555371.122406417]: Camera calibration file /home/tupy/.ros/camera_i
nfo/depth_A70775V00559208A.yaml not found.
[ WARN] [1668555371.122706988]: Using default parameters for IR camera calibrati
on.
[ INFO] [1668555372.546315377]: Stopping device RGB and Depth stream flush.

```

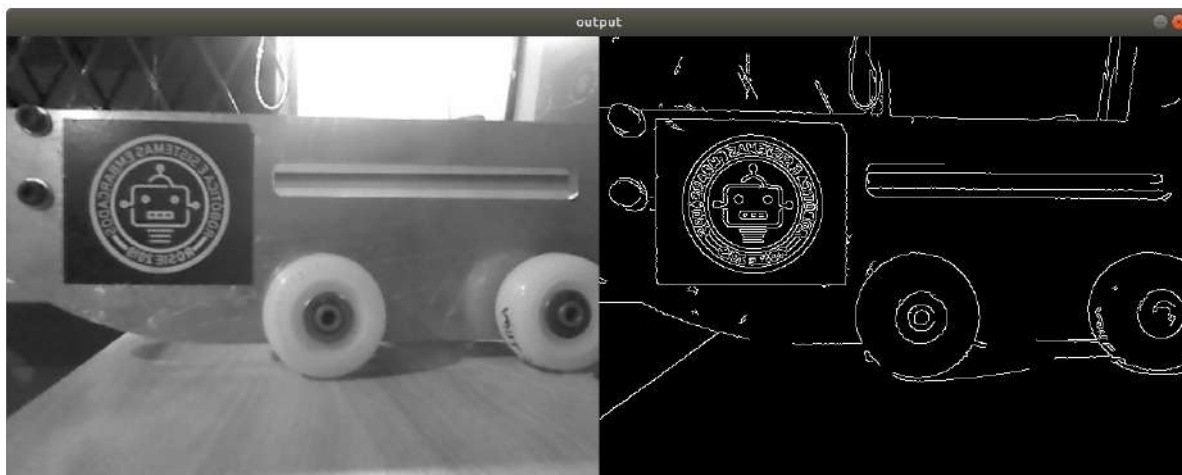
Fonte: O Autor (2022).

Nota-se, que os Tópicos estão prontos para serem inscritos. Então, inicia-se o pacote **Canny Kinect**, com os seguintes comandos no terminal:

- **export ROS\_MASTER\_URI=http://192.168.15.21:11311**
- **export ROS\_IP=192.168.15.22**
- **roslaunch img\_processor csf\_robotCS\_cannyKinect.py**

Com tudo certo, é retornado uma tela de visualização em tempo real com imagens em escala de cinza e com o rastreo das arestas, como se observa na Figura 115.

**Figura 115: Canny Kinect em execução.**



Fonte: O Autor (2022).

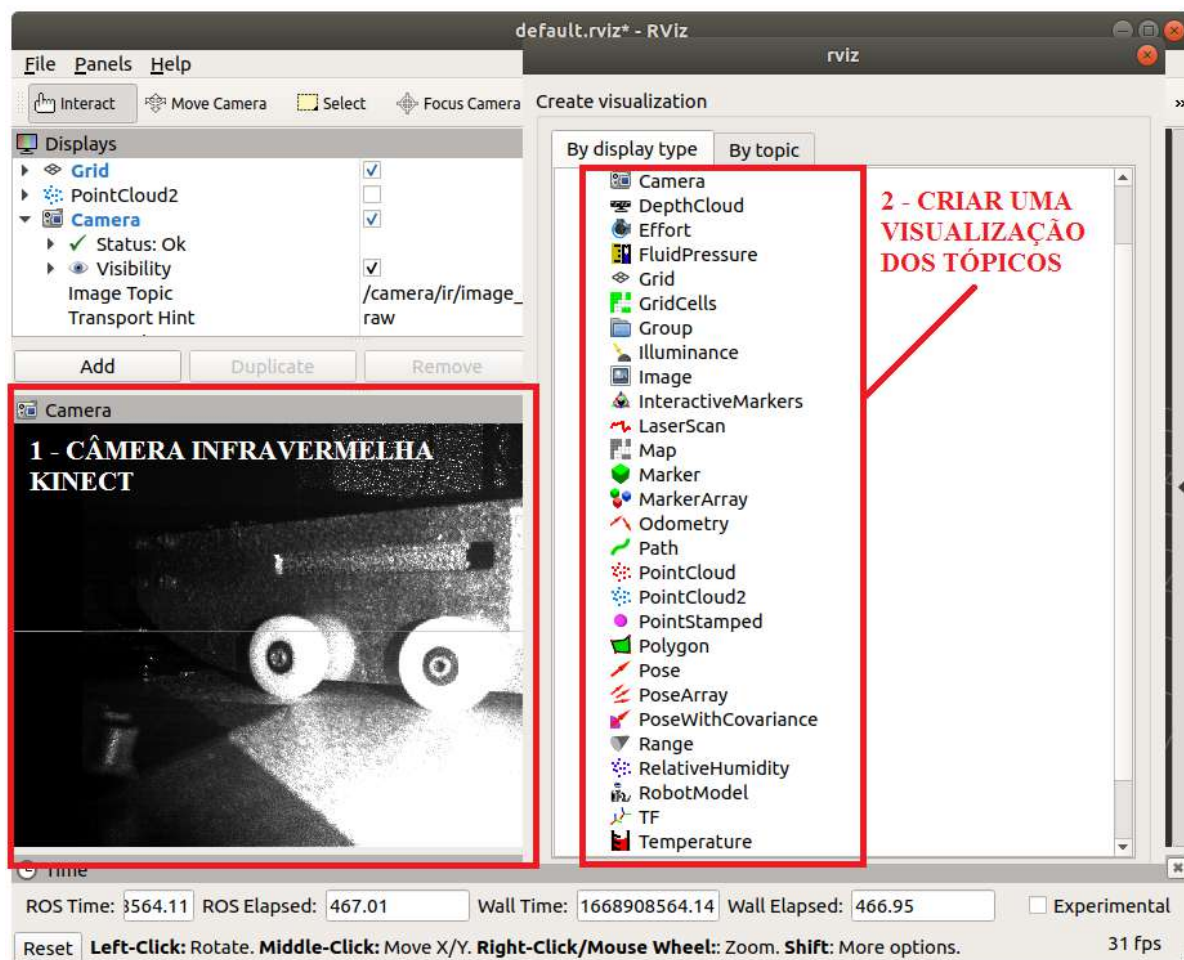
#### **4.1.2 Visualizando pacotes do ROS com Rviz e rtq\_graph**

A grande vantagem do uso do Rviz, é a facilidade da visualização de Tópicos ativos no ROS. Esta ferramenta é fundamental para a realização de testes durante o desenvolvimento no ambiente ROS, pois pode-se verificar que a comunicação entre os Nós do ROS está correta e se os drivers do Kinect foram iniciados corretamente. A Figura 116, demonstra através do detalhe (2) algumas destas opções disponíveis para criação. Já o detalhe (1), é possível visualizar a saída do tópico infravermelho do Kinect criado.

- **export ROS\_MASTER\_URI=http://192.168.15.21:11311**
- **export ROS\_IP=192.168.15.22**
- **rviz**



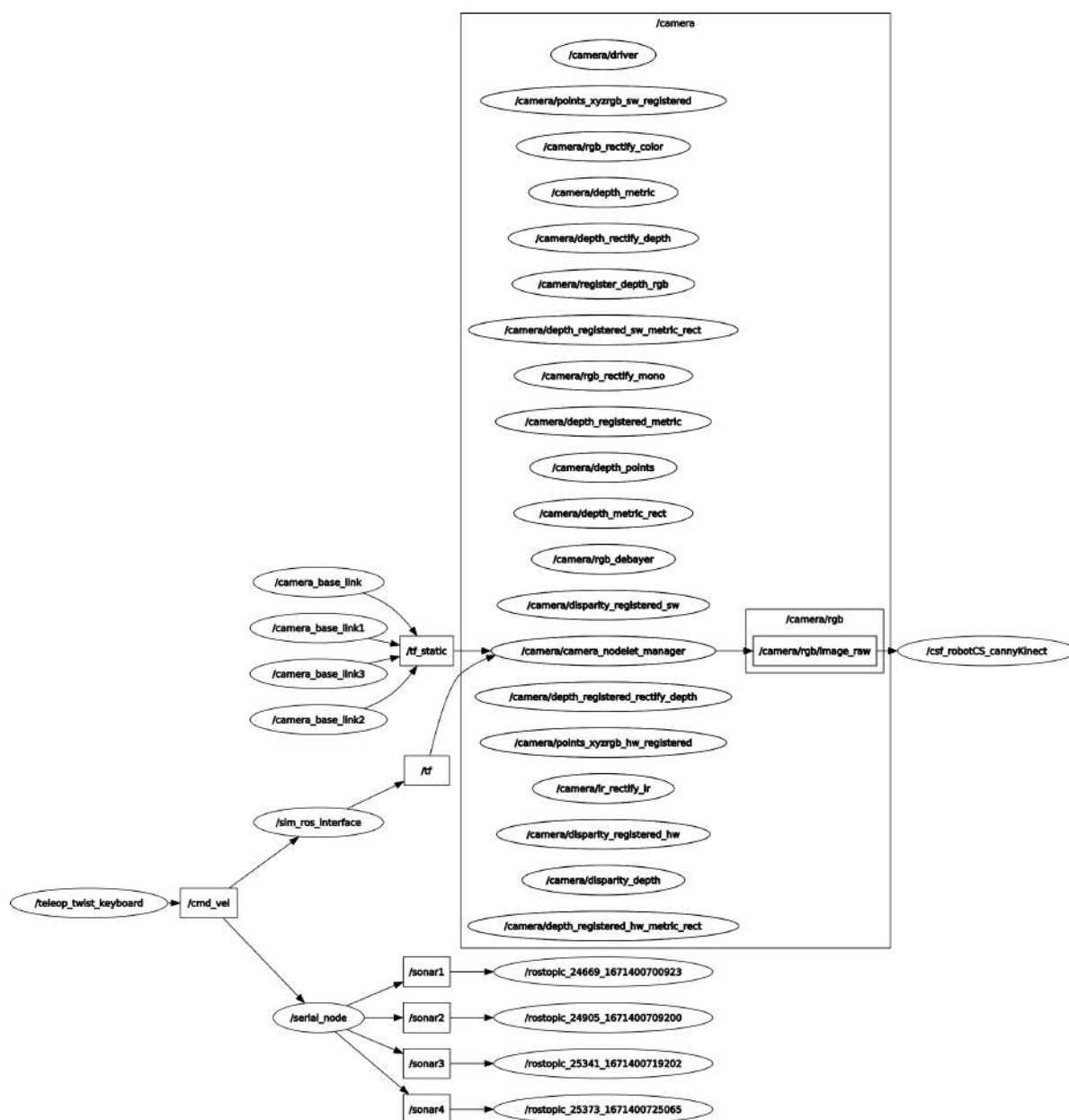
**Figura 116: Visualizando o Tópico infravermelho da câmera.**



Fonte: O Autor (2022).

Outra maneira de visualizar os tópicos ativos, é utilizar a ferramenta gráfica nativa do ROS `rtg_graph`, conforme Figura 117.

Figura 117: Arquitetura dos tópicos do robô Tupy com rtq\_graph.



Fonte: O Autor (2022).

É possível notar (Figura 117) que o *driver* Freenect inicia vários Tópicos referentes a câmera. Porém, apenas o Tópico **/camera/rgb** é necessário para o funcionamento do pacote Canny. Já o **teleop\_twist\_keyboard** realiza o envio de mensagens geométricas através do tópico **cmd\_vel** para as interfaces de simulação virtual (**sim\_ros\_interface**) e o robô Tupy físico através do Nó **serial\_node** que, também, faz a gestão dos tópicos referentes aos sensores ultrassônicos (sonares), publicando mensagens de distância (*Range*). Dessa forma tem-se o DT arquitetado pelo *framework* ROS.

### 4.1.3 Resumo da inicialização dos serviços do Gêmeo Digital

Portanto, em suma, o Tupy foi iniciado na sequência dos comandos inseridos no terminal do *desktop*, conforme ordem a seguir:

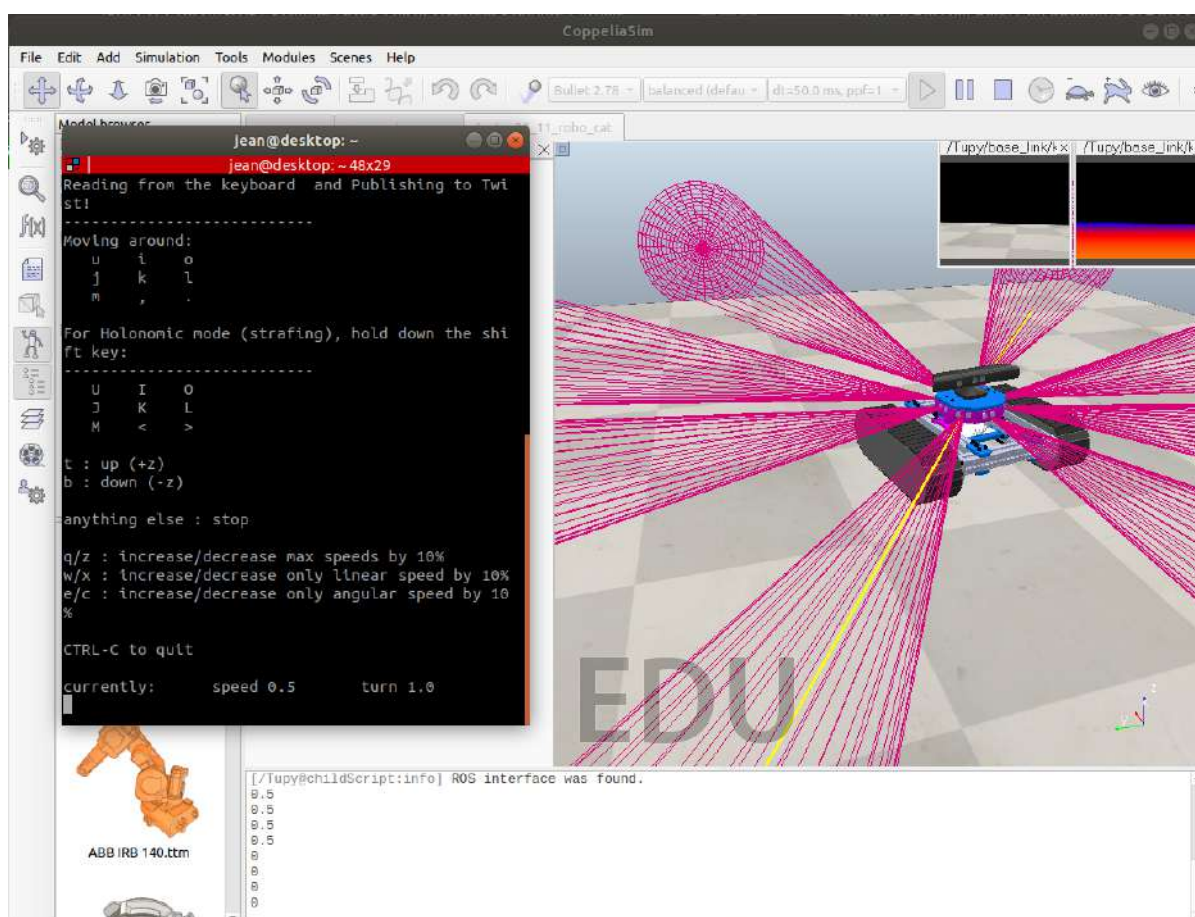
- No terminal 1, inicia-se o ROS *Master*:
  - jean@desktop: \$ roscore
- No terminal 2, inicia-se o *Freenect Driver*:
  - jean@desktop: \$ ssh tupy@192.168.15.22
  - tupy@tupy-robot: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - tupy@tupy-robot: \$ export ROS\_IP=192.168.15.22
  - tupy@tupy-robot: \$ roslaunch freenect\_launch freenect.launch
- No terminal 3, inicia-se a comunicação serial do Arduino:
  - jean@desktop: \$ ssh tupy@192.168.15.22
  - tupy@tupy-robot: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - tupy@tupy-robot: \$ export ROS\_IP=192.168.15.22
  - tupy@tupy-robot: \$ rosrun rosserial\_python serial\_node.py /dev/ttyACM0
- No terminal 4, inicia-se a leitura do primeiro sensor ultrassônico:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rostopic echo /sonar1
- No terminal 5, inicia-se a leitura do segundo sensor ultrassônico:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rostopic echo /sonar2
- No terminal 6, inicia-se a leitura do terceiro sensor ultrassônico:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rostopic echo /sonar3
- No terminal 7, inicia-se a leitura do quarto sensor ultrassônico:

- jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rostopic echo /sonar4
- No terminal 8, inicia-se a execução do pacote Canny:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rosrun img\_processor csf\_robotCS\_cannyKinect.py
- No terminal 9, inicia-se a teleoperação com pacote Twist:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py
- No terminal 10, inicia-se o ambiente virtual do CoppeliaSim:
  - jean@desktop: \$ export ROS\_MASTER\_URI=http://192.168.15.21:11311
  - jean@desktop: \$ export ROS\_IP=192.168.15.21
  - jean@desktop: \$ coppelia

#### **4.1.4 Tupy no ambiente ciber-físico como gêmeo digital**

Finalizando todas as etapas demonstradas, parte-se para a inicialização do cenário virtual no CoppeliaSim, então abre-se o cenário com a modelagem do robô Tupy e inicia-se a simulação. Com isso, tem-se o DTMR Tupy no ambiente ciber, conforme Figura 118.

**Figura 118: Tupy no ambiente CoppeliaSim.**



Fonte: O Autor (2022).

Então, a partir do momento em que comandos são inseridos no terminal da teleoperação, o robô virtual e físico passam a se movimentar, com o pacote da câmera, gera imagens do ambiente e, por fim, através da comunicação serial com o Arduino, medir as distâncias dos obstáculos com as mensagens de distâncias enviadas pela leitura dos sensores ultrassônicos, conforme a Figura 119, do robô Tupy no ambiente real. Portanto, através deste desenvolvimento, tem-se o Gêmeo Digital, aplicados à robótica móvel DTMR.

**Figura 119: Tupy no ambiente Real.**



Fonte: O Autor (2022).