

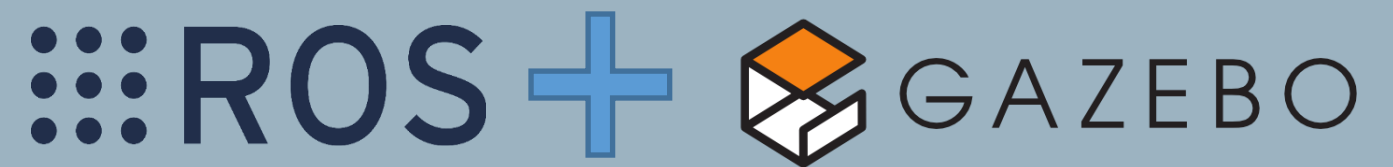
POLITECNICO DI MILANO

AUTONOMOUS VEHICLES

S. Arrigoni



POLITECNICO
MILANO 1863



Important prerequisites

Let's initialize Gazebo

Install examples for Gazebo before test it!

Install in your ws folder! (ws_new?)

1. 1. Install Simulation Package

The **TurtleBot3 Simulation Package** requires `turtlebot3` and `turtlebot3_msgs` packages as prerequisite. Without these prerequisite packages, the Simulation cannot be launched.

Please follow the [PC Setup](#) instructions if you did not install required packages and dependent packages.

```
$ cd ~/catkin_ws/src/  
$ git clone -b kinetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git  
$ cd ~/catkin_ws && catkin_make
```

<https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/>

Software requirements

Not mandatory: (just if you want to collect data directly from Matlab)



- Matlab 2022a(**recommended**) / 2022b

Install from add-on explorer: “ROS Toolbox Support Package for TurtleBot-Based Robots”

The screenshot shows the MATLAB Add-on Explorer interface. At the top, there's a dark blue header with navigation icons and a search bar. Below the header, the main content area displays the details for the "ROS Toolbox Support Package for TurtleBot-Based Robots". On the left, there's a small image of a TurtleBot robot. To the right of the image, the package name is prominently displayed, followed by the author "MathWorks Robotics and Autonomous Systems Team" and a "STAFF" badge. Below this, a brief description states "Drive your TurtleBot and acquire sensor data." and a "Hardware Support" icon is shown. To the right of the package details, there's a star rating (4.5 stars), download count (3.2K Downloads), and update date (Updated 14 Sep 2022). Below these, there are two buttons: "Learn More" and "Install". The "Install" button is highlighted, and a dropdown menu is open, showing options: "Install" and "Download Only...". Below the package details, there's a section for "Requires" which shows a green checkmark next to "ROS Toolbox". At the bottom, there's a section for "MATLAB Release Compatibility" which states "Created with R2016a".

ROS Toolbox Support Package for TurtleBot-Based Robots

by MathWorks Robotics and Autonomous Systems Team **STAFF**

Drive your TurtleBot and acquire sensor data.

Hardware Support

★★★★★ (3)
3.2K Downloads
Updated 14 Sep 2022

Learn More Install

Install
Download Only...

Requires

✓ ROS Toolbox

MATLAB Release Compatibility

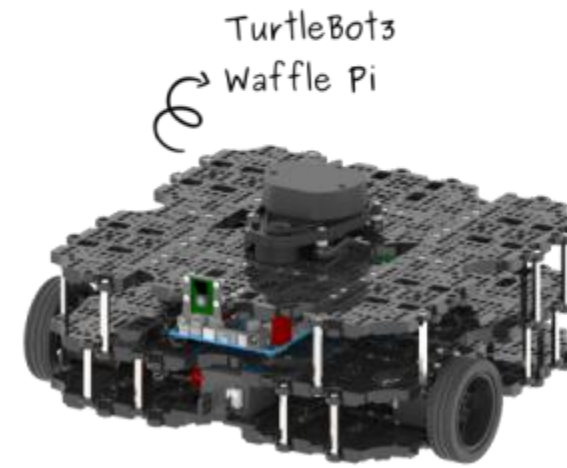
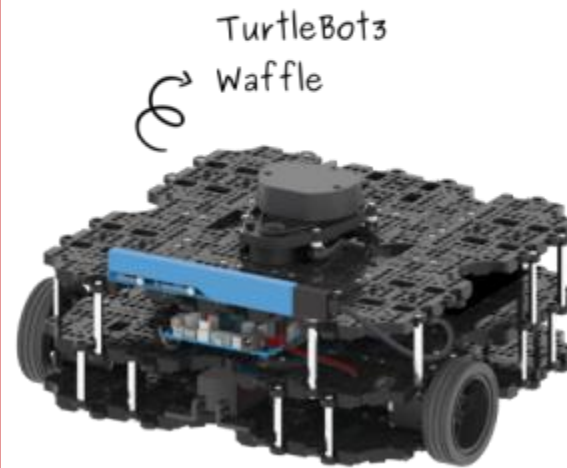
Created with R2016a



TURTLEBOT3

Intro

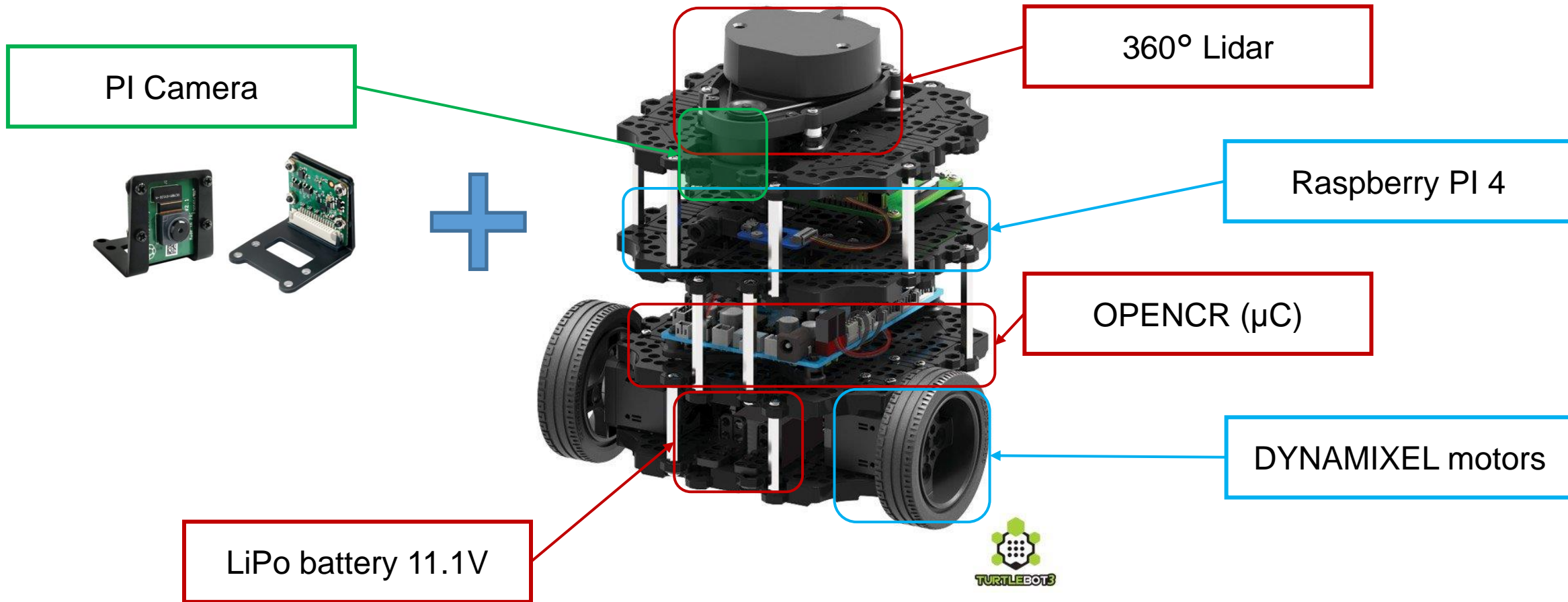
Turtlebot3



Source:

<https://emanual.robotis.com/>

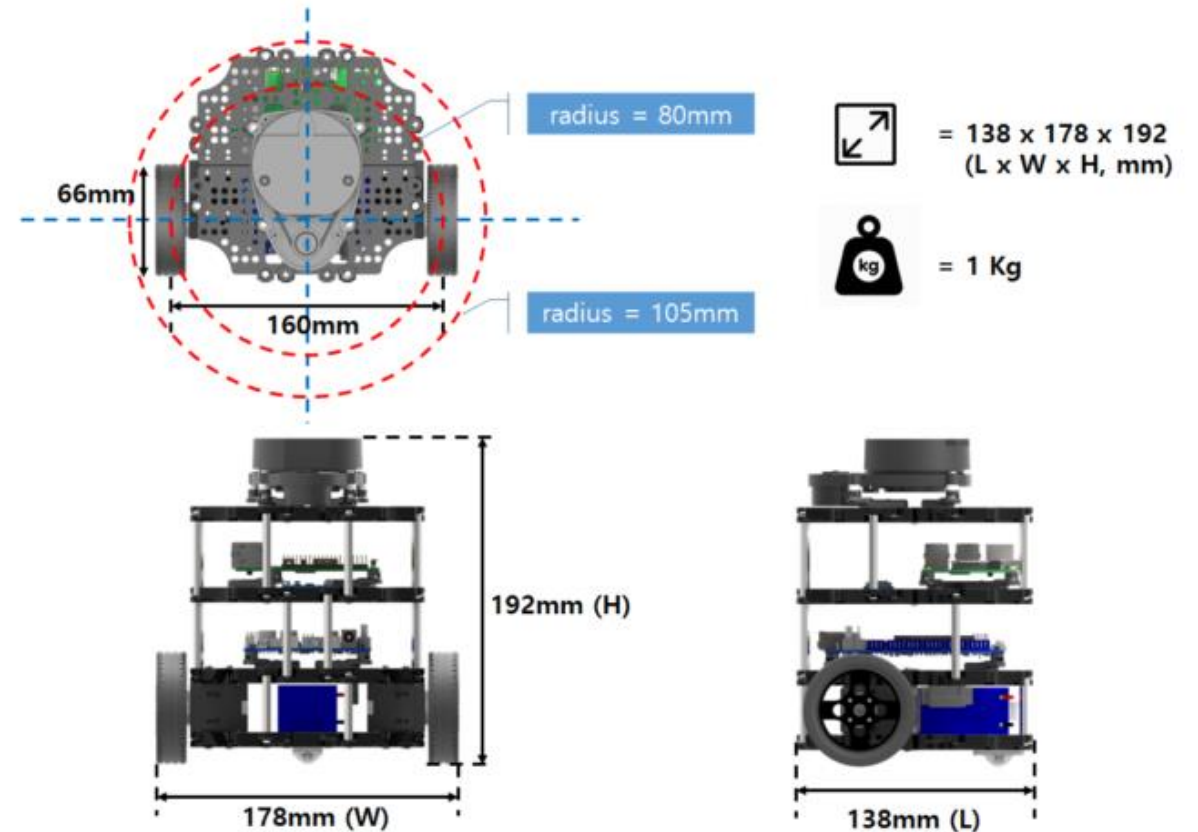
Turtlebot3



Turtlebot3

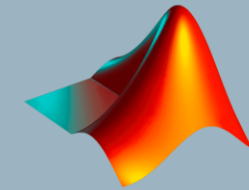
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L x W x H)	138mm x 178mm x 192mm
Weight	1kg
Threshold of climbing	10 mm or lower
Expected oper / charg time	2h 30m
SBC (Single Board Computers)	Raspberry Pi 4
Actuator	XL430-W250
LDS(Laser Distance Sensor)	360 Lidar LDS-02
Camera	PI Camera
IMU	Gyroscope 3 Axis Accelerometer 3 Axis
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C

TurtleBot3 Burger



Introduction to

ROS



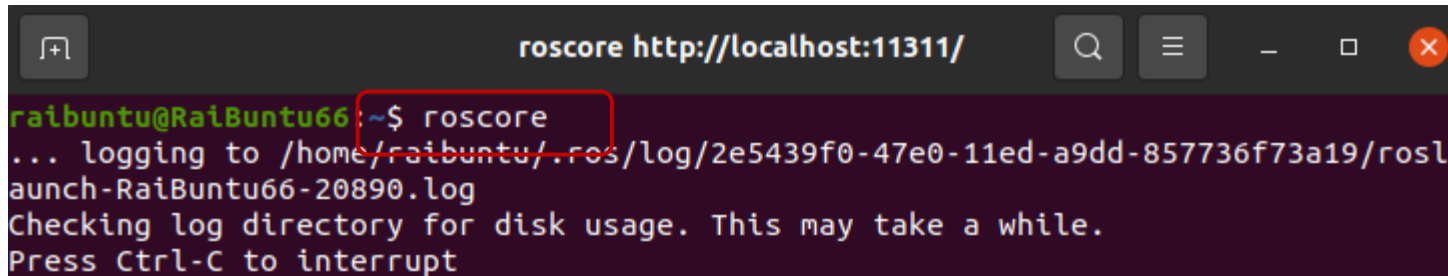
MATLAB®



GAZEBO

TURTLEBOT3

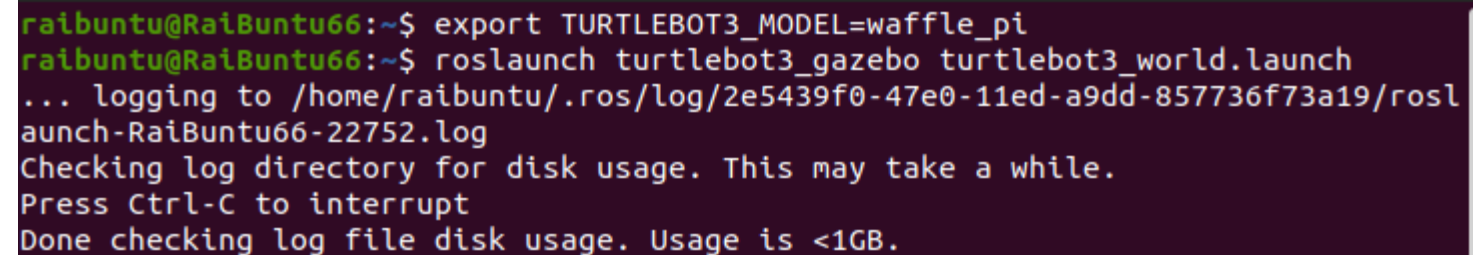
Let's initialize ROS + Gazebo environment

A terminal window titled 'roscore http://localhost:11311/' with standard window controls. The prompt is 'raibuntu@RaIBuntu66:~\$'. The command 'roscore' is entered and highlighted with a red box. The output shows logging to a directory and a disk usage check.

```
raibuntu@RaIBuntu66:~$ roscore
... logging to /home/raibuntu/.ros/log/2e5439f0-47e0-11ed-a9dd-857736f73a19/rosl
aunch-RaIBuntu66-20890.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
```

Initialize: Ros

Initialize: Gazebo +
Turtlebot3

A terminal window showing the execution of two commands: setting the Turtlebot3 model and launching the gazebo world. The output shows logging and a disk usage check.

```
raibuntu@RaIBuntu66:~$ export TURTLEBOT3_MODEL=waffle_pi
raibuntu@RaIBuntu66:~$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
... logging to /home/raibuntu/.ros/log/2e5439f0-47e0-11ed-a9dd-857736f73a19/rosl
aunch-RaIBuntu66-22752.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

```
export TURTLEBOT3_MODEL=waffle_pi
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Let's use:



Connect to virtual robot

ipaddress = 'localhost'; % # IP of real turtlebot / or other PC

tbot = turtlebot(ipaddress)

```
>> tbot= turtlebot('localhost')
```

The value of the ROS_HOSTNAME environment variable, localhost, will be used to set the advertised address for the ROS node.

```
tbot = |
```

turtlebot with properties:

```
    Velocity: [1x1 struct]
    ColorImage: [1x1 struct]
    GrayImage: [1x1 struct]
    DepthImage: [1x1 struct]
    PointCloud: [1x1 struct]
    LaserScan: [1x1 struct]
    Odometry: [1x1 struct]
    OdometryReset: [1x1 struct]
    IMU: [1x1 struct]
    TransformFrames: {2x1 cell}
    TopicNames: {30x1 cell}
```

Let's use:



#1: read Odometry

position and orientation as [x y z] coordinates and [yaw pitch roll] angles

```
>> odom = getOdometry(tbot)
```

```
odom =
```

struct with fields:

```
Position: [-1.9999 -0.4996 -0.0010]  
Orientation: [0.0023 0.0032 -3.3219e-06]
```


Let's use:



#2: read camera : Let's make an example!

```
%% assign topic to control it!  
tbot.Velocity.TopicName = '/cmd_vel';  
%% camera  
desiredRate = 2;  
rate = rateControl(desiredRate);
```

Define *topic_name* to control turtlebot

Define control / visualization frequency

```
reset(rate)  
for i = 1:20  
    setVelocity(tbot,0.01,0.5)  
    img = getColorImage(tbot,0);  
    imshow(img)  
    waitfor(rate);  
end
```

Provide control inputs to turtlebot

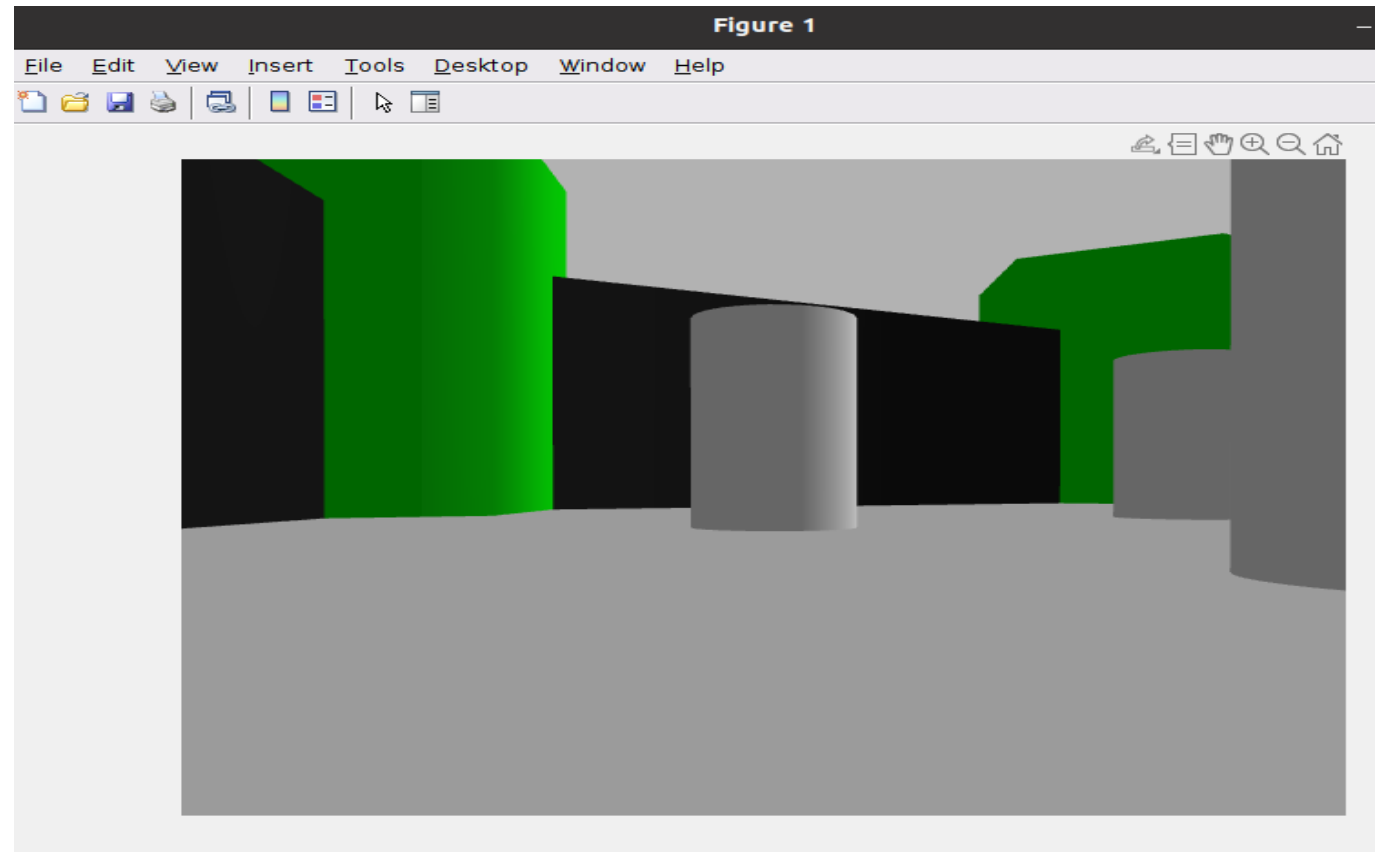
Acquire / visualize camera

Let's use:



#2: read camera : Let's make an example!

expected result



Let's use:



#3: read laser scanner: Let's make an example!

```
[scan,scanMsg] = getLaserScan(tbot);  
% scan = range [meters] and angle [radiants]  
% scanMsg = object of ROS message  
plot(scanMsg)
```

acquire scan data

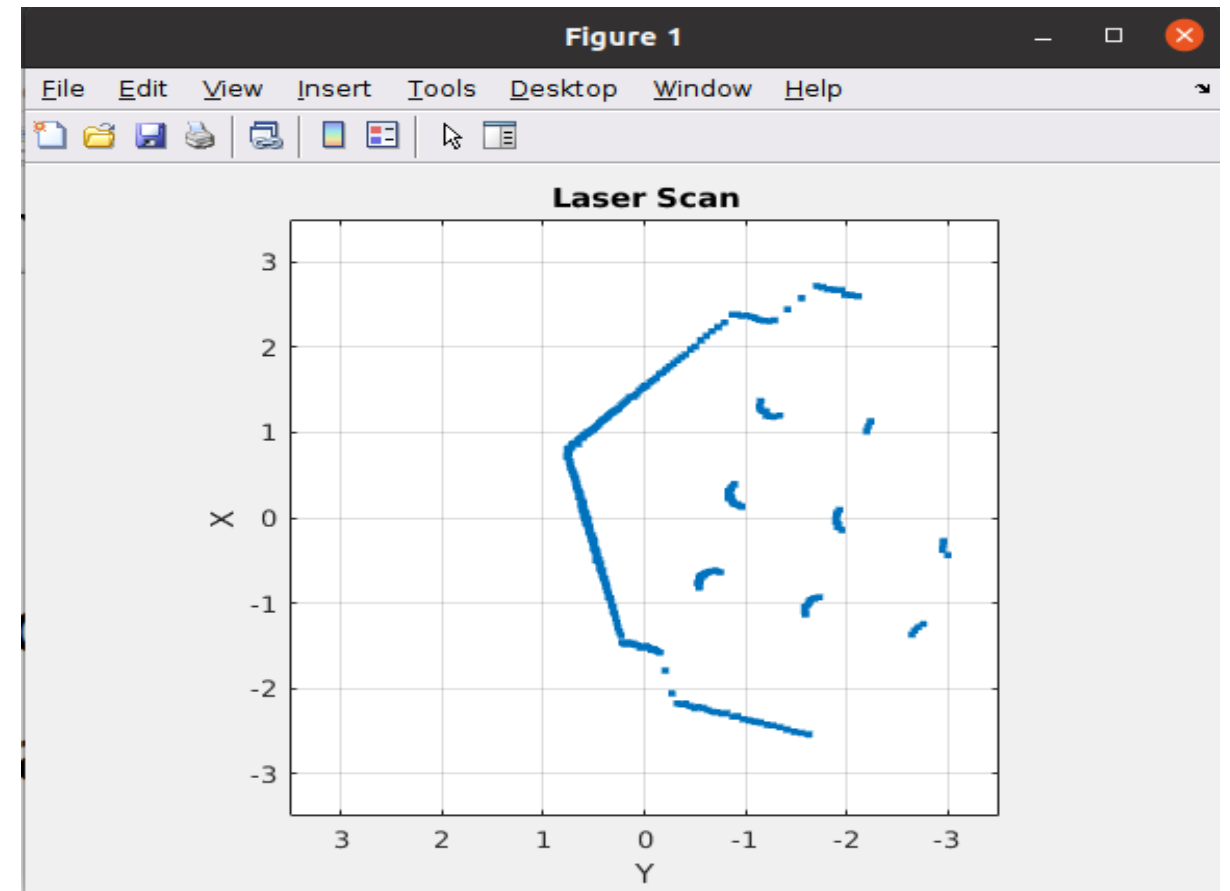
visualize results (x,y)

Let's use:



#3: read laser scanner: Let's make an example!

expected result



Let's use:



Initialize ROS in MATLAB

rosinit

Create a new simulink file and initialize it to ROS (*last lesson*)

Let's use:



#1 read generic sensors:

- ✓ You can always read all data coming from ROS topics (*subscriber*)

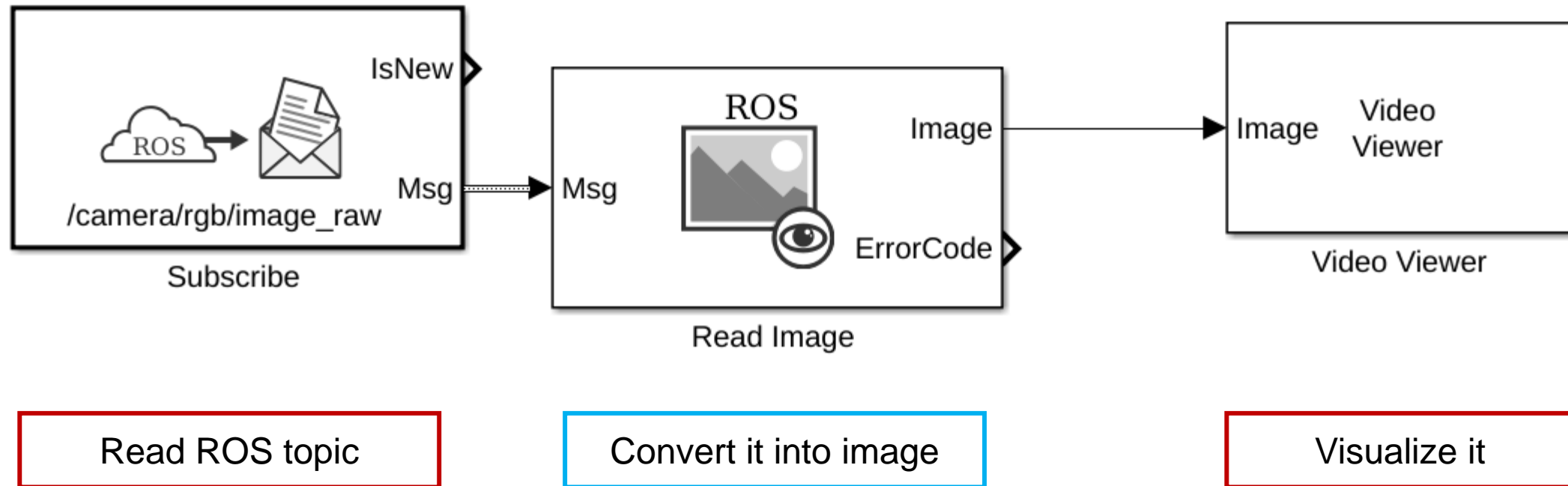
Let's analyze how to deal with *specific* sensor data from Turtlebot3:

- ✓ Camera
- ✓ Laser scanner

Let's use:



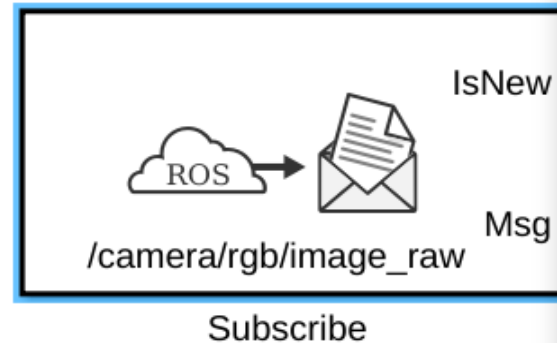
#2: camera



Let's use:



#2: camera



Block Parameters: Subscribe

ROS Subscribe (mask) (link)

Receive messages from ROS network.

The Msg block output is a ROS message (bus signal). Use a Bus Selector block to extract signals you want to work with. The IsNew block output is a boolean indicating whether a message was received during the previous time step. When IsNew is true, Msg holds the newly-received message. When IsNew is false, Msg holds the last received message.

To select from a list of topics available in an active ROS network, set the Topic source parameter to "Select from ROS network" and use the "Select..." button. You must be connected to a ROS network to get a list of active topics. The message type for the selected topic is set automatically.

To enter a custom topic without an active ROS connection, set Topic source to "Specify your own". Use the Topic parameter to specify the name, and the "Select..." button to select the message type.

[Configure network addresses](#)

Main Code Generation

Topic source: **Select from ROS network**

Topic: /camera/rgb/image_raw **Select ...**

Message type: sensor_msgs/Image

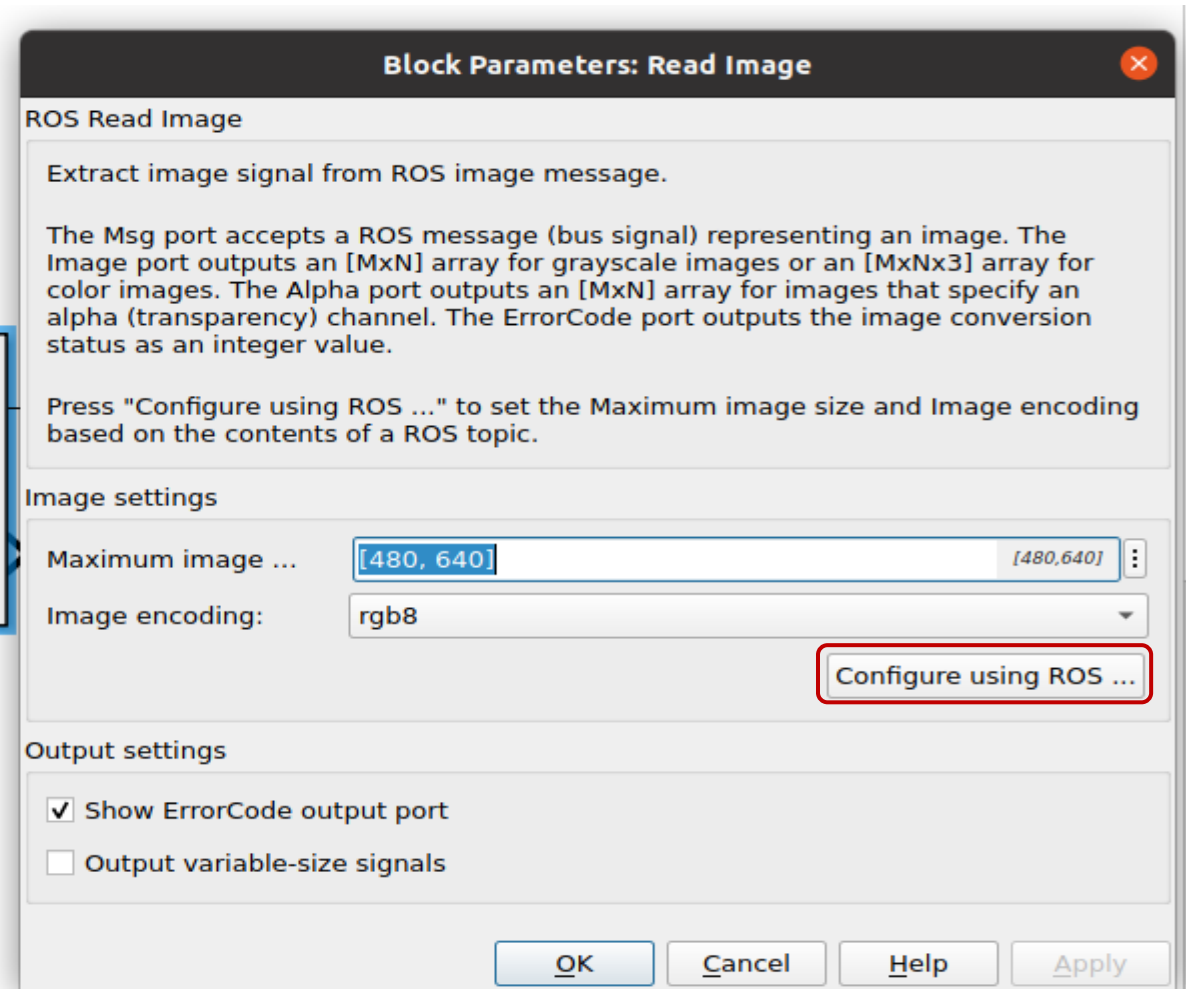
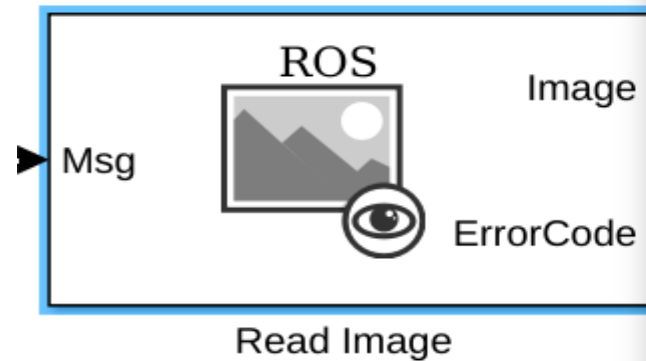
Sample time: -1

OK Cancel Help Apply

Let's use:



#2: camera

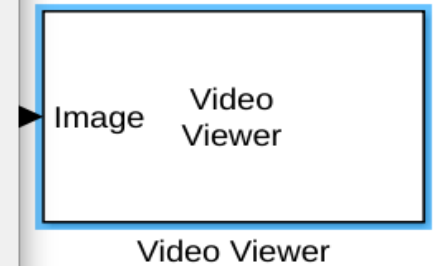
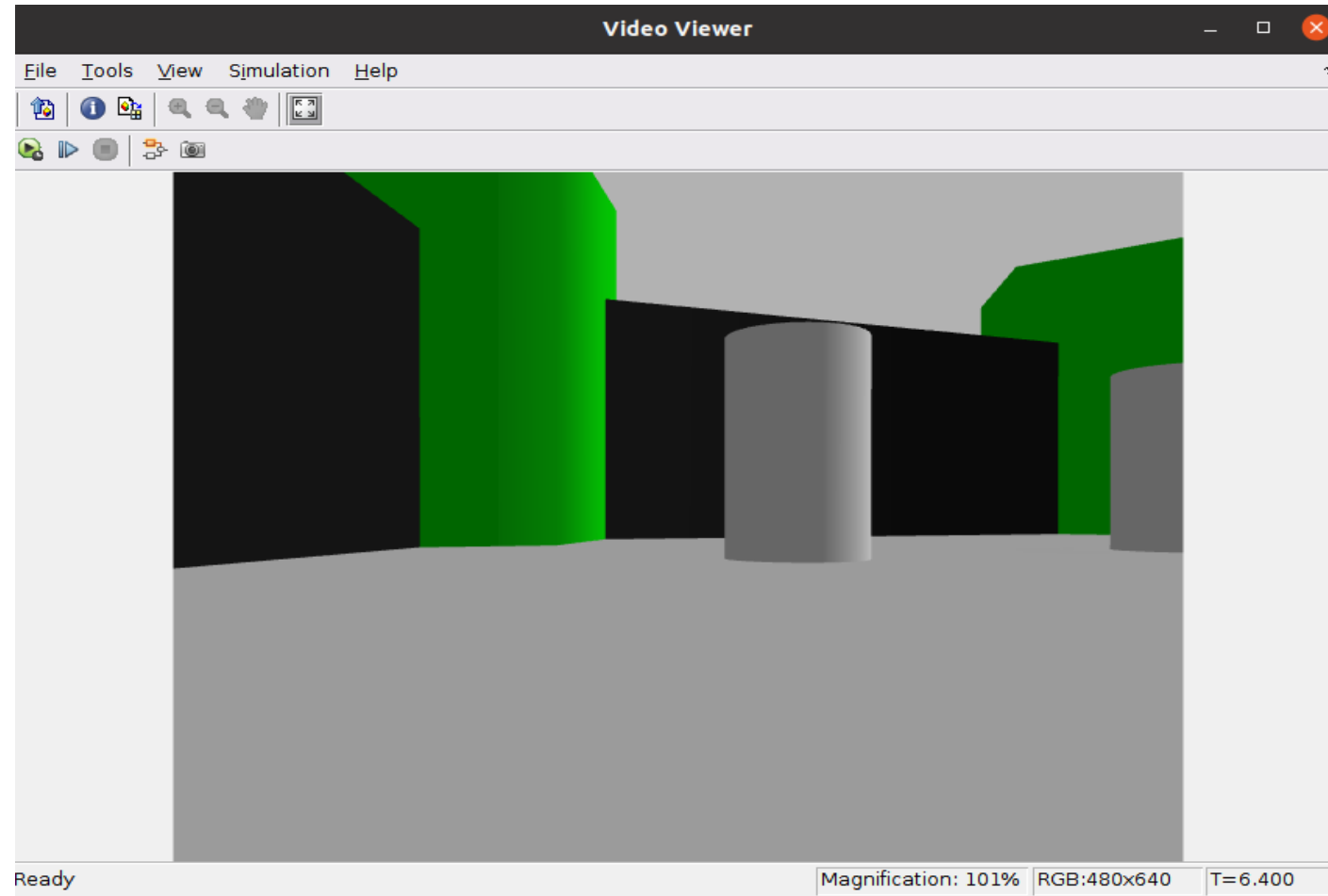


Let's use:



#2: camera

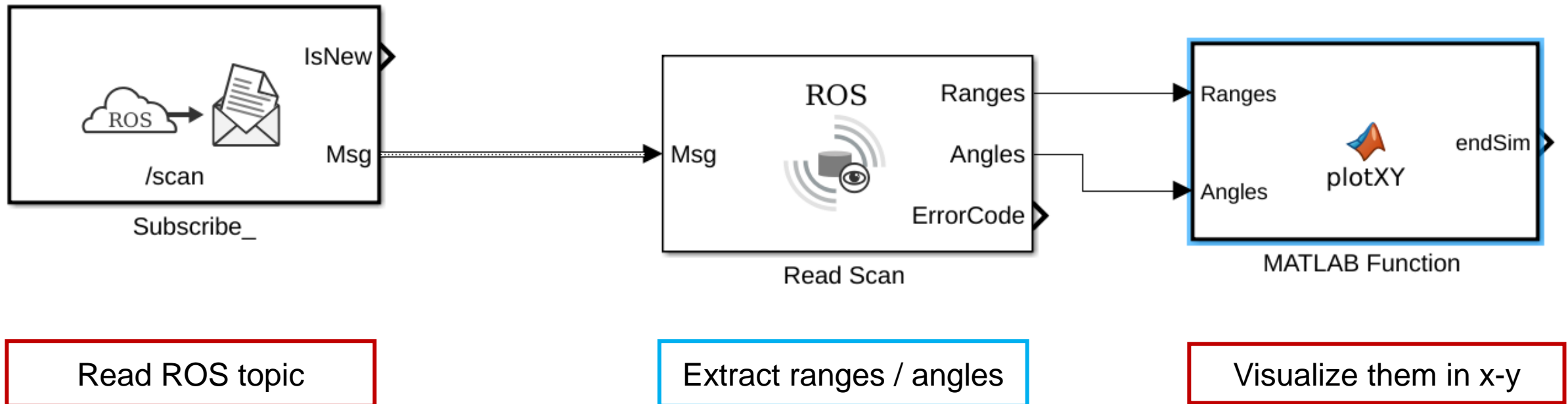
expected result



Let's use:



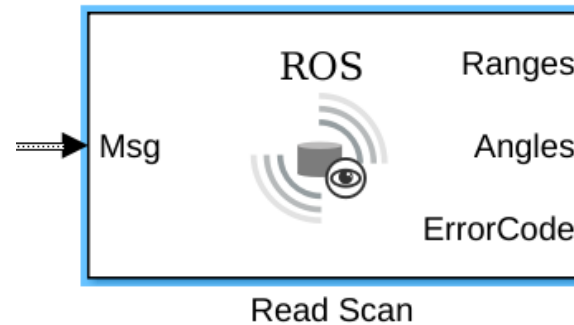
#3: laser scanner



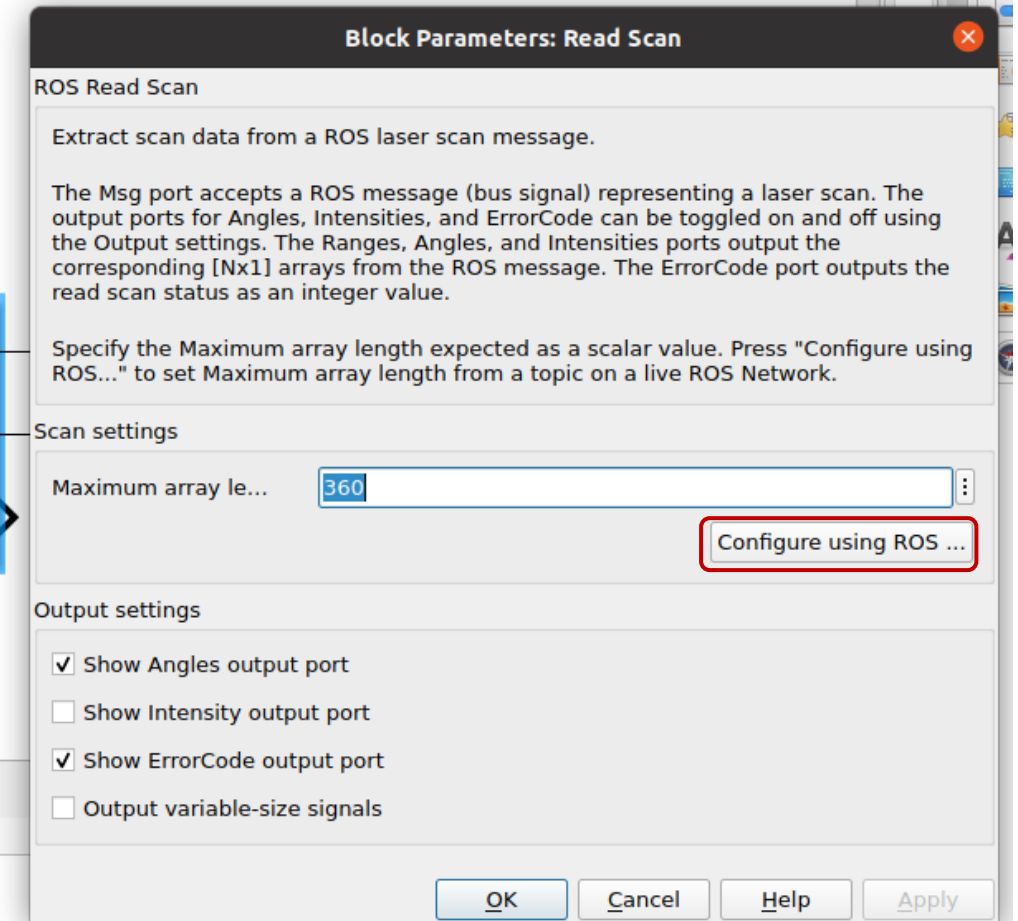
Let's use:



#3: laser scanner



1 address for the ROS node.



Let's use:



#3: laser scanner

From data: if $\neq 0$

Project in x-y plane

Plot it in a figure

```
example_2 ► MATLAB Function

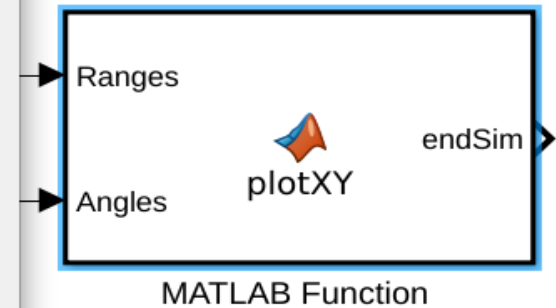
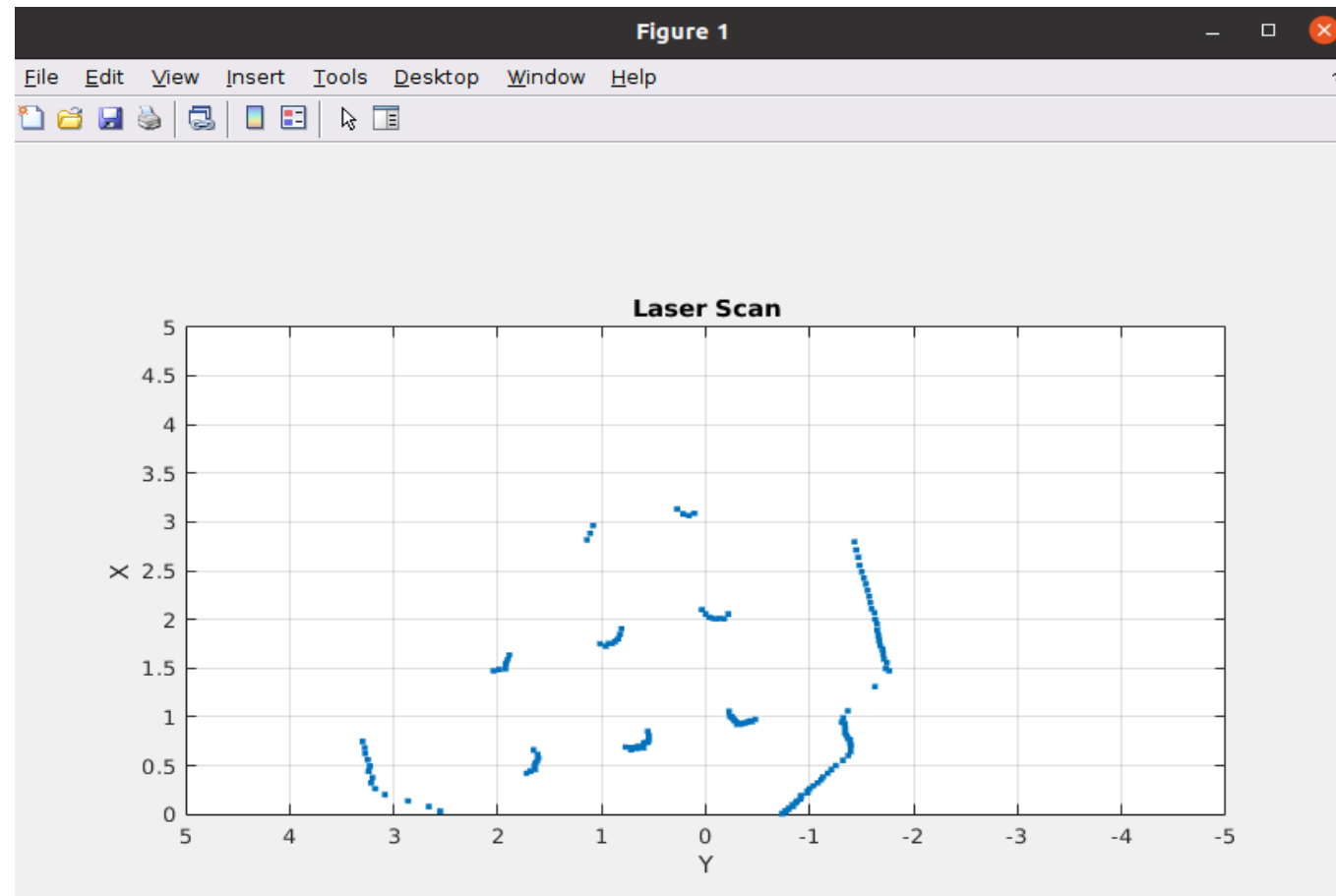
1  function endSim = plotXY(Ranges, Angles)
2  % Read Scan block will output all zeros until the Subscriber blocks
3  % outputs a message
4  endSim = any(Ranges>0); % =1 if at least one range is not zero
5  if endSim
6      x = Ranges.*cos(Angles);
7      y = Ranges.*sin(Angles);
8      % only return finite values
9      xValid = x(isfinite(x));
10     yValid = y(isfinite(y));
11     plot(xValid,yValid,marker=".",LineStyle="none", MarkerSize=8)
12     title('Laser Scan')
13     xlabel("X")
14     ylabel("Y")
15     set(gca, 'YDir','reverse')
16     grid on
17     axis equal
18     ylim([-5 5])
19     xlim([0 5])
20     view([90 -90])
21 end
22
```

Let's use:



#3: laser scanner

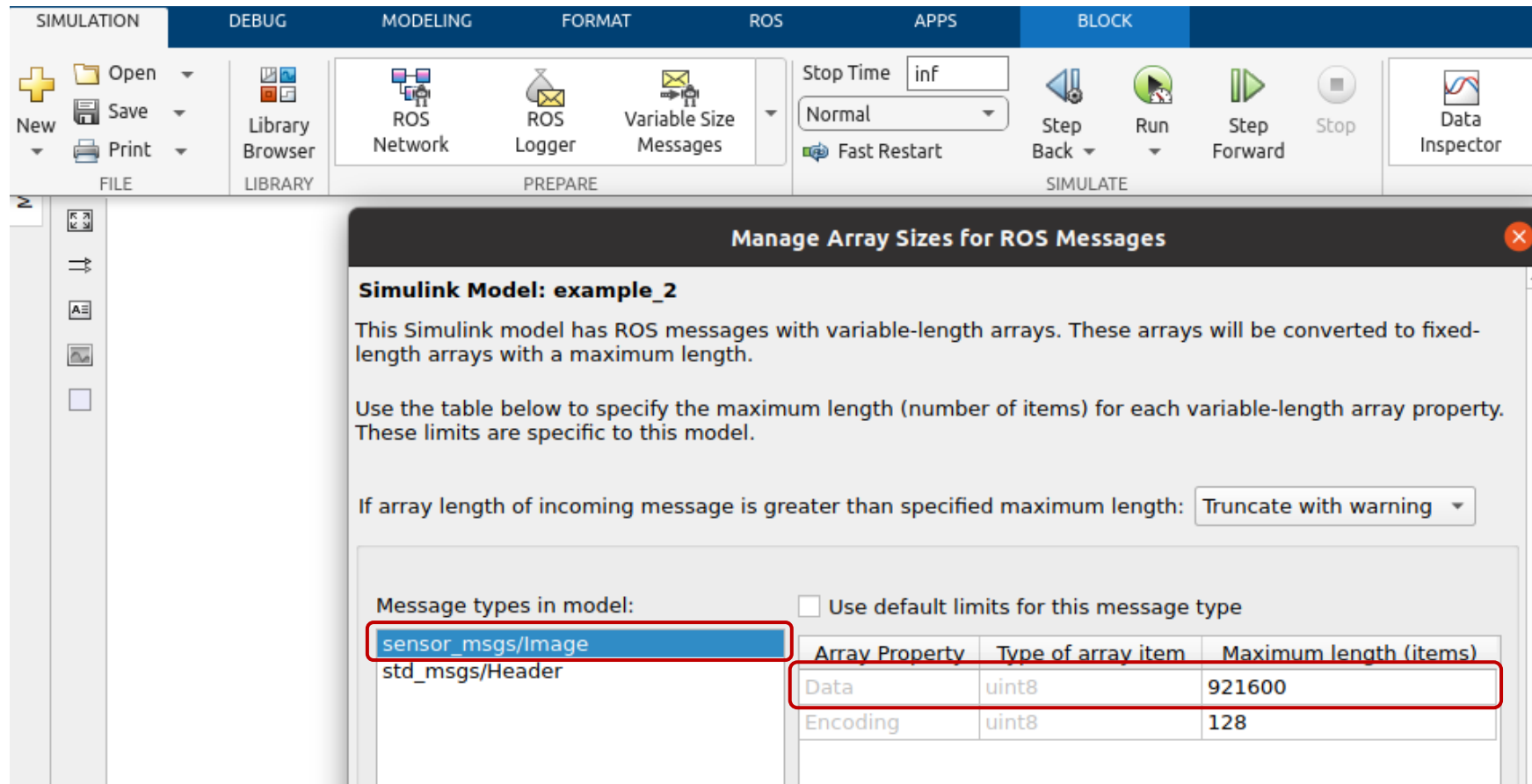
expected result



Let's use:



#fix for both data processing#



Manage Array Sizes for ROS Messages

Simulink Model: example_2

This Simulink model has ROS messages with variable-length arrays. These arrays will be converted to fixed-length arrays with a maximum length.

Use the table below to specify the maximum length (number of items) for each variable-length array property. These limits are specific to this model.

If array length of incoming message is greater than specified maximum length: Truncate with warning

Message types in model:

- sensor_msgs/Image
- std_msgs/Header

☐ Use default limits for this message type

Array Property	Type of array item	Maximum length (items)
Data	uint8	921600
Encoding	uint8	128

Use of the Bag

Let's save all messages!

Save all messages

rosvag record -a

Let's replay them!

- **terminal**

rosvag play nome.bag

- **graphics**

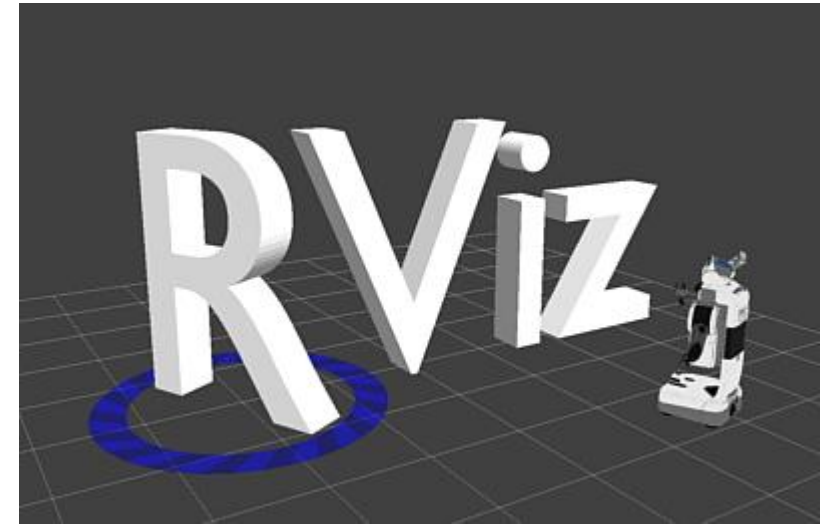
rqt_bag

Let's use Rviz

Rviz = a 3D visualizer for ROS

A powerful 3D visualization tool for ROS that allows to:

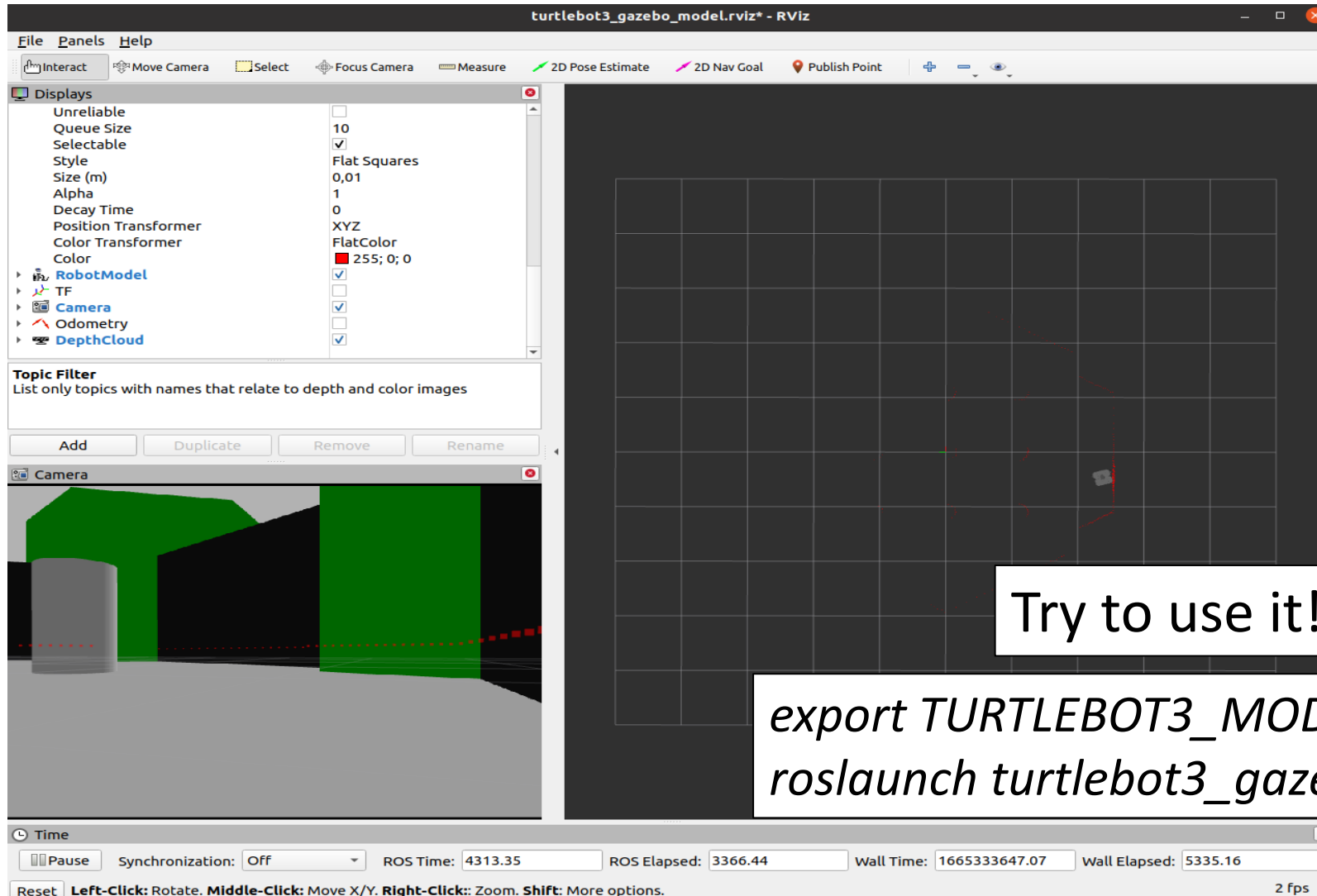
- view the **robot model**
- **display** and/or log sensor information
- **replay** the logged sensor information



It can displays:

- 3D sensor data from stereo cameras, lasers, Kinects, and other 3D devices in the form of **point clouds** or **depth images**;
- 2D sensor data from webcams, RGB cameras, and 2D laser rangefinders in the form of **image data**.

Let's use Rviz



```
export TURTLEBOT3_MODEL=waffle_pi  
roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

That's it for today...

See you next time!

S. Arrigoni



POLITECNICO
MILANO 1863