Get Started with ROS in Matlab

Robot Operating System (ROS) is a communication interface that enables different parts of a robot system to discover each other, and send and receive data between them. MATLAB® supports ROS with a library of functions that enables you to exchange data with ROS-enabled physical robots or robot simulators such as Gazebo®.

This example introduces how to:

- Set up ROS within MATLAB
- Get information about capabilities in a ROS network
- Get information about ROS messages

ROS Terminology

- A *ROS network* comprises different parts of a robot system (such as a planner or a camera interface) that communicate over ROS. The network can be distributed over several machines.
- A *ROS master* coordinates the different parts of a ROS network. It is identified by a *Master URI* (Uniform Resource Identifier) that specifies the hostname or IP address of the machine where the master is running.
- A *ROS node* contains a collection of related ROS capabilities (such as publishers, subscribers, and services). A ROS network can have many ROS nodes.
- *Publishers*, *subscribers*, and *services* are different kinds of ROS entities that process data. They exchange data using *messages*.
- A publisher sends messages to a specific *topic* (such as "odometry"), and subscribers to that topic receive those messages. A single topic can be associated with multiple publishers and subscribers.

For more information, see Robot Operating System (ROS) and the Concepts section on the ROS website.

Initialize ROS Network

Use `rosinit` to initialize ROS. By default, `rosinit` creates a *ROS master* in MATLAB and starts a *global node* that is connected to the master. The global node is automatically used by other ROS functions.

```
rosinit
```

```
Launching ROS Core...
.Done in 1.8742 seconds.
Initializing ROS master on http://172.27.160.1:11311.
Initializing global node /matlab_global_node_50463 with NodeURI http://desktop-sqc5nrv:57432/ and MasterURI http://1
```

Use `rosnode list` to see all nodes in the ROS network. Note that the only available node is the global node created by `rosinit`.

```
rosnode list
```

```
/matlab_global_node_50463
```

Use exampleHelperROSCreateSampleNetwork to populate the ROS network with three additional nodes and sample publishers and subscribers.

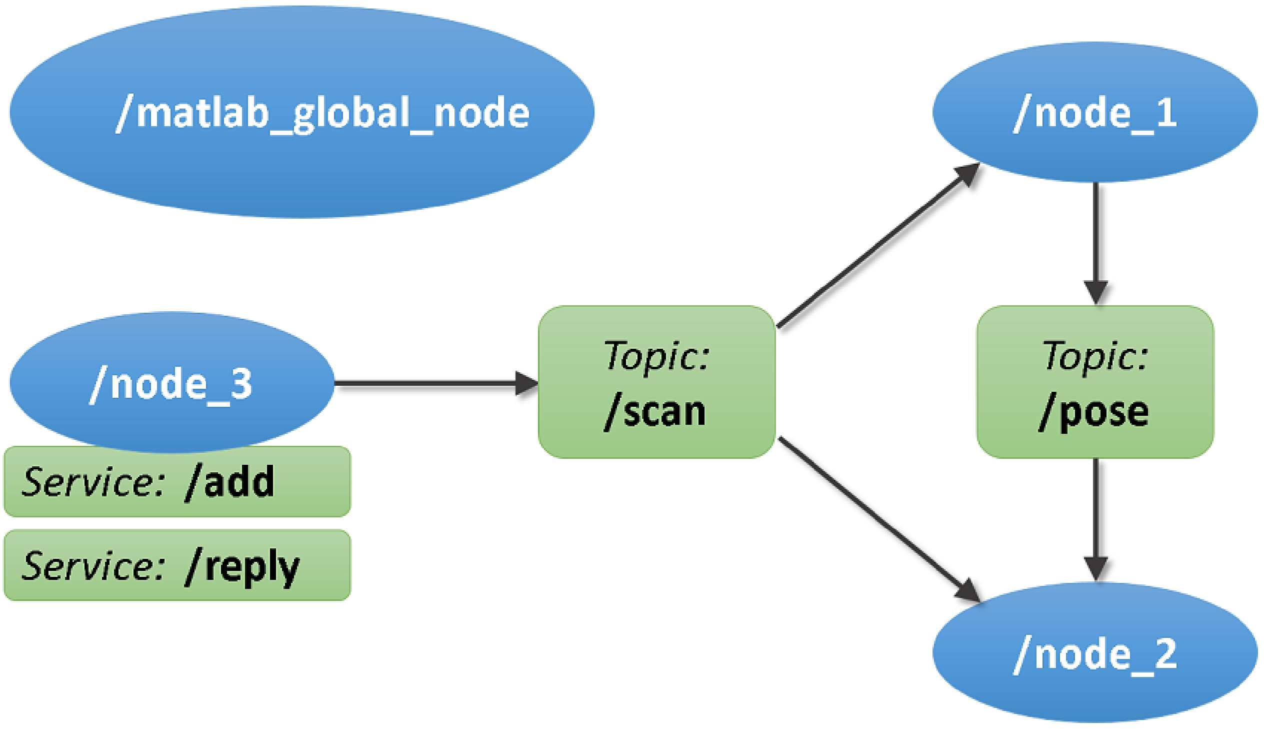
```
exampleHelperROSCreateSampleNetwork
```

Use rosnode list again to see the three new nodes (node_1, node_2, and node_3).

```
rosnode list
```

```
/matlab_global_node_50463
/node_1
/node_2
/node_3
```

The figure shows the current state of the ROS network. The MATLAB global node is disconnected since it currently does not have any publishers, subscribers or services.



Topics

Use rostopic list to see available topics in the ROS network. There are four active topics: /pose, /rosout, /scan and /tf. The default topics: rosout and tf are always present in the ROS network. The other two topics were created as part of the sample network.

```
rostopic list
```

```
/pose
/rosout
/scan
/tf
```

Use rostopic info *<topicname>* to get specific information about a specific topic. The command below shows that /node_1 publishes (sends messages to) the /pose topic, and /node_2 subscribes (receives messages from) to that topic. See Exchange Data with ROS Publishers and Subscribers for more information.

```
rostopic info /pose
```

```
Type: geometry_msgs/Twist

Publishers:
* /node_1 (http://desktop-sqc5nrv:57437/)

Subscribers:
* /node_2 (http://desktop-sqc5nrv:57442/)
```

Use rosnode info *<nodename>* to get information about a specific node. The command below shows that node_1 publishes to /pose, /rosout and /tf topics, subscribes to the /scan topic and provides services: /node_1/get_loggers and /node_1/set_logger_level. The default logging services: get_loggers and set_logger_level are provided by all the nodes created in ROS network.

```
rosnode info /node_1
```

```
Node: [/node_1]
URI: [http://desktop-sqc5nrv:57437/]

Publications (3 Active Topics):
 * /pose
 * /rosout
 * /tf

Subscriptions (1 Active Topics):
 * /scan

Services (2 Active):
 * /node_1/get_loggers
 * /node_1/set_logger_level
```

Messages

Publishers, subscribers, and services use ROS messages to exchange information. Each ROS message has an associated *message type* that defines the datatypes and layout of information in that message (See Work with Basic ROS Messages).

Use rostopic type *<topicname>* to see the message type used by a topic. The command below shows that the /pose topic uses messages of type geometry_msgs/Twist.

```
rostopic type /pose
```

```
geometry_msgs/Twist
```

Use rosmsg show *<messagetype>* to view the properties of a message type. The geometry_msgs/Twist message type has two properties, Linear and Angular. Each property is a message of type geometry_msgs/Vector3, which in turn has three properties of type double.

```
rosmsg show geometry_msgs/Twist
```

```
% This expresses velocity in free space broken into its Linear and Angular parts.
Vector3  Linear
Vector3  Angular
```

```
rosmsg show geometry_msgs/Vector3
```

```
% This represents a vector in free space.
% It is only meant to represent a direction. Therefore, it does not
% make sense to apply a translation to it (e.g., when applying a
% generic rigid transformation to a Vector3, tf2 will only apply the
% rotation). If you want your data to be translatable too, use the
% geometry_msgs/Point message instead.

double X
double Y
double Z
```

Use `rosmsg list` to see the full list of message types available in MATLAB.

Shut Down ROS Network

Use `exampleHelperROSShutDownSampleNetwork` to remove the sample nodes, publishers, and subscribers from the ROS network. This command is only needed if the sample network was created earlier using `exampleHelperROSStartSampleNetwork`.

```
exampleHelperROSShutDownSampleNetwork
```

Use `rosshutdown` to shut down the ROS network in MATLAB. This shuts down the ROS master that was started by `rosinit` and deletes the global node. Using `rosshutdown` is the recommended procedure once you are done working with the ROS network.

```
rosshutdown
```

```
Shutting down global node /matlab_global_node_50463 with NodeURI http://desktop-sqc5nrv:57432/ and MasterURI http:/
Shutting down ROS master on http://172.27.160.1:11311.
```

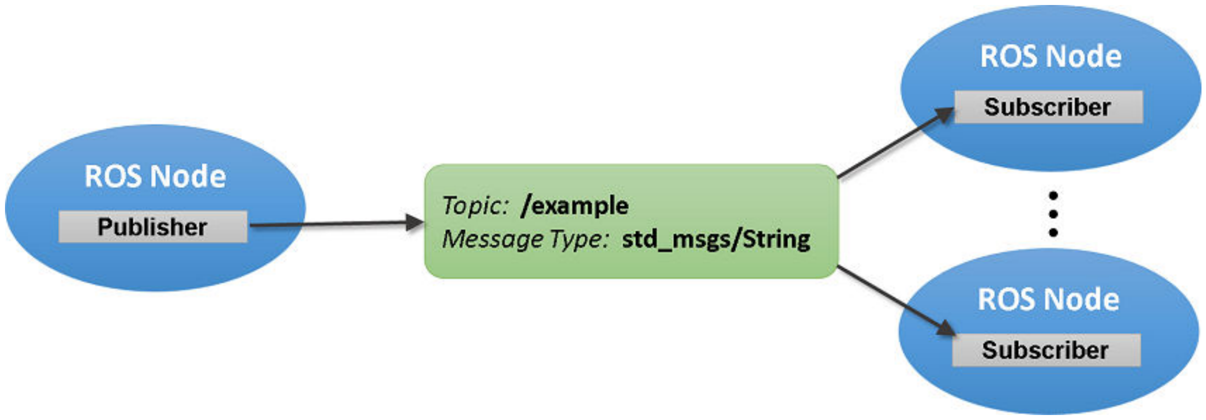
Exchange Data with ROS Publishers and Subscribers

The primary mechanism for ROS nodes to exchange data is sending and receiving *messages*. Messages are transmitted on a *topic,* and each topic has a unique name in the ROS network. If a node wants to share information, it uses a *publisher* to send data to a topic. A node that wants to receive that information uses a *subscriber* to that same topic. Besides its unique name, each topic also has a *message type*, which determines the types of messages that are capable of being transmitted under that topic.

This publisher and subscriber communication has the following characteristics:

- Topics are used for many-to-many communication. Many publishers can send messages to the same topic and many subscribers can receive them.
- Publishers and subscribers are decoupled through topics and can be created and destroyed in any order. A message can be published to a topic even if there are no active subscribers.

The concept of topics, publishers, and subscribers is illustrated in the figure:



This example shows how to publish and subscribe to topics in a ROS network. It also shows how to:

- Wait until a new message is received
- Use callbacks to process new messages in the background

Prerequisites: Get Started with ROS, Connect to a ROS Network

Subscribe and Wait for Messages

Start the ROS master in MATLAB® using the `rosinit` command.

```
rosinit
```

```
Launching ROS Core...
.Done in 1.221 seconds.
Initializing ROS master on http://172.27.160.1:11311.
Initializing global node /matlab_global_node_81714 with NodeURI http://desktop-sqc5nrv:57470/ and MasterURI http://
```

Create a sample ROS network with several publishers and subscribers using the provided helper function `exampleHelperROSCreateSampleNetwork`.

```
exampleHelperROSCreateSampleNetwork
```

Use `rostopic list` to see which topics are available.

```
rostopic list
```

```
/pose
/rosout
/scan
/tf
```

Use `rostopic info` to check if any nodes are publishing to the `/scan` topic. The command below shows that node_3 is publishing to it.

```
rostopic info /scan
```

```
Type: sensor_msgs/LaserScan

Publishers:
* /node_3 (http://desktop-sqc5nrv:57485/)

Subscribers:
* /node_1 (http://desktop-sqc5nrv:57475/)
* /node_2 (http://desktop-sqc5nrv:57480/)
```

Use `rossubscriber` to subscribe to the `/scan` topic. If the topic already exists in the ROS network (as is the case here), `rossubscriber` detects its message type automatically, so you do not need to specify it. Use messages in struct format for better efficiency.

```
laser = rossubscriber("/scan","DataFormat","struct");
pause(2)
```

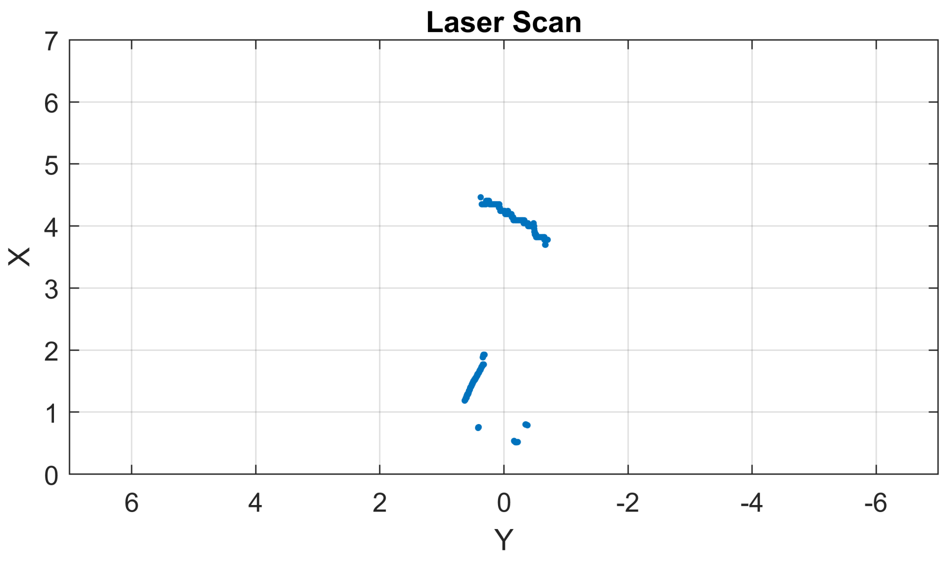
Use `receive` to wait for a new message. (The second argument is a time-out in seconds.) The output `scandata` contains the received message data.

```
scandata = receive(laser,10)
```

```
scandata = struct with fields:
       MessageType: 'sensor_msgs/LaserScan'
            Header: [1×1 struct]
          AngleMin: -0.5467
          AngleMax: 0.5467
    AngleIncrement: 0.0017
     TimeIncrement: 0
          ScanTime: 0.0330
          RangeMin: 0.4500
          RangeMax: 10
            Ranges: [640×1 single]
       Intensities: []
```

Some message types have visualizers associated with them. For the LaserScan message, `plot` plots the scan data. The `MaximumRange` name-value pair specifies the maximum plot range.

```
figure
rosPlot(scandata,"MaximumRange",7)
```



Subscribe Using Callback Functions

Instead of using `receive` to get data, you can specify a function to be called when a new message is received. This allows other MATLAB code to execute while the subscriber is waiting for new messages. Callbacks are essential if you want to use multiple subscribers.

Subscribe to the `/pose` topic, using the callback function `exampleHelperROSPoseCallback`.

```
robotpose = rossubscriber("/pose",@exampleHelperROSPoseCallback,"DataFormat","struct")
```

```
robotpose = 
  Subscriber with properties:

        TopicName: '/pose'
    LatestMessage: []
      MessageType: 'geometry_msgs/Twist'
       BufferSize: 1
    NewMessageFcn: @exampleHelperROSPoseCallback
       DataFormat: 'struct'
```

One way of sharing data between your main workspace and the callback function is to use global variables. Define two global variables `pos` and `orient`.

```
global pos
global orient
```

The global variables pos and orient are assigned in the exampleHelperROSPoseCallback function when new message data is received on the /pose topic.

Wait for a few seconds to make sure that the subscriber can receive messages. The most current position and orientation data will always be stored in the pos and orient variables.

```
pause(2)
pos
```

```
pos =

     []
```

```
orient
```

```
orient =

     []
```

If you type in pos and orient a few times in the command line, you can see that the values are continuously updated.

Stop the pose subscriber by clearing the subscriber variable

```
clear robotpose
```

Note: There are other ways to extract information from callback functions besides using globals. For example, you can pass a handle object as additional argument to the callback function. See the Callback Definition documentation for more information about defining callback functions.

Publish Messages

Create a publisher that sends ROS string messages to the /chatter topic (see Work with Basic ROS Messages).

```
chatterpub = rospublisher("/chatter","std_msgs/String","DataFormat","struct")
```

```
chatterpub = 
  Publisher with properties:

         TopicName: '/chatter'
    NumSubscribers: 0
        IsLatching: 1
       MessageType: 'std_msgs/String'
        DataFormat: 'struct'
```

```
pause(2) % Wait to ensure publisher is registered
```

Create and populate a ROS message to send to the /chatter topic.

```
chattermsg = rosmessage(chatterpub);
chattermsg.Data = 'hello world'
```

```
chattermsg = struct with fields:
    MessageType: 'std_msgs/String'
           Data: 'hello world'
```

Use `rostopic list` to verify that the `/chatter` topic is available in the ROS network.

```
rostopic list
```

```
/chatter
/pose
/rosout
/scan
/tf
```

Define a subscriber for the `/chatter` topic. `exampleHelperROSChatterCallback` is called when a new message is received and displays the string content in the message.

```
chattersub = rossubscriber("/chatter",@exampleHelperROSChatterCallback,"DataFormat","struct")
```

```
chattersub = 
  Subscriber with properties:

        TopicName: '/chatter'
    LatestMessage: []
      MessageType: 'std_msgs/String'
       BufferSize: 1
    NewMessageFcn: @exampleHelperROSChatterCallback
       DataFormat: 'struct'
```

Publish a message to the `/chatter` topic. The string is displayed by the subscriber callback.

```
send(chatterpub,chattermsg)
pause(2)
```

The `exampleHelperROSChatterCallback` function was called as soon as you published the string message.

Shut Down ROS Network

Remove the sample nodes, publishers, and subscribers from the ROS network. Clear the global variables `pos` and `orient`.

```
exampleHelperROSShutDownSampleNetwork
clear global pos orient
```

Shut down the ROS master and delete the global node.

```
rosshutdown
```

```
Shutting down global node /matlab_global_node_81714 with NodeURI http://desktop-sqc5nrv:57470/ and MasterURI http:/
Shutting down ROS master on http://172.27.160.1:11311.
```

Get Started with Gazebo and Simulated TurtleBot

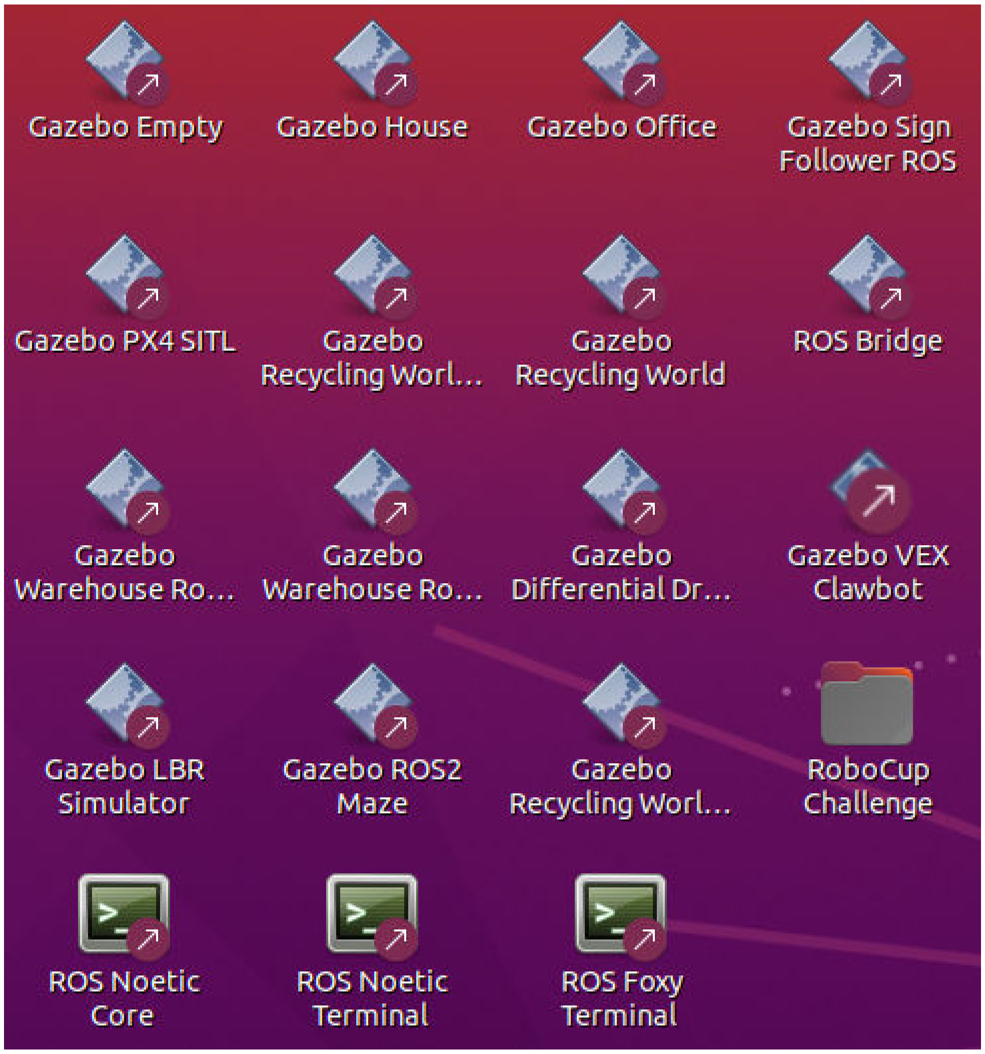
This example shows how to set up the Gazebo® simulator engine. This example prepares you for further exploration with Gazebo and also for exploration with a simulated TurtleBot®.

Gazebo is a simulator that allows you to test and experiment realistically with physical scenarios. Gazebo is a useful tool in robotics because it allows you to create and run experiments rapidly with solid physics and good graphics. MATLAB® connects to Gazebo through the ROS interface.

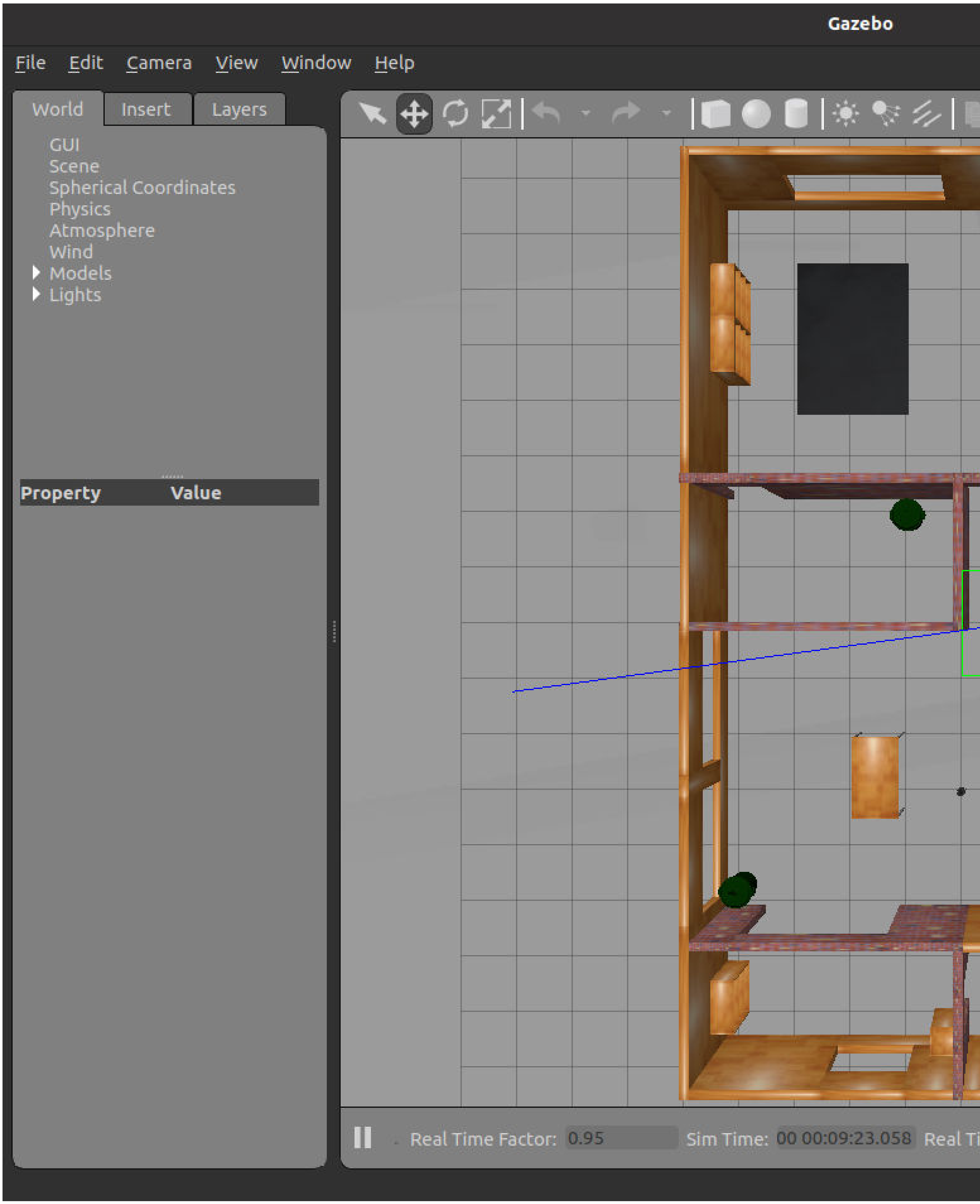
Download Virtual Machine

You can download a virtual machine image that already has ROS and Gazebo installed. This virtual machine is based on Ubuntu® Linux® and is pre-configured to support the examples in ROS Toolbox™.

- Download and install the ROS Virtual Machine.
- Launch the virtual machine.
- On the Ubuntu desktop you see multiple Gazebo world start-up scripts, as well as other utility shortcuts. For the TurtleBot® examples, use the **Gazebo Empty**, **Gazebo House**, **Gazebo Office**, or **Gazebo Sign Follower ROS** icons.

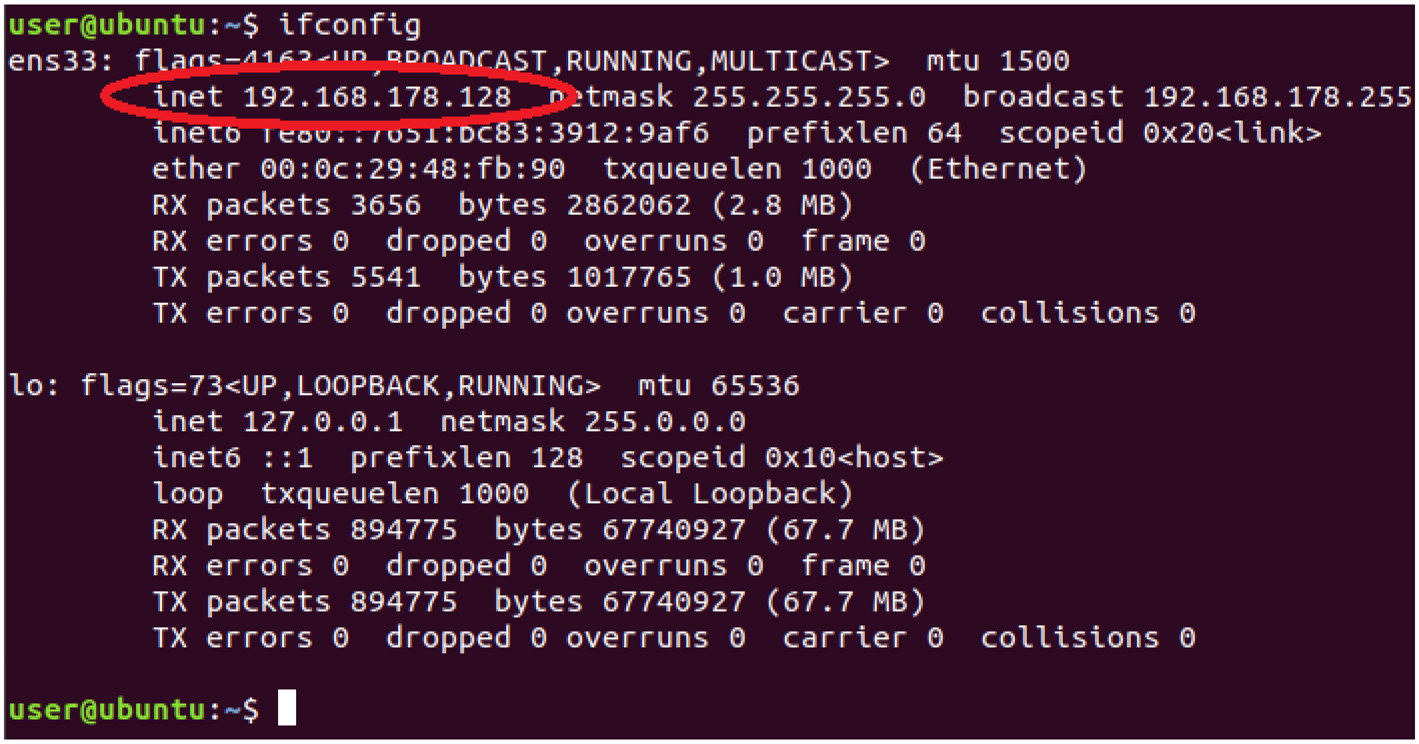


- Click **Gazebo House**. A world opens.


Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property
Value
Real Time Factor:
0.95
Sim Time:
00 00:09:23.058
Real Ti

Note: If the Gazebo screen looks entirely black, refresh the image by minimizing it and then maximizing it.

- Open a new terminal in the Ubuntu virtual machine.
- Type `ifconfig` and return to see the networking information for the virtual machine.
- Under `eth0`, the `inet addr` displays the IP address for the virtual machine.

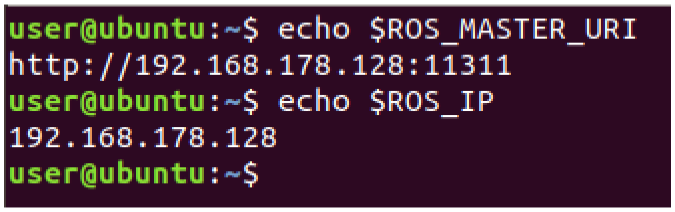


```
user@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.178.128  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 fe80::7651:bc83:3912:9af6  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:48:fb:90  txqueuelen 1000  (Ethernet)
        RX packets 3656  bytes 2862062 (2.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5541  bytes 1017765 (1.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 894775  bytes 67740927 (67.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 894775  bytes 67740927 (67.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

user@ubuntu:~$
```

- Two ROS environment variables must be set to set up the network: ROS_MASTER_URI and ROS_IP. If you are using the demos from the desktop of the Linux® virtual machine, these variables are usually automatically set at startup.
- **(Optional) If you are using your own virtual machine** set up the variables by executing the following commands in the terminal. **Replace `IP_OF_VM` with the IP address retrieved through `ifconfig`:**

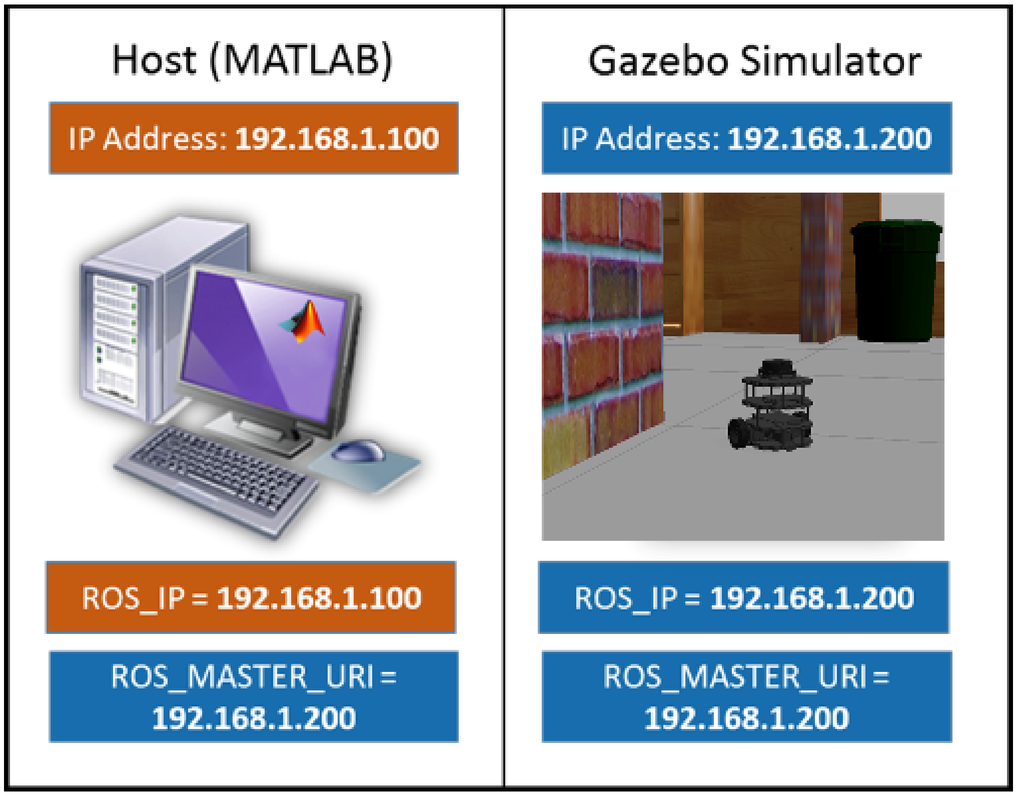
```
echo export ROS_MASTER_URI=http://IP_OF_VM:11311 >> ~/.bashrc
echo export ROS_IP=IP_OF_VM >> ~/.bashrc
```

- Check the environment variables using `echo $ENV_VAR` (replacing ENV_VAR with the appropriate environment variable). You can close and reopen your terminal for it to take effect.



```
user@ubuntu:~$ echo $ROS_MASTER_URI
http://192.168.178.128:11311
user@ubuntu:~$ echo $ROS_IP
192.168.178.128
user@ubuntu:~$
```

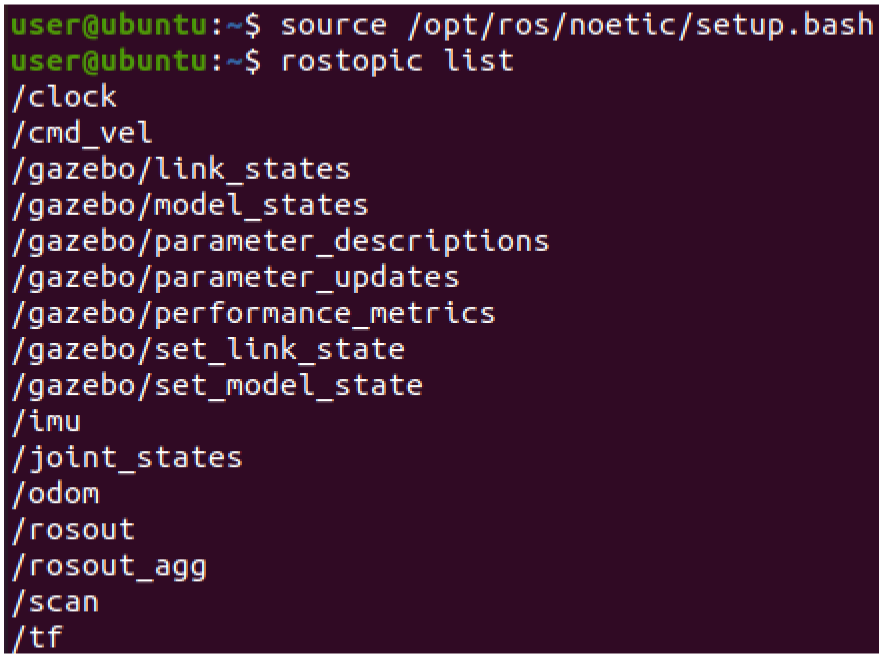
- The following diagram illustrates correct environment variable assignments (with fake IP addresses)


Host (MATLAB)
IP Address: 192.168.1.100
ROS_IP = 192.168.1.100
ROS_MASTER_URI = 192.168.1.200
Gazebo Simulator
IP Address: 192.168.1.200
ROS_IP = 192.168.1.200
ROS_MASTER_URI = 192.168.1.200

Connect to an Existing Gazebo Simulator

If you already have Gazebo running on a Linux distribution, set up the simulator as described here:

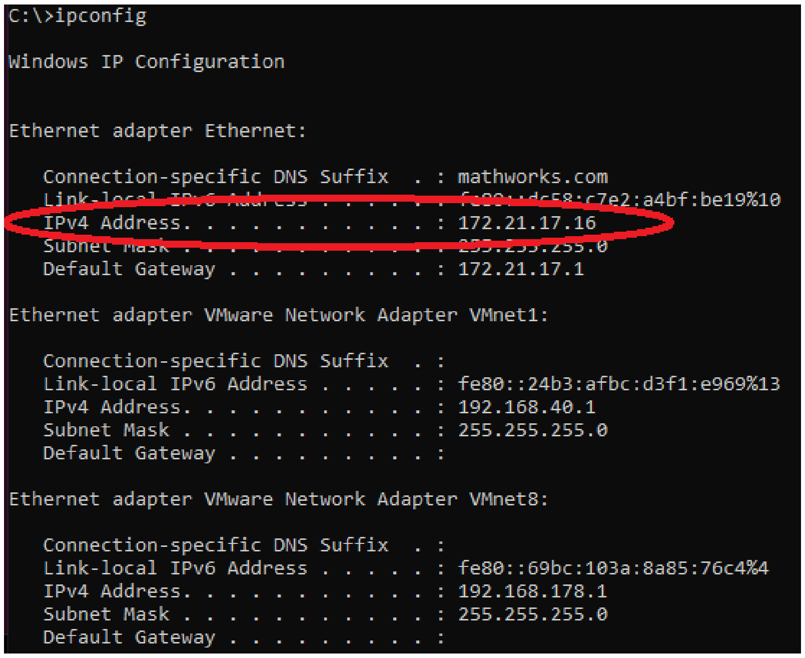
- On the ROS website, download the appropriate packages for TurtleBot.
- Follow the instructions on the ROS website to get the TurtleBot running in a simulated Gazebo environment.
- Make sure the environment variables are appropriately set and that you can ping back and forth between your host computer and the Gazebo computer. There are many ways to set up the network. The Connect to a ROS Network example contains tips on how to verify connectivity between devices in the ROS network.
- To use any ROS commands in the Linux machine terminals, the terminal environment needs to be set to use the proper ROS installation. Source the appropriate ROS environment setup script in the terminal before running any ROS commands. In the VM, the command is: `source /opt/ros/noetic/setup.bash`
- Make sure you have access to the following topics. In the terminal on the Linux machine, enter `rostopic list` to see the at least these available topics.



```
user@ubuntu:~$ source /opt/ros/noetic/setup.bash
user@ubuntu:~$ rostopic list
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/imu
/joint_states
/odom
/rosout
/rosout_agg
/scan
/tf
```

Host Computer Setup

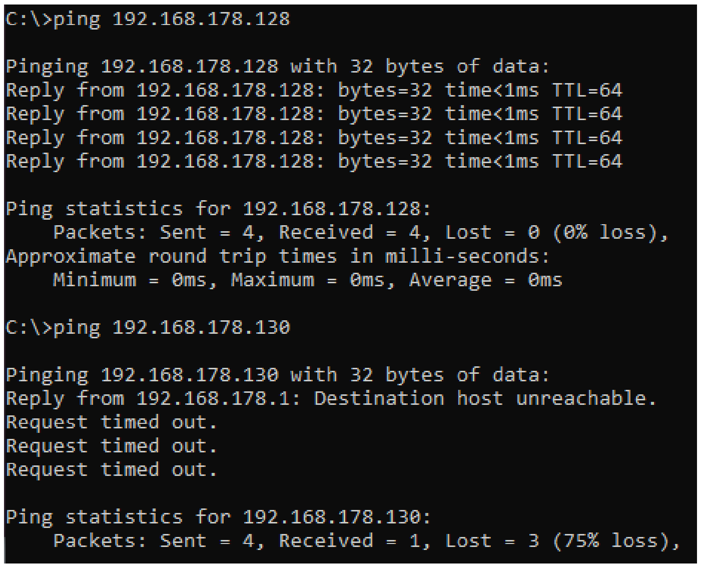
- Find the IP address of your host computer on the network. On a Windows® machine, at the command prompt, type `ipconfig`. On a Mac or Linux machine, open a terminal and type `ifconfig`. An example of `ipconfig` is shown.



```
C:\>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : mathworks.com
   Link-local IPv6 Address . . . . . : fe80::dc58:c7e2:a4bf:be19%10
   IPv4 Address. . . . . . . . . . . : 172.21.17.16
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.21.17.1

Ethernet adapter VMware Network Adapter VMnet1:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::24b3:afbc:d3f1:e969%13
   IPv4 Address. . . . . . . . . . . : 192.168.40.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter VMware Network Adapter VMnet8:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::69bc:103a:8a85:76c4%4
   IPv4 Address. . . . . . . . . . . : 192.168.178.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Note: The connection type can vary depending on how you are connected to the laptop. In this case you use the Ethernet, however, in many cases the wireless (wlan) is the appropriate connection.

- Ping the simulator machine `ping IP_OF_VM`. A successful `ping` is shown first, followed by an unsuccessful `ping`.



```
C:\>ping 192.168.178.128

Pinging 192.168.178.128 with 32 bytes of data:
Reply from 192.168.178.128: bytes=32 time<1ms TTL=64
Reply from 192.168.178.128: bytes=32 time<1ms TTL=64
Reply from 192.168.178.128: bytes=32 time<1ms TTL=64
Reply from 192.168.178.128: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.178.128:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>ping 192.168.178.130

Pinging 192.168.178.130 with 32 bytes of data:
Reply from 192.168.178.1: Destination host unreachable.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.178.130:
    Packets: Sent = 4, Received = 1, Lost = 3 (75% loss),
```

Communicate with the TurtleBot

This example introduces the TurtleBot® platform and the ways in which MATLAB® users can interact with it. Specifically, the code in this example demonstrates how to publish messages to the TurtleBot (such as velocities) and how to subscribe to topics that the TurtleBot publishes (such as odometry).

The TurtleBot must be running for this example to work.

Prerequisites: Get Started with Gazebo and a Simulated TurtleBot or Get Started with a Real TurtleBot

Connect to the TurtleBot

The TurtleBot must be running. If you are using a real TurtleBot and followed the hardware setup steps in Get Started with a Real TurtleBot, the robot is running. If you are using a TurtleBot in simulation and followed the setup steps in Get Started with Gazebo and a Simulated TurtleBot, launch one of the Gazebo® worlds from the desktop (`Gazebo Office`, for instance).

In your MATLAB instance on the host computer, run the following command. Replace `ipaddress` with the IP address of the TurtleBot. This line initializes ROS and connects to the TurtleBot.

```
ipaddress = "http://192.168.38.131:11311";
rosinit(ipaddress)
```

```
Initializing global node /matlab_global_node_90045 with NodeURI http://192.168.38.1:57508/ and MasterURI http://192
```

If the network you are using to connect to the TurtleBot is not your default network adapter, you can manually specify the IP address of the adapter that is used to connect to the robot. This might happen if you use a Wireless network, but also have an active Ethernet connection. Replace `IP_OF_TURTLEBOT` with the IP address of the TurtleBot and `IP_OF_HOST_COMPUTER` with the IP address of the host adapter that is used to connect to the robot:

```
rosinit("IP_OF_TURTLEBOT","NodeHost","IP_OF_HOST_COMPUTER");
```

Display all the available ROS topics using:

```
rostopic list
```

If you do not see any topics, then the network has not been set up properly. Refer to the beginning of this document for network setup steps.

Move the Robot

You can control the movement of the TurtleBot by publishing a message to the `/cmd_vel` topic. The message has to be of type `geometry_msgs/Twist`, which contains data specifying desired linear and angular velocities. The TurtleBot's movements can be controlled through two different values: the linear velocity along the *X*-axis controls forward and backward motion and the angular velocity around the *Z*-axis controls the rotation speed of the robot base.

Set a variable `velocity` to use for a brief TurtleBot movement.

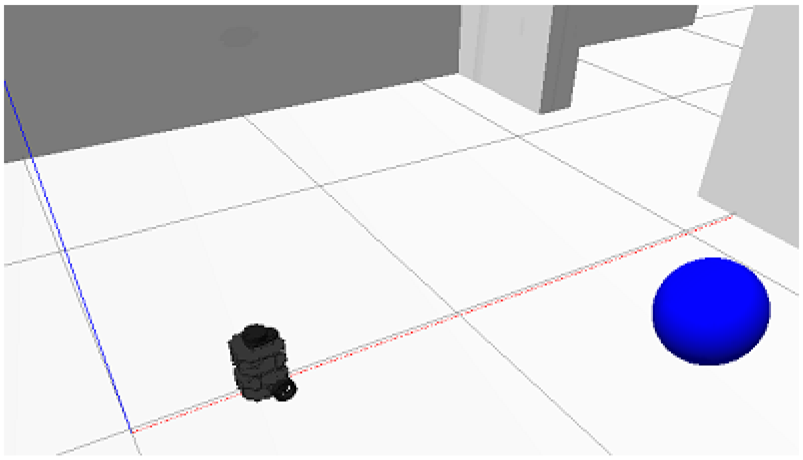
```
velocity = 2;     % meters per second
```

Create a publisher for the `/cmd_vel` topic and the corresponding message containing the velocity values. Use the publisher with structure format messages for better performance.

```
robotCmd = rospublisher("/cmd_vel","DataFormat","struct") ;
velMsg = rosmessage(robotCmd);
```

Set the forward velocity (along the *X*-axis) of the robot based on the `velocity` variable and publish the command to the robot. Let it move for a moment, and then bring it to a stop.

```
velMsg.Linear.X = velocity;
send(robotCmd,velMsg)
pause(4)
velMsg.Linear.X = 0;
send(robotCmd,velMsg)
```



To view the type of the message published by the velocity topic, execute the following:

```
rostopic type /cmd_vel
```

```
geometry_msgs/Twist
```

The topic expects messages of type `geometry_msgs/Twist`, which is exactly the type of the `velMsg` created above.

To view which nodes are publishing and subscribing to a given topic, use the command: `rostopic info TOPICNAME`. The following command lists the publishers and subscribers for the velocity topic. MATLAB is listed as one of the publishers.

```
  rostopic info /cmd_vel
```

```
Type: geometry_msgs/Twist

Publishers:
* /matlab_global_node_90045 (http://192.168.38.1:57508/)

Subscribers:
* /gazebo (http://192.168.38.131:43977/)
```

Receive Robot Position and Orientation

The TurtleBot uses the `/odom` topic to publish its current position and orientation (collectively denoted as pose). Since the TurtleBot is not equipped with a GPS system, the pose will be relative to the pose that the robot had when it was first turned on.

Create a subscriber for the odometry messages

```
odomSub = rossubscriber("/odom","DataFormat","struct");
```

Wait for the subscriber to return data, then extract the data and assign it to variables `x`, `y`, and `z`:

```
odomMsg = receive(odomSub,3);
```

```
pose = odomMsg.Pose.Pose;
x = pose.Position.X;
y = pose.Position.Y;
z = pose.Position.Z;
```

Note: If you see an error, then it is likely that the `receive` command timed out. Make sure that odometry is being published and that your network is set up properly.

Display the *x*, *y*, and *z* values

```
[x y z]
```

```
ans = 1×3
   -3.0003    1.0016   -0.0010
```

The orientation of the TurtleBot is stored as a quaternion in the `Orientation` property of `pose`. Use `quat2eul` to convert into the more convenient representation of Euler angles. To display the current orientation, `theta`, of the robot in degrees, execute the following lines.

```
quat = pose.Orientation;
angles = quat2eul([quat.W quat.X quat.Y quat.Z]);
theta = rad2deg(angles(1))
```

```
theta = 1.4078
```

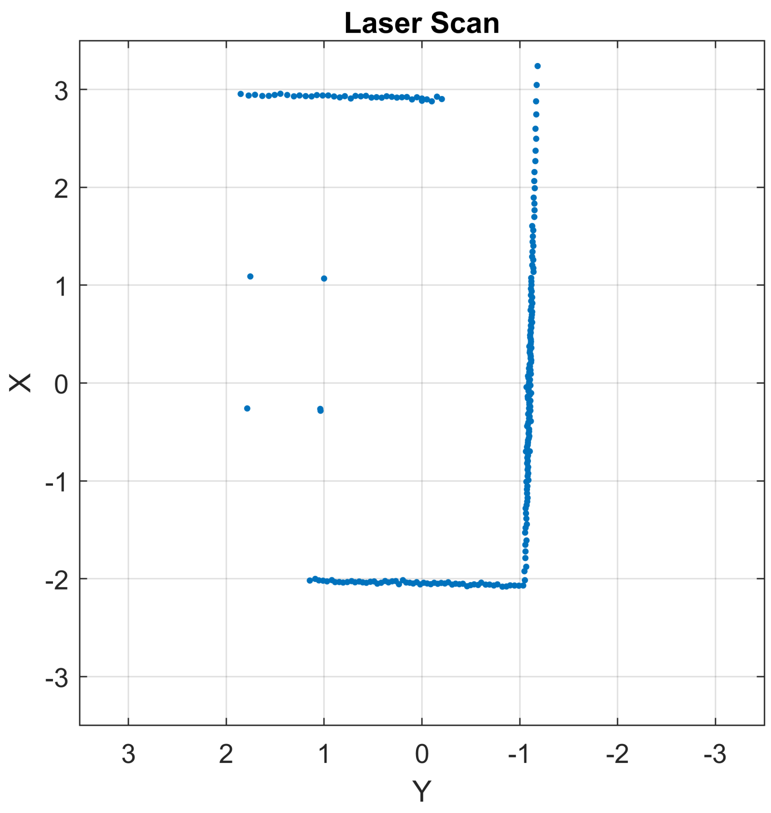
Receive Lidar Data

Subscribe to the lidar topic:

```
lidarSub = rossubscriber("/scan","DataFormat","struct");
```

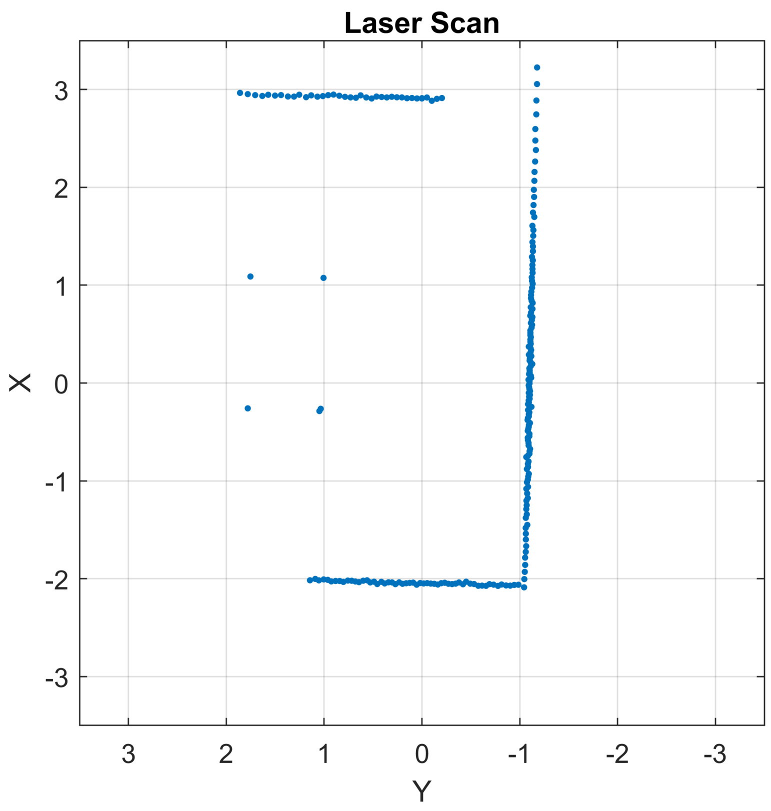
After subscribing to the lidar topic, wait for the data and then display it with rosPlot.

```
scanMsg = receive(lidarSub);
figure
rosPlot(scanMsg)
```



To continuously display updating lidar scans while the robot turns for a short duration, use the following while loop:

```
velMsg.Angular.Z = velocity;
send(robotCmd,velMsg)
tic
while toc < 20
  scanMsg = receive(lidarSub);
  rosPlot(scanMsg)
end
```



```
velMsg.Angular.Z = 0;
send(robotCmd,velMsg)
```

Disconnect from the Robot

Clear the workspace of publishers, subscribers, and other ROS-related objects when you are finished with them.

```
clear
```

Use `rosshutdown` once you are done working with the ROS network. Shut down the global node and disconnect from the TurtleBot.

```
rosshutdown
```

```
Shutting down global node /matlab_global_node_90045 with NodeURI http://192.168.38.1:57508/ and MasterURI http://19
```