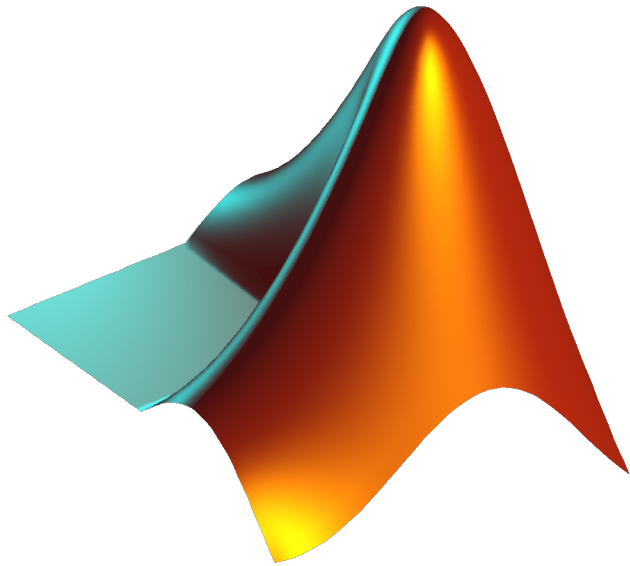
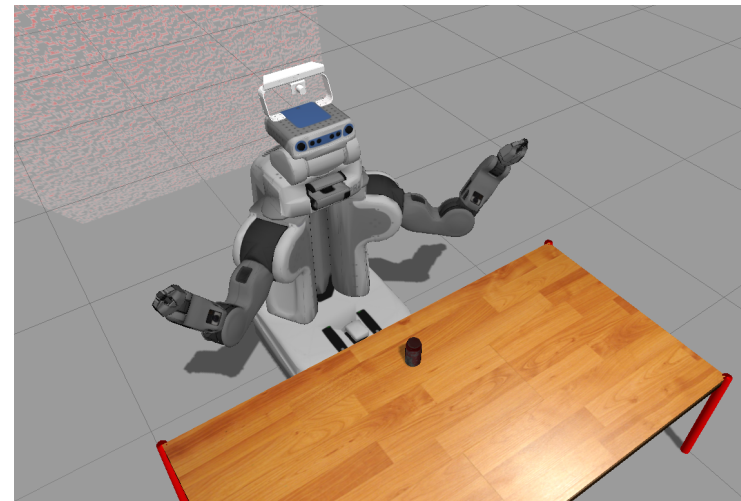
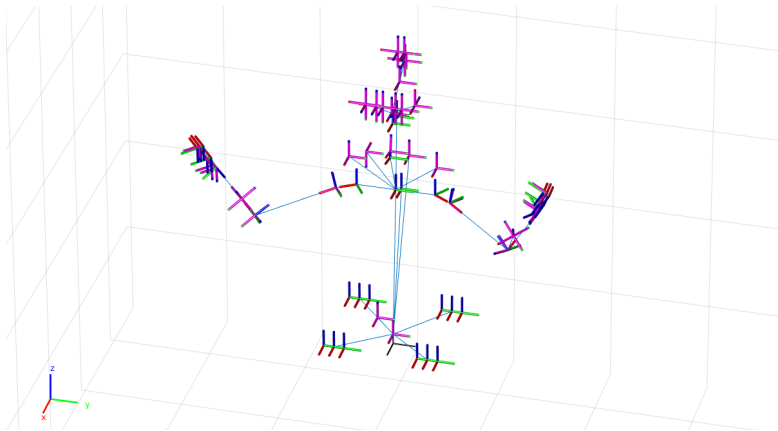


# ROS MATLAB



 ROS



# Conectando na rede ROS e testando os comandos

- Rode o roscore para iniciar o master
- Atribuindo “alias” nos comandos no .bashrc

Exemplo:

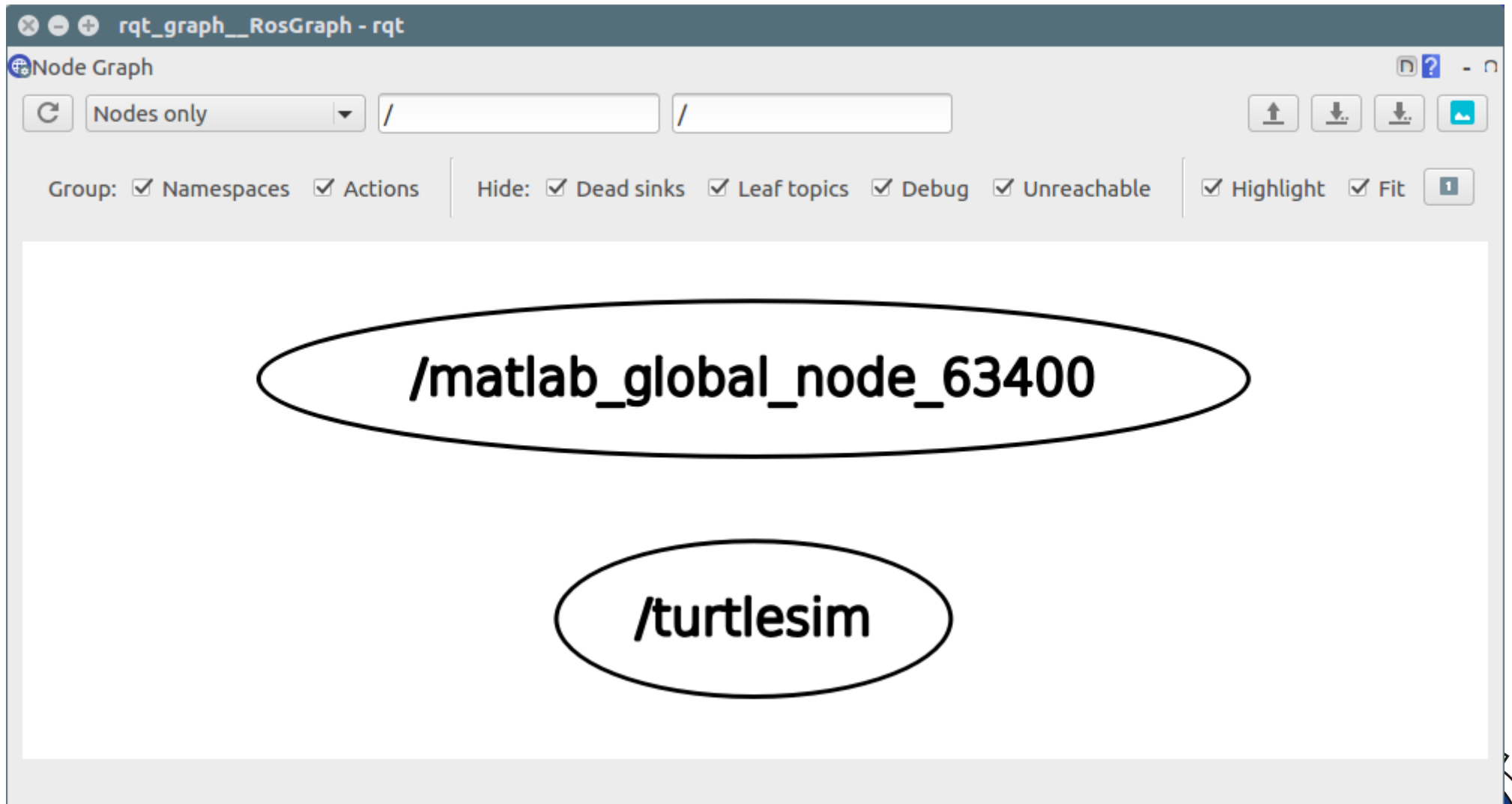
– alias bixo='roslaunch turtlesim turtlesim\_node'

- Iniciando ROS no matlab

**ip = '172.16.21.128'      %seu ip na máquina**  
**rosinit(ip)**



# Visualizando no RQT



# Interagindo pelo matlab

- Criando um publisher dentro do nó global do matlab:

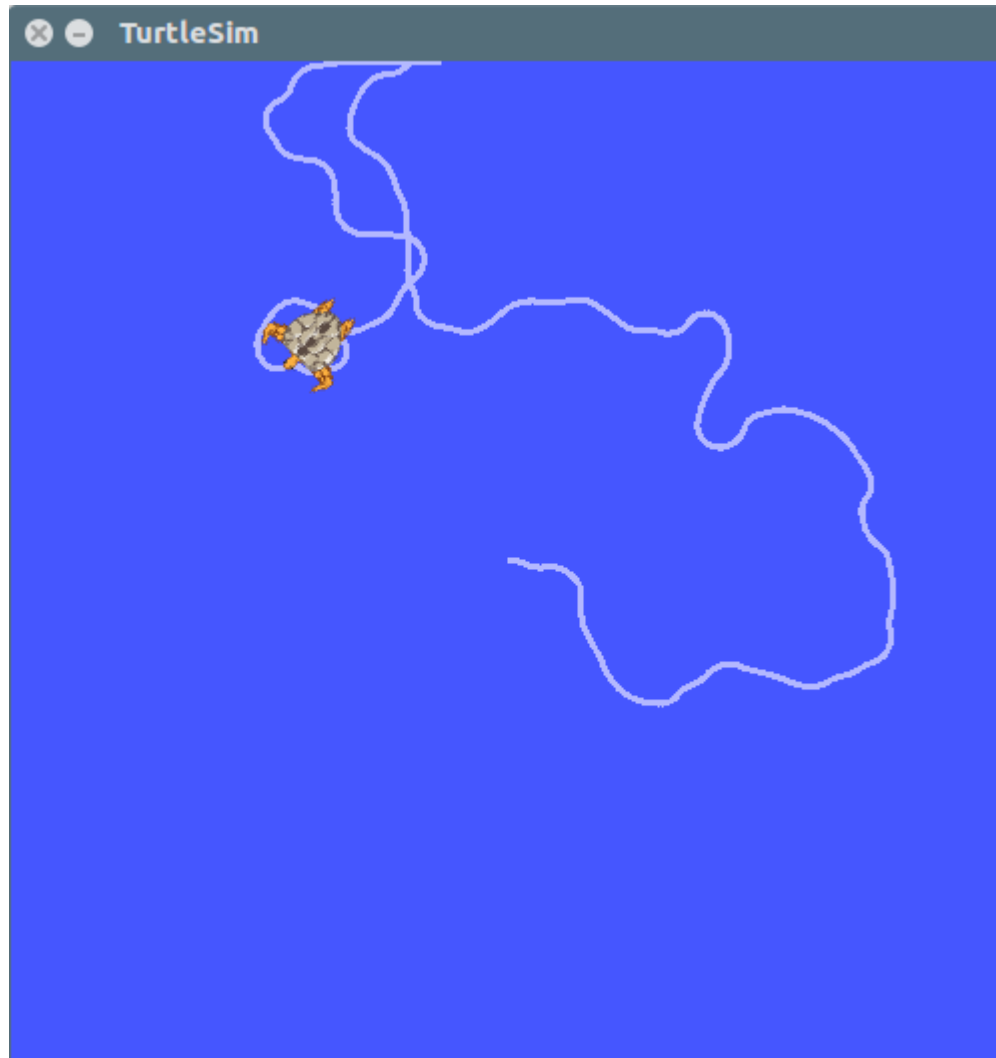
```
Command Window
New to MATLAB? See resources for Getting Started.

>> publisher = rospublisher('/turtle1/cmd_vel');
>> msg = rosmessage(publisher);
>> msg.Linear.X = 2;
fx >> while true
    msg.Angular.Z = (rand * 2 - 1)*10;
    send(publisher,msg);
    pause(0.1);
end
```

A msg tem velocidade linear constante e velocidade angular aleatória (rand gera um aleatório de 0 a 1)



# EITA



# Registrando um subscriber

```
Command Window
New to MATLAB? See resources for Getting Started.
>> subscriber = rossubscriber('/turtle1/pose');
>> msg = rosmessage(subscriber);
>> msg = receive(subscriber)

msg =

ROS Pose message with properties:

    MessageType: 'turtlesim/Pose'
           X: 5.7762
           Y: 5.9897
        Theta: -1.2708
LinearVelocity: 0
AngularVelocity: 0

Use showdetails to show the contents of the message

fx >>
```



# Códigos do github

- **Criar pasta ~/Documents/MATLAB**
  - `mkdir ~/Documents/MATLAB`
- **Clonar repositório dentro da pasta MATLAB**
  - `git clone https://github.com/gustavollps/ROSMatlab.git`



# Criação de nós dedicados

Exemplo com um publisher e um subscriber

basicNode.m

```
clear;
```

```
ip = '127.0.0.1';
```

```
%cria node (nome, ip_do_master)
```

```
node_1 = robotics.ros.Node('node_1', ip);
```

```
%registra o publisher
```

```
twistPub = robotics.ros.Publisher(node_1, '/pose', 'geometry_msgs/Twist');
```

```
pause(1);
```

```
%gera objeto de mensagem para o tópico especificado
```

```
twistPubmsg = rosmessage(twistPub);
```

```
%carrega mensagem
```

```
twistPubmsg.Linear.X = 2;
```

```
twistPubmsg.Linear.Y = 1;
```

```
twistPubmsg.Linear.Z = 0;
```

```
%registra o subscriber
```

```
twistSub = robotics.ros.Subscriber(node_1, '/pose');
```

```
pause(1); %tempo para garantir subscribe no tópico
```

```
send(twistPub, twistPubmsg);
```

```
pause(0.1); %tempo para callback ser chamada e a mensagem recebida
```

```
data = twistSub.LatestMessage %ou data = receive(twistSub)
```





# Parabéns! Você criou um node inútil que só fala com ele mesmo

`/matlab_global_node_63400`

`/pose`



# Publisher dedicado

## basicPublisher.m

```
ip = '127.0.0.1';

%cria node (nome, ip_do_master)
node_pub = robotics.ros.Node('node_publisher', ip);

%cria publisher (nome, topic, tipo_de_msg)
basicPub = robotics.ros.Publisher(node_pub, '/data', 'std_msgs/Float32');

%cria mensagem (objeto_publisher)
basicMsg = rosmessage(basicPub);

%carrega msg
basicMsg.Data = 1.54123;

%publica msg
send(basicPub, basicMsg);
```



# Subscriber dedicado

## basicSubscriber.m

```
ip = '127.0.0.1';

%cria node (nome, ip_do_master)
node_sub = robotics.ros.Node('node_subscriber', ip);

%registra subscriber com callBack
basicSub = robotics.ros.Subscriber(node_sub, '/data', @callBack);
```

## callBack.m

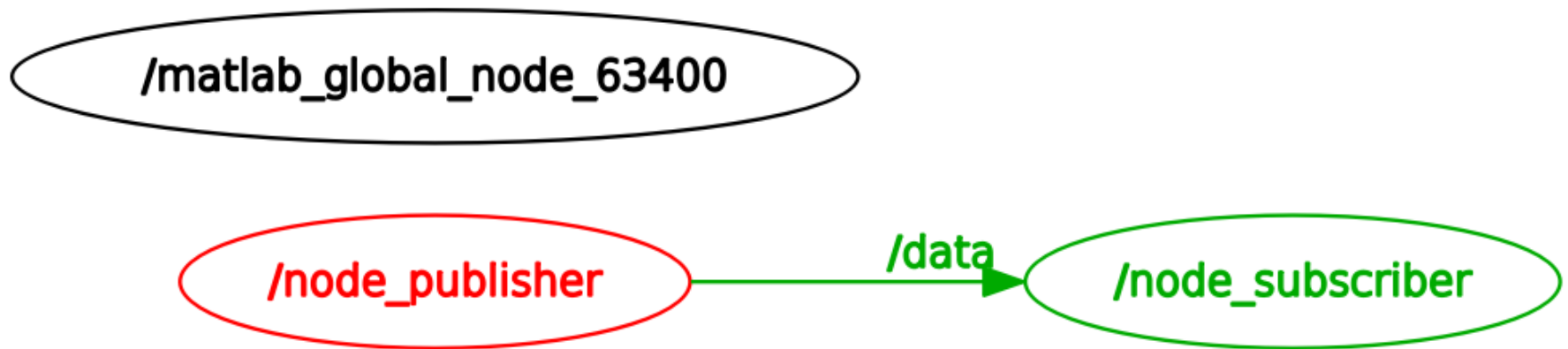
```
function callBack(~,msg)
    %cria variável global para ser usada fora
    global data;

    %recebe informação na variavel global
    data = msg.Data;

    %debug pra saber que a callBack foi chamada (só para teste)
    disp('Chamou a callBack')
end
```



# rqt\_graph



# Serviços

## ServiceNode.m

```
ip = '127.0.0.1';  
node_service = robotics.ros.Node('node_service', ip);  
  
%cria server do serviço (node, nome_do_serviço, tipo, callBack)  
service_server =  
robotics.ros.ServiceServer(node_service, '/le_service', 'roscpp_tutorials/TwoInts', @serviceCallBack)  
  
%cria cliente do serviço (node, serviço)  
service_client = robotics.ros.ServiceClient(node_service, '/le_service')  
  
%cria msg do serviço para ser enviado na chamada  
request = rosmesssage(service_client)  
  
%carrega msg de requisição  
request.A = 1  
request.B = 2
```

## serviceCallBack.m

```
function resp = serviceCallBack(~, req, resp)  
    disp('HELLO')  
    resp.Sum = req.A + req.B;  
end
```



# Timer

- **Gera a execução automática de uma tarefa especificada a cada  $T$  [s] escolhido**
- **Executa em paralelo com o código principal**



# Timer

```
clear;
```

```
ip = '127.0.0.1'; %ip
```

```
node_timer = robotics.ros.Node('node_timer', ip);
```

```
timerPub = robotics.ros.Publisher(node_timer, '/timer_topic', 'std_msgs/String')
```

```
timerMsg = rosmessage(timerPub);
```

```
%cria struct como parametro para passar para o timer, no caso o publisher
```

```
param.pub = timerPub;
```

```
%variavel global para ser usada dentro do timer
```

```
global i;
```

```
i=0;
```

```
%cria timer (time_delay, {callback, param1, param2, ...})
```

```
rosTimer = ExampleHelperROSTimer(1, {@timerCallback,param});
```



# Plot em realtime

```
clear;
```

```
%cria subscriber
```

```
subscribe = rossubscriber('/turtle1/pose',@callback);  
pause(2);
```

```
%variavel global para armazenar msg do tópico /turtle1/pose (atualizada na  
%callBack)
```

```
global pose;
```

```
%fechar figuras anteriores
```

```
close all;
```

```
%nova figura
```

```
fig = figure;
```

```
hold on
```





# Plot em realtime

```
%define limites dos eixos X e Y
```

```
xlim([0 11.1]);
```

```
ylim([0 11.1]);
```

```
while true
```

```
    clf(fig);
```

```
    scatter(pose.X,pose.Y,") %plot ponto com a localização
```

```
%define limites dos eixos X e Y
```

```
xlim([0 11.1]);
```

```
ylim([0 11.1]);
```

```
%pausa para não jogar um loop INFINITO-INFINITAMENTE rápido
```

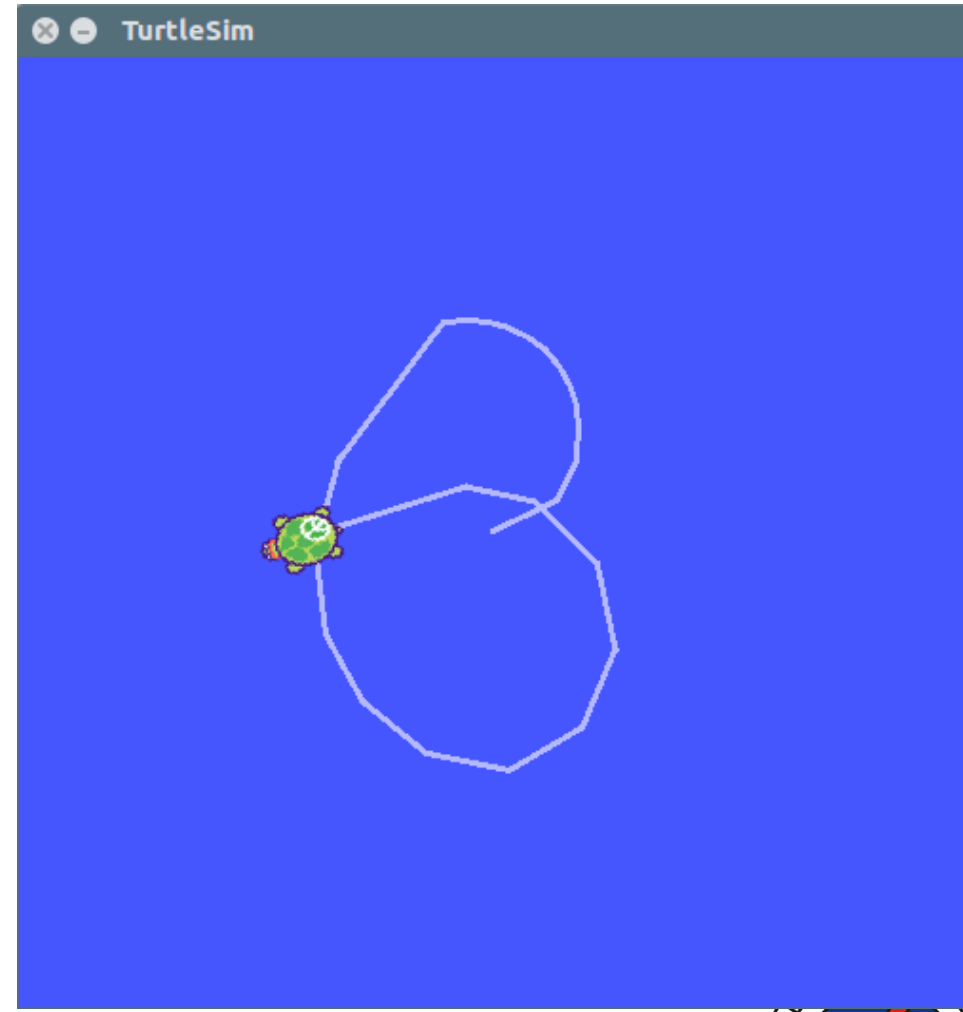
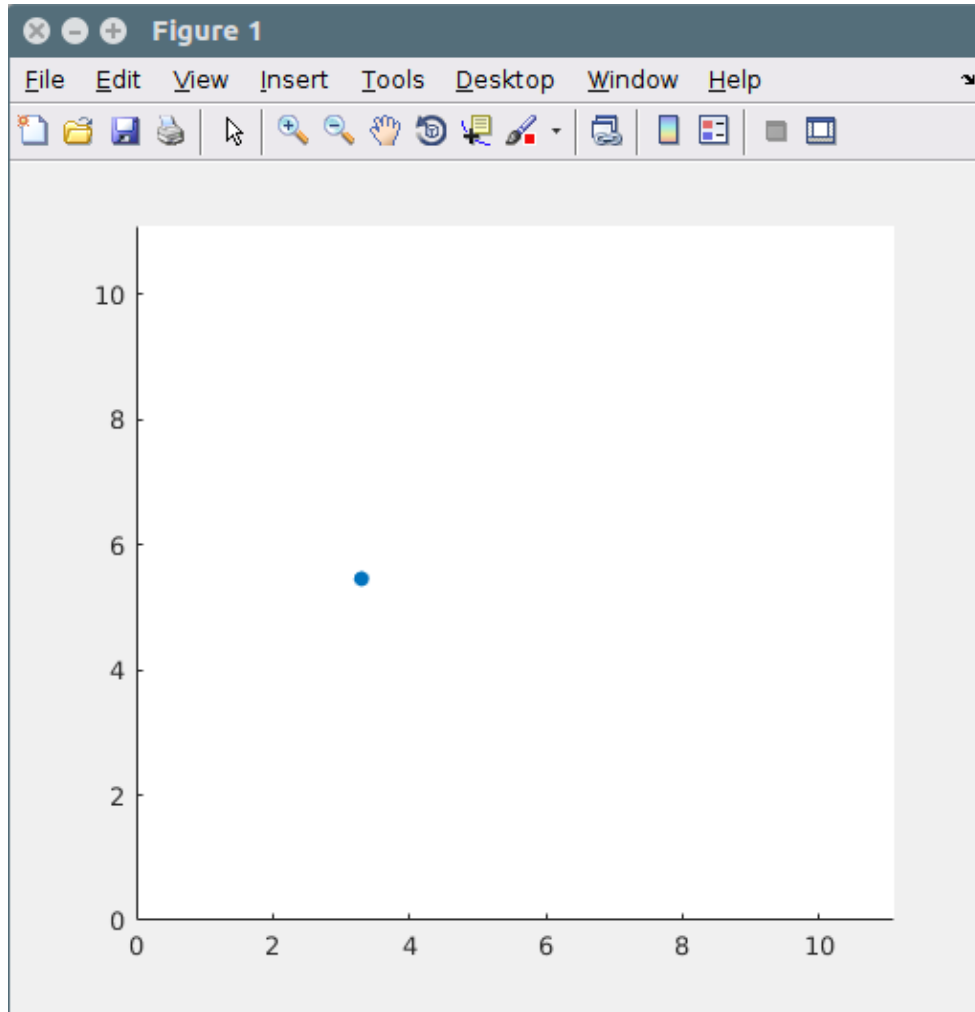
```
    pause(0.05);
```

```
end[0 11.1]);
```

```
end
```

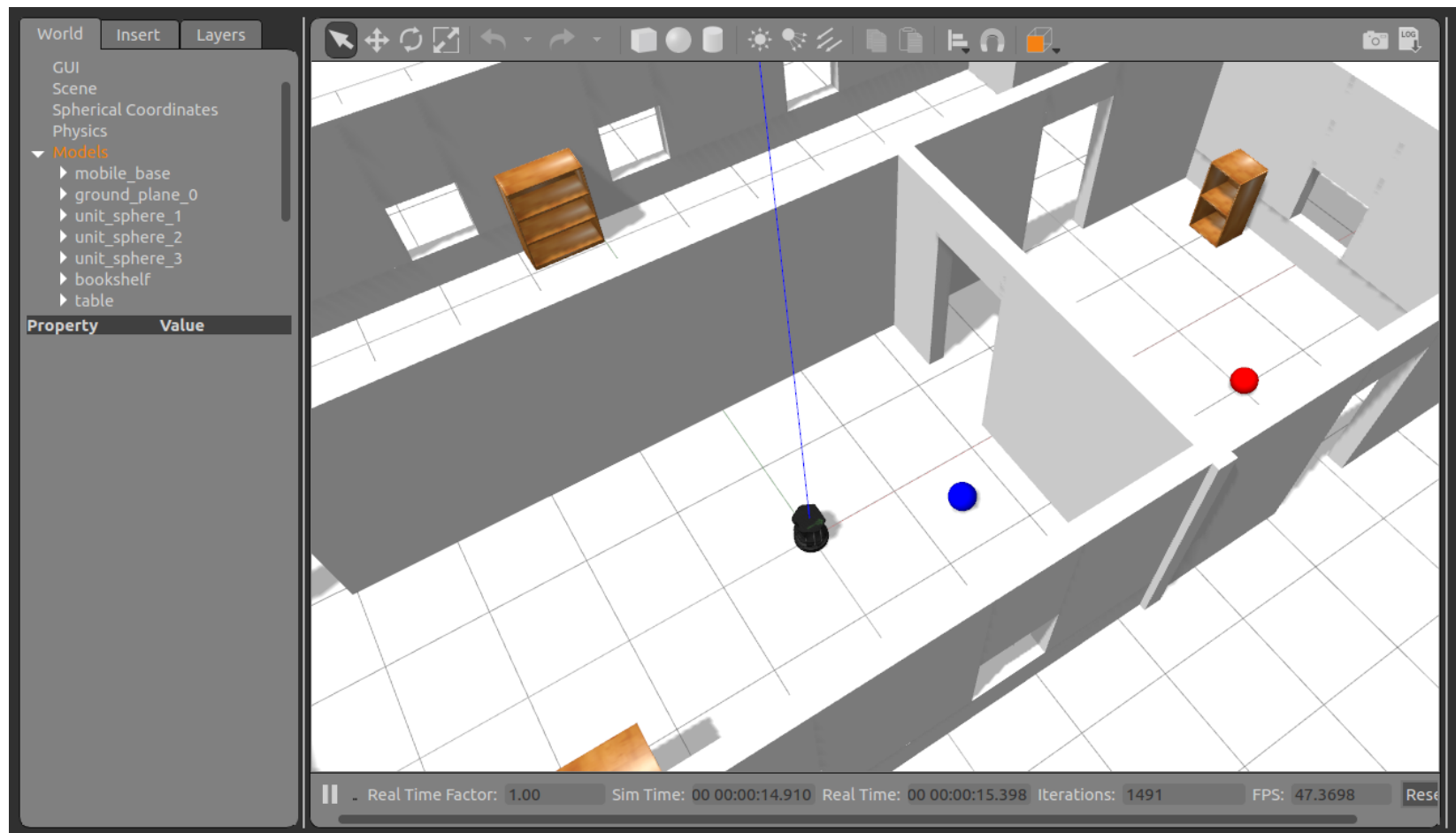


# Plot em realtime



# Simulações no Gazebo

- Abrir Gazebo TurtleBot World



# Tópicos de interesse

- **/camera/rgb/image\_raw**

- Tópico com imagem da câmera do robô

MSG: sensor\_msgs/Image

- **/mobile\_base/commands/velocity**

- Controle de velocidade do robô

MSG: geometry\_msgs/Twist

- **/odom**

- Odometria do robô (posição estimada)

MSG: nav\_msgs/Odometry

(NOTA – usar `quat2eul(msg.Pose.Pose.Orientation.readQuaternion)` para converter a msg em ângulo HUMANAMENTE entendível)

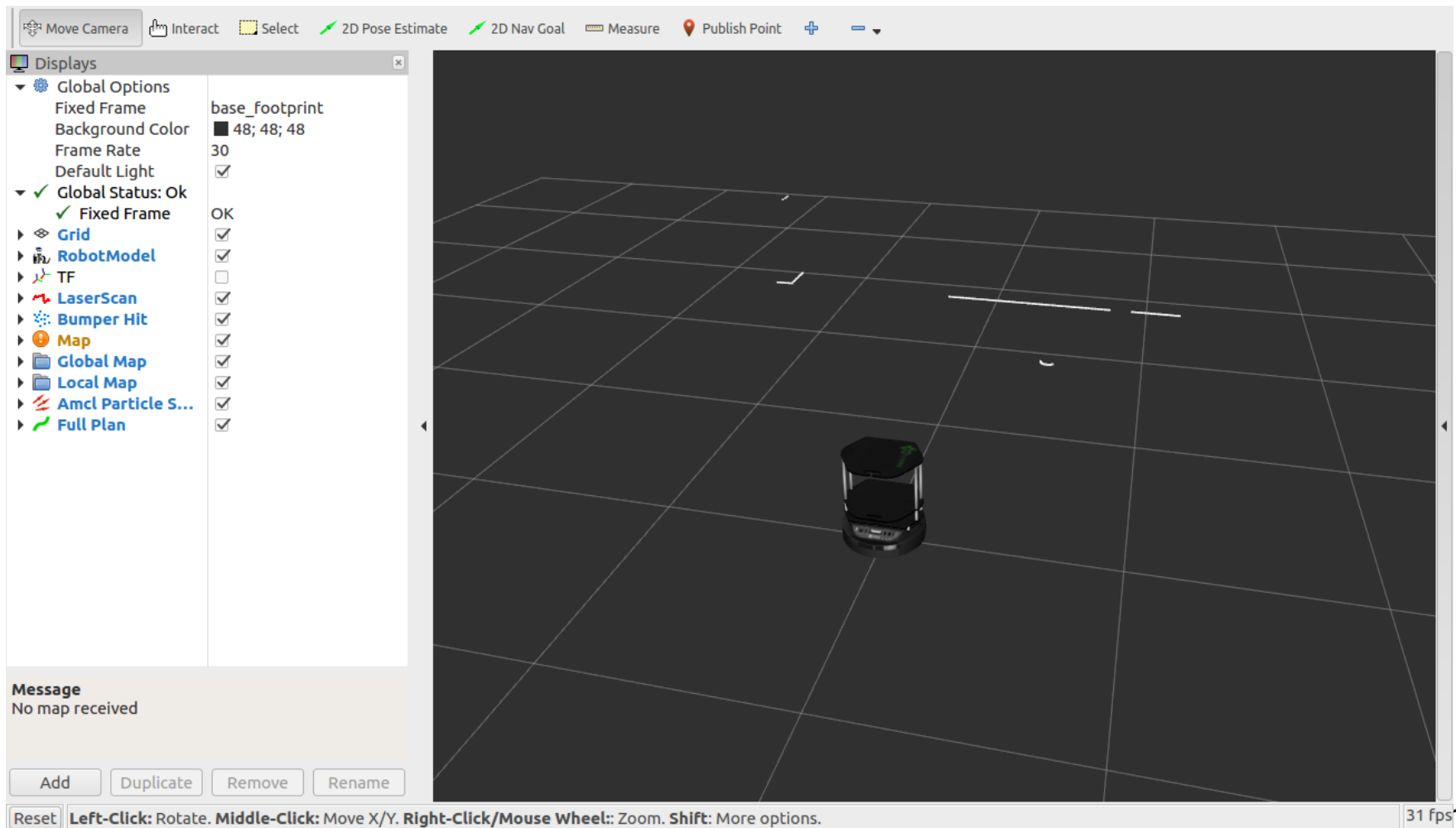
- **/scan**

- Escaneamento 2D dos arredores com o Kinect

MSG: sensor\_msgs/LaserScan



# Visão do Scan



# Objetivo

- **Criação de um node para fazer o robô evitar obstáculos a frente fazendo curvas de 90 graus para um sentido aleatório**
  - Fazer controle de ângulo
  - Utilizar o sensor de distância para tomada de decisão
  - Controlar velocidade em relação ao obstáculo a frente



# Simulink

- **Tools → Robot Operating System → Configure ROS Network Address**

**Configure ROS Network Addresses**

Configure ROS network addresses for simulation. These apply to Publish, Subscribe, Get Parameter, and Set Parameter blocks in all open Simulink models.

**ROS Master**

Network Address: Custom

Hostname/IP Address: 172.16.21.128

Port: 11311

**Node Host**

The address that external ROS nodes should use to connect to a Simulink node.

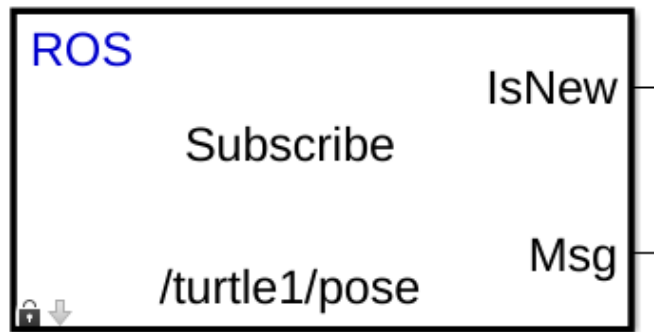
Network Address: Default

Simulink will use the ROS\_HOSTNAME or ROS\_IP environment variable, or local hostname

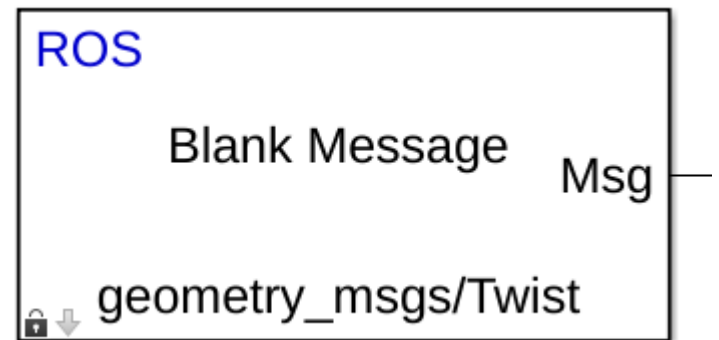


# Simulink - Elementos básicos

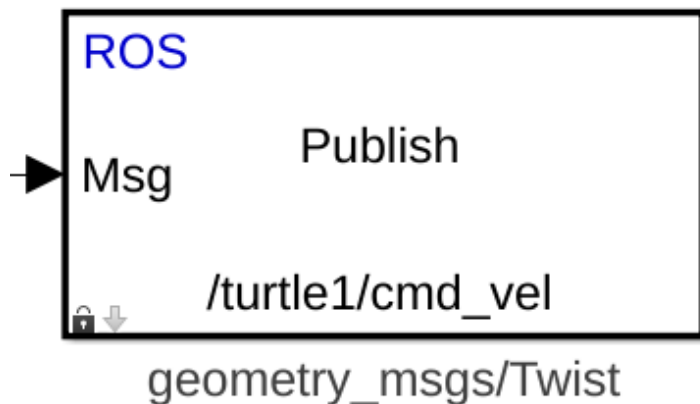
- **Módulo subscriber**



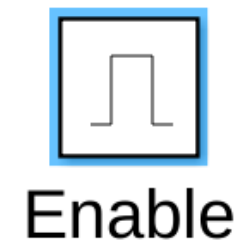
- **Módulo Blank message**



- **Módulo publisher**



- **Módulo EnablePort**

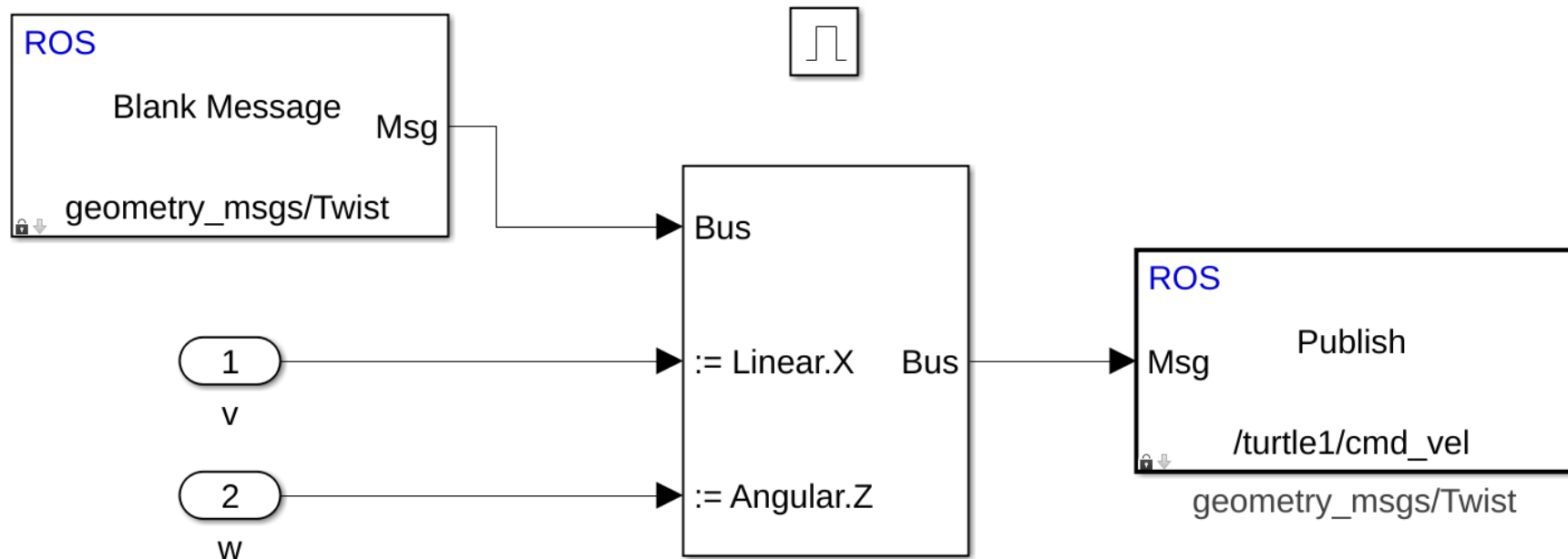




# Simulink - Publicando

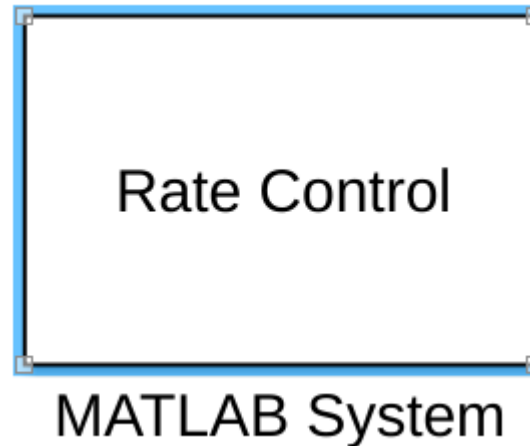
- **Utilizando Subsystem**

- EnablePort para execução a cada nova msg
- BusAssignment block para carregar a nova msg



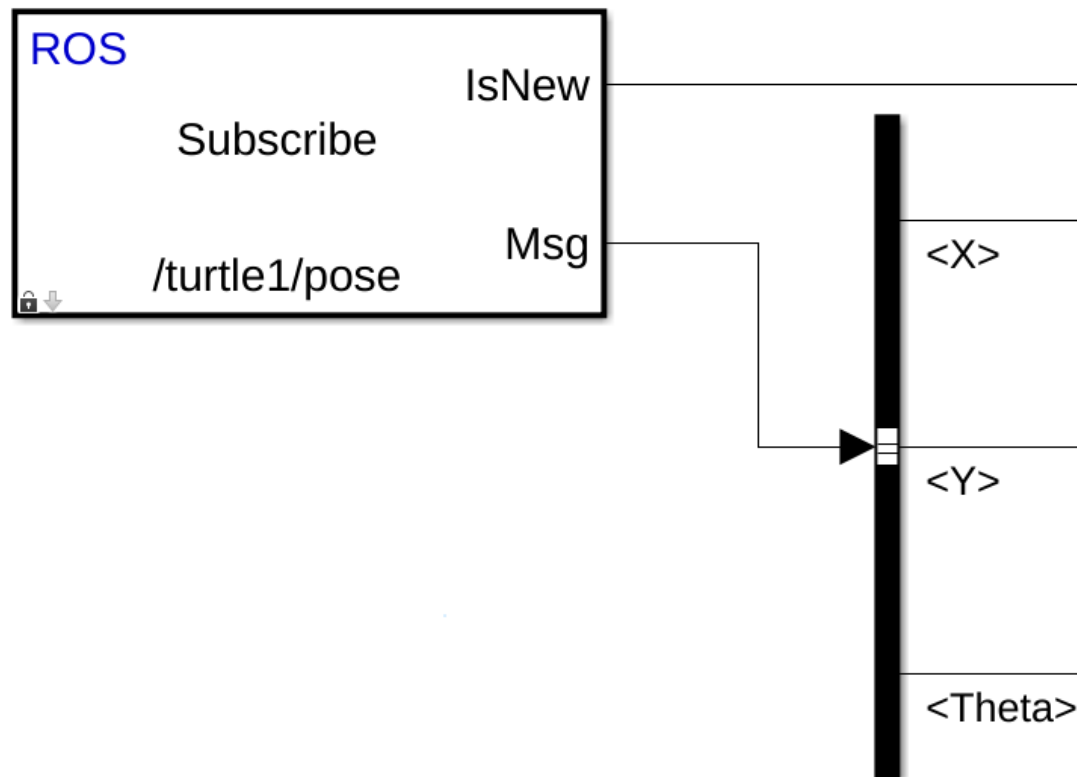
# Simulink

- **Controle de frequência de execução**
  - “Matlab System” tipo ExampleHelperSimulationRateControl



# Simulink - Subscrivendo

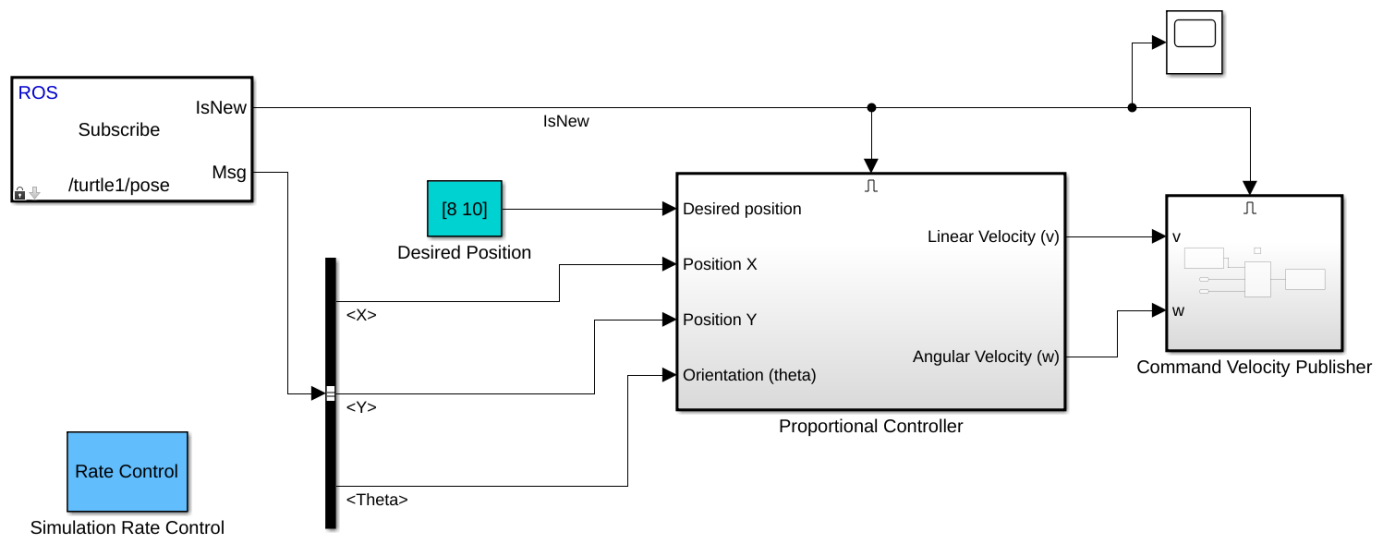
- **Módulo de subscribe**
  - BusSelector para separar a msg



# Simulink - Controle P turtlesim

- Simulink → turtleControl.slx

Feedback Control of a ROS-enabled Robot



# Simulink - Controle P turtlesim

- Simulink → turtleControl.slx

Feedback Control of a ROS-enabled Robot

