Department of Electrical and Electronic Engineering

ELEN90066 Embedded System Design

Workshop Six (W06) - (Remote)

Controlling a virtual robot

Welcome to Workshop 6 for Embedded System Design. In this workshop, you will be designing a Finite State Machine (FSM) modelled in Stateflow in MATLAB to control the robot Kobuki that is being simulated in the Gazebo environment. The goal of this workshop is to learn how to integrate Simulink, Stateflow, ROS and Gazebo in order to control the Kobuki robot to perform a task. This is the final structured workshop for the subject and as such, you will be expected to be comfortable using all of the software to develop your robot obstacle course algorithm as part of the project for the subject.

There are again two options for using the software in this workshop:

1. **Locally on your own computer**

   You will need to have the following software installed before the workshop class:

   - MATLAB R2020b;
   - Stateflow add-on for MATLAB;
   - ROS Toolbox add-on for MATLAB;
   - VirtualBox for your host computer operating system with:
     - Embedded System Design Virtual Machine (ESD VM)

2. **Remotely via MyUniApps**

   You will be using all applications on the remote machine (MATLAB, Gazebo, ROS). See the Workshop 3 notes for further details on using the ESD VM.

Pre-workshop background

In this workshop you will be reading the Kobuki bumper sensor data in order to control it. The Kobuki has three bumper sensors located on the front of the unit - one in the centre and one either side, left and right of centre. An event on any of the sensors generates a BumperEvent message in ROS. See here for further details on the BumperEvent Message definition for the Kobuki robot and what the `bumper` and `state` inputs will represent to the FSM that you will design.

Pre-workshop questions

1. What is the role of the `state` signal in the Kobuki BumperEvent message?
2. What is the role of the `bumper` signal in the Kobuki BumperEvent message? Could you use this signal to report multiple, simultaneous bumper events?

Workshop Exercises

In this workshop you will do the following

- Adapt your 'Random Walker' FSM from Workshop 5 to be compatible with Kobuki's sensors. (TASK 1 - 10 marks)
- Control the Kobuki robot to perform a random walk in a Gazebo world. (TASK 2 - 15 marks)

There are 25 marks available for successful completion of all tasks in this workshop, worth 1% of your final mark in the subject.

Modifying the Workshop 5 'Random Walker' Stateflow model

Your first task is to make the Workshop 5 'Random Walker' Stateflow model compatible with the Kobuki sensor data and motor commands.

1. Open up MATLAB either on your local machine if doing everything locally or via MyUniApps if using the EDS 12 Desktop. The machine that is running MATLAB will be referred to as the 'host' machine from now on.
2. Create a new blank Simulink model and save it in a known location where you normally store your files.
3. Open up the Simulink model you created for Task 3 in Workshop 5, copy the Stateflow FSM and paste it into your new Simulink model. If you didn't save your work from Workshop 5, you will have to quickly recreate the FSM.
4. Modify the FSM so that it now has two Simulink input ports `bumper` and `state` and two Simulink output ports `linear` and `angular`. You can delete the other outputs from Workshop 5 using the Model Explorer.

TASK 1 Change the FSM so that it implements the Random Walker algorithm as follows:

1. Drive forwards.
2. If a bump is detected:
   (a) Reverse for 1 second at a fixed linear speed of your choice.
   (b) Rotate a random amount.
3. Goto 1.

Your two outputs `linear` and `angular` now must be scalar numbers that represent the Linear.x and Angular.z components of a `/mobile_base/commands/velocity` topic in ROS. Recall Task 2 from Workshop 4 for a refresher on these parameters.

Provide suitable input signals to the `bumper` and `state` inputs and connect Simulink scopes to the outputs to show that your FSM is working according to specification.

Note : Your demonstrator will assess this task once you have finished it.

There are 10/25 marks for successful completion of this task.

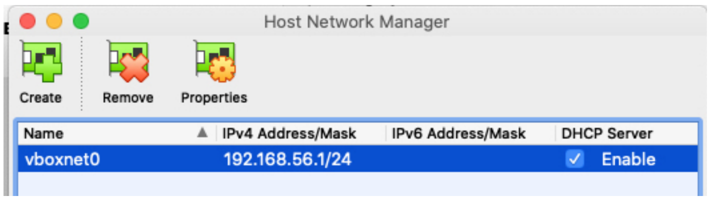
Connecting MATLAB to ROS and Gazebo

It's now time to test out your Random Walker algorithm on the real[1] thing! Firstly, the ROS/Gazebo environment and the MATLAB environment need to be setup so that they can communicate with one another.

- If you are running everything via MyUniApps, this has already been done for you.
- If you are running the Virtual Machine on your own computer you **MUST** follow the directions below in order for you to be able to connect ROS/Gazebo to MATLAB.

*****For those running the ESD VM on their own machine*****

1. Shutdown the ESD VM so it is in the 'Powered Off' state. If it is suspended you need to resume it and fully shut it down.
2. Go to the File→Host Network Manager menu item in VirtualBox.
3. Click on 'Create'.



Host Network Manager

Name	IPv4 Address/Mask	IPv6 Address/Mask	DHCP Server
vboxnet0	192.168.56.1/24		✓ Enable

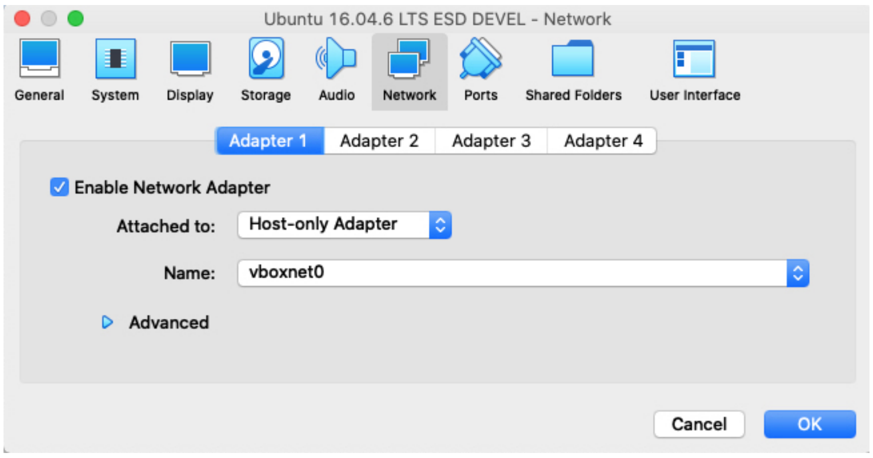
Figure 1: Host Network Manager in VirtualBox

4. By default it should create an entry ‘vboxnet0’ with an IPv4 Address/Mask of `192.168.56.1/24` as shown in Figure 1. If not, you will need to manually edit this by clicking on the ‘Properties’ button. Make sure that the DHCP Server is enabled by checking the Enable box.

5. Click ‘Close’.

6. Select the Virtual Machine and click on ‘Settings’.

7. Under the ‘Network’ tab, choose ‘Attached to:’ to be equal to ‘Host-only Adapter’ and Name as ‘vboxnet0’ as shown in Figure 5.



Ubuntu 16.04.6 LTS ESD DEVEL - Network

✓ Enable Network Adapter

Attached to: Host-only Adapter

Name: vboxnet0

Figure 2: Network Adapters in VirtualBox

*****END for those running the ESD VM on their own machine*****

In the ESD VM

1. Open up a NEW terminal window.

2. Enter the following to register the ROS environment variables with the system:

[1]Technically, ‘real virtual’.

```
$ export ROS_MASTER_URI=http://192.168.56.101:11311
$ export ROS_IP=192.168.56.101
```

3. Initiate a ROS master and load up a basic Gazebo world with a Kobuki spawned into it by typing

```
$ roslaunch kobuki_gazebo kobuki_empty_world.launch
```

On the Host (MATLAB) computer

1. Load MATLAB if it is not already running.
2. Find the IP address of the VirtualBox Host adapter in MATLAB:
   - on a Mac computer - enter the following at the MATLAB command prompt
     `>> system('ifconfig -a')`
   - on a Windows PC - enter the following at the MATLAB command prompt
     `>> system('ipconfig -all')`
3. Look for the `vboxnet0:` adapter in the output. It should have an IP address of `192.168.56.1`. If not, write down its address.
4. Now type the following two commands at the MATLAB command prompt to set the appropriate MATLAB environment variables (assuming the addresses are the default):

```
>> setenv('ROS_MASTER_URI','http://192.168.56.101:11311')
>> setenv('ROS_IP','192.168.56.1')
```

Summary of settings

With everything setup as above, you should have the configuration in Figure 3.

Now for the moment of truth... Verify that everything is working by typing `rosinit` to initialise a ROS master and then `rostopic list` in MATLAB. You should be able to see all of the ROS and Gazebo topics in MATLAB! If not, retrace the steps in this section to make sure everything is set correctly. You can shutdown the ROS master in MATLAB by using the `rosshutdown` command.

Using ROS in Simulink to communicate with the Kobuki

Now that everything is setup, you can start controlling the robot using MATLAB.

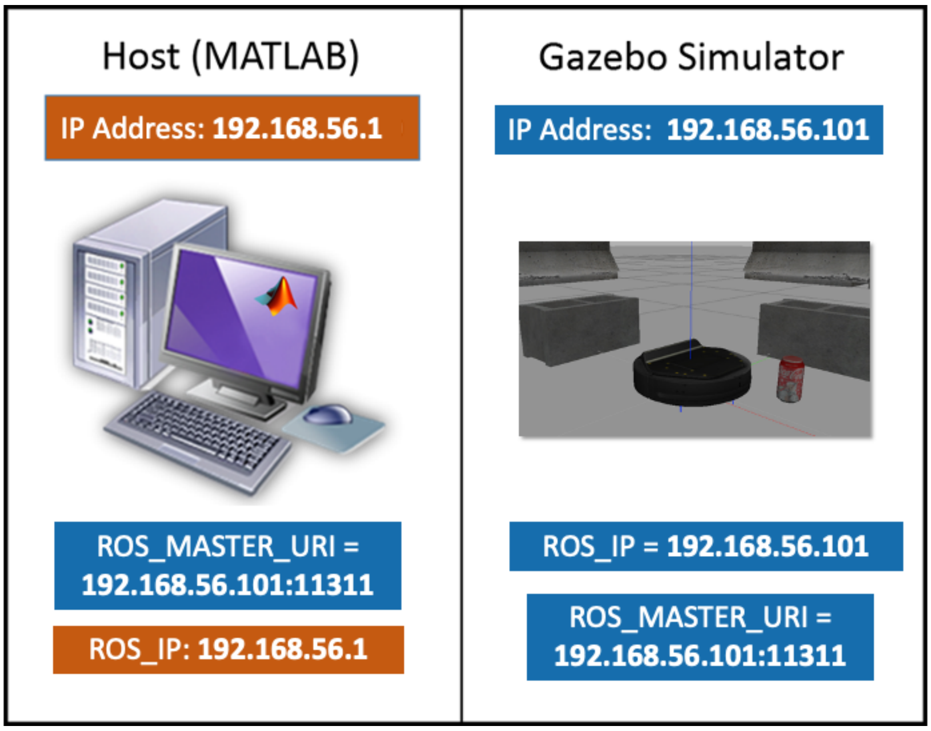

Host (MATLAB)
IP Address: 192.168.56.1
ROS_MASTER_URI = 192.168.56.101:11311
ROS_IP: 192.168.56.1
Gazebo Simulator
IP Address: 192.168.56.101
ROS_IP = 192.168.56.101
ROS_MASTER_URI = 192.168.56.101:11311

Figure 3: Summary of network setup

Send Velocity Commands To the Robot

In this section you will create a publisher that sends control commands (linear and angular velocities) to ROS/Gazebo. You can make these velocities adjustable by using Slider Gain blocks.

ROS uses a right-handed coordinate system, so the X-axis is forward, Y-axis is left, and Z-axis is up. Control commands are sent using a `geometry_msgs/Twist` message, where Linear.X indicates linear forward velocity (in m/s), and Angular.Z indicates angular velocity around the Z-axis (in rad/s).

Configure a Publisher Block

1. Open a new Simulink model.
2. From the **ROS Toolbox** → **ROS** tab in the Library Browser, drag a **Publish** block to the model. Double-click the block.
3. Set **Topic source** field to **Select From ROS network**. Click **Select** next to **Topic**, select `/mobile_base/commands/velocity`, and click **OK**. Note that the message type (`geometry_msgs/Twist`) is set automatically.

Configure a Message Block

1. From the **ROS Toolbox** → **ROS** tab in the Library Browser, drop a **Blank Message** block to the model. Double-click the block.

2. Click **Select** next to **Message type** and select `geometry_msgs/Twist`.

3. Set **Sample time** to `0.01` and click **OK**.

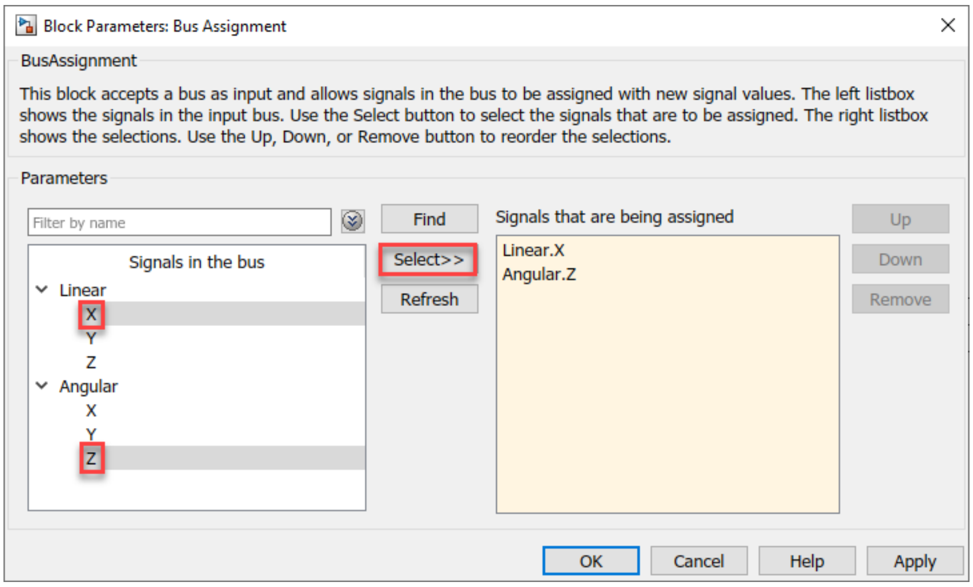
Configure Message Inputs

1. From the **Simulink** → **Signal Routing** tab in the Library Browser, drag a **Bus Assignment** block to the model.

2. Connect the `Msg` output of the **Blank Message** block to the Bus input of the **Bus Assignment** block, and the Bus output to the Msg input of the **Publish** block.

3. From the **Modeling** tab, click **Update Model** to ensure that the bus information is correctly propagated. Ignore the error, "Selected signal 'signal1' in the Bus Assignment block 'untitled/Bus Assignment' cannot be found in the input bus signal", if it appears. The next step will resolve this error.

4. Double-click the **Bus Assignment** block. Select `??? signal1` in the right list box and click **Remove**. In the left list box, expand both Linear and Angular properties. Select **Linear** → **X** and **Angular** → **Z** and click **Select** as shown in Figure 4. Click **OK** to close the block mask.



Block Parameters: Bus Assignment

BusAssignment

This block accepts a bus as input and allows signals in the bus to be assigned with new signal values. The left listbox shows the signals in the input bus. Use the Select button to select the signals that are to be assigned. The right listbox shows the selections. Use the Up, Down, or Remove button to reorder the selections.

Parameters

Signals that are being assigned: Linear.X, Angular.Z

Figure 4: Bus Assignments

5. Add a **Constant** block, a **Gain** block, and two **Slider Gain** blocks. Connect them together as shown in Figure 5, and set the **Gain** value to `-1`.

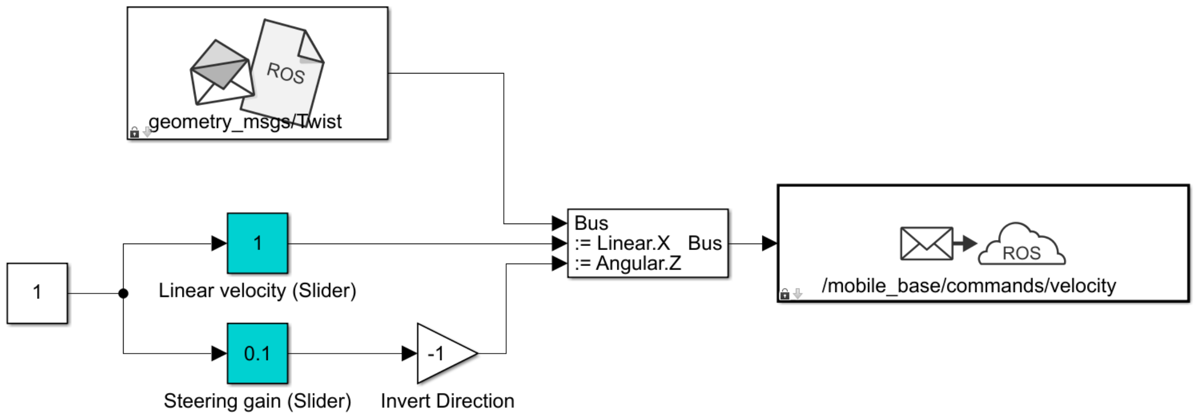


Figure 5: Simulink schematic

Set the limits and current parameters of the linear velocity slider to `0.0` to `2.0`, and `1.0` respectively. Set the corresponding parameters of the steering gain slider to `-1.0` to `1.0`, and `0.1`.

Receive Location Information from Gazebo

Gazebo and ROS information can be received by Simulink as long as it is published in ROS.

The code for the Kobuki model [2] publishes its position to the `/odom` topic in ROS. This data is obtained from wheel odometry and the yaw, taken directly from the built in IMU.

You will now create a subscriber to receive messages sent to the `/odom` topic in ROS, which will extract the location of the robot and plot it's path in the XY-plane.

Configure a Subscriber block

1. From the **ROS Toolbox** → **ROS** tab in the Library Browser, drag a **Subscribe** block to the model. Double-click the block.
2. Set **Topic source** to **Select From ROS network**, and click **Select** next to the **Topic** box. Select "/odom" for the topic and click **OK**. Note that the message type `nav_msgs/Odometry` is set automatically.

Read Message Output

1. From the **Simulink** → **Signal Routing** tab in the Library Browser, drag a **Bus Selector** block to the model.
2. Connect the output port of the **Subscribe** block to the input port of the **Bus Selector** block. In the **Modeling** tab, click **Update Model** to ensure that the bus information is correctly propagated.

[2] Available here for those who are super interested.

3. Double-click the **Bus Selector** block. Select `???  signal1` and `???  signal2` in the right listbox and click **Remove**. In the left listbox, expand **Pose** → **Pose** → **Position** and select **X** and **Y**. Click **Select** and then **OK**.

4. From the **Simulink** → **Sinks** tab in the Library Browser, drag an **XY Graph** block to the model. Connect the X and Y output ports of the **Bus Selector** block to the input ports of the **XY Graph** block.

Figure 6 shows the entire Simulink model.

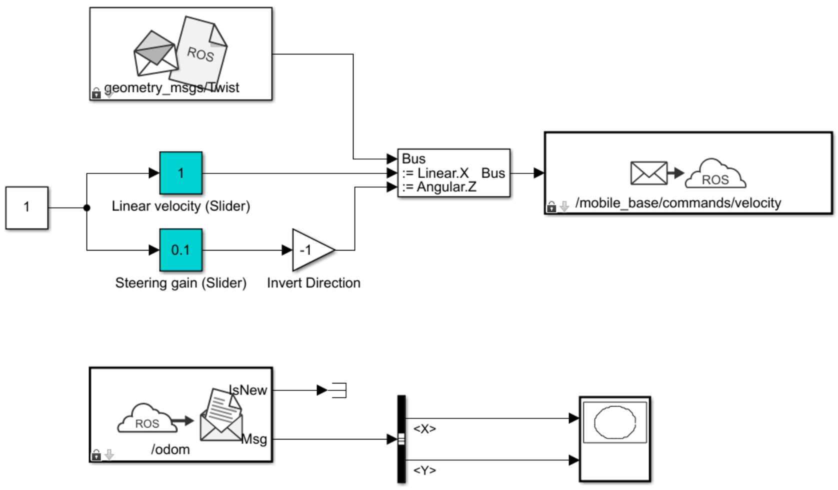


Figure 6: Simulink schematic

Configure and Run the Model

1. From the **Modeling** tab, click **Model Settings**. In the **Solver** pane, set **Type** to **Fixed-step** and **Fixed-step size** to `0.01`.

2. Set simulation Stop time to `Inf`.

3. From the **Run** button drop-down menu, click **Simulation Pacing** and set the **Simulation time per wall clock second** to `1`.

4. Click **Run** to start the simulation.

5. In both the simulator and XY plot, you should see the robot moving in a circle.

6. While the simulation is running, change the values of **Slider Gain** blocks to control the robot. Double-click the **XY Graph** block and change the X and Y axis limits if needed. (You can do this while the simulation is running.)

7. To stop the simulation, click **Stop**.

Once you have verified that you can control the robot in Gazebo, you can now move on to Task 2, where you will control the Kobuki with your Random Walker algorithm from Task 1.

TASK 2 For this task, you will control the Kobuki with your Random Walker FSM from Task 1 and test it in an appropriate world in Gazebo.

1. Remove the slider inputs that are connected to the Linear.X and Angular.Z velocity inputs of the Bus Selector block in Figure 6 and insert your Random Walker FSM from Task 1 into the schematic. Connect the outputs of your FSM to the appropriate Linear.X and Angular.Z bus inputs.
2. Add another Subscribe block to the model to receive bumper messages from the Kobuki and use a Bus Selector to select the appropriate signals from the Msg output to go to the `bumper` and `state` inputs of your Finite State Machine.
3. Create an arena for your Kobuki for testing in Gazebo. You might like to consider using your `launch` and `world` files that you created for Task 1 from Workshop 4.

Observe the behaviour of the robot in the Gazebo GUI and use Simulink scopes to track the robot's position over time to show that your FSM is working according to specification.

Note : Your demonstrator will assess this task once you have finished it.

There are 15/25 marks for successful completion of this task.