Autonomous Golf Cart: ROS

Manual

Florida Polytechnic University

Contents

1 Introduction to ROS

Robot Operating System (ROS) is a communication interface that sends and receives data between robotic system. ROS will be used in the golf cart for sending and receiving data. This section will cover how to use ROS with MATLAB. ROS Toolbox can be downloaded from the MATLAB add on.

Initialize the ROS Network: To initialize ROS use the `rosinit` command. `rosinit` creates the ROS master with default global node. `rosinit('IP address')` connects to the ROS master to the IP address. `rosshutdown` ends the ROS master and deletes the nodes.

```
>> rosinit
Initializing ROS master on http://DESKTOP-V2C8LI0:11311/.
Initializing global node /matlab_global_node_76389 with NodeURI http://DESKTOP-V2C8LI0:65028/
>> rosshutdown
Shutting down global node /matlab_global_node_76389 with NodeURI http://DESKTOP-V2C8LI0:65028/
Shutting down ROS master on http://DESKTOP-V2C8LI0:11311/.
```

Nodes & Topics: Nodes are used to exchange date by sending (publishing) and receiving (subscribing) message.. Topics carry the message and have a unique name e.g. `/example` and has the message type associated with the name `std_msgs/String`.

The following diagrams and list are exerts from the ROS Toolbox Reference and User's Guide Document from MathWorks and highlight the commonly used commands used for project. Additional help can be found by going to MathWorks website.

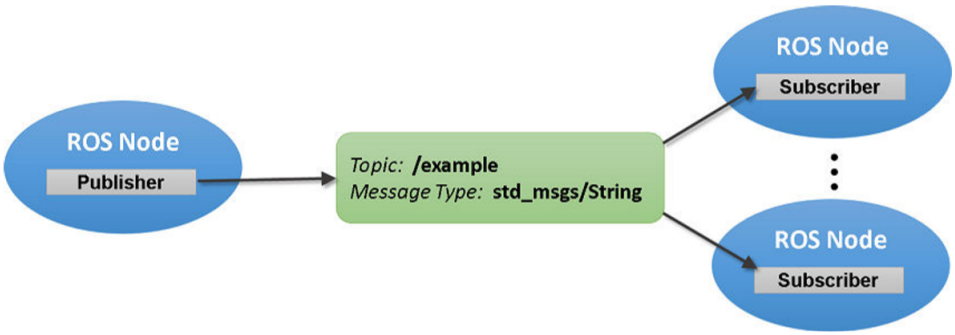


Figure 1: Example of node and topics

- `N = ros.Node('nodename')` - Creates multiple ROS nodes for use on the same ROS network in MATLAB.

- `pub = rospublisher('topicname','msgtype')` only works with the global node using rosinit. It does not require a node object handle as an argument.
- `pub = ros.Publisher(node,'topicname','msgtype')` creates a publisher for a topic with name, topicname. node is the robotics.ros.Node object handle that this publisher attaches to. If node is specified as [], the publisher tries to attach to the global node.
- `sub = rossubscriber('topicname','msgtype')` works only with the global node using rosinit. It does not require a node object handle as an argument.
- `sub = ros.Subscribe(node,'topicname','msgtype')` works with additional nodes that are created using ros.Node. It requires a node object handle as the first argument.
- `msg = rosmessage(pub)` creates an empty message determined by the topic published by pub. `msg.Data = pacevalue` specifies the `Data` of the message.
- `send(pub,msg)` publishes a message to the topic specified by the publisher, pub. This message can be received by all subscribers in the ROS network that are subscribed to the topic specified by pub.

2 Golf Cart Network

This section covers the step for creating the ROS network for the autonomous golf cart.

2.1 MATHLAB ROS Network:

The diagram in figure 2 is a visual representation of the MATHLAB code below. The ovals represent the nodes and the arrows represent the topics. If the arrow is leaving the node, the topic is being published and the the node receiving the arrow is subscribing to the topic. There are six nodes in the network. The `/AppNode` publishes three input and receives three feedback. The `/PedalNode`, `/SteeringNode`, and `/BrakingNode` publishes to either the `/ModelNode` or `/CartNode` and receives a feedback. The `/ModelNode` and `/CartNode` publishes addition data to the default global node.

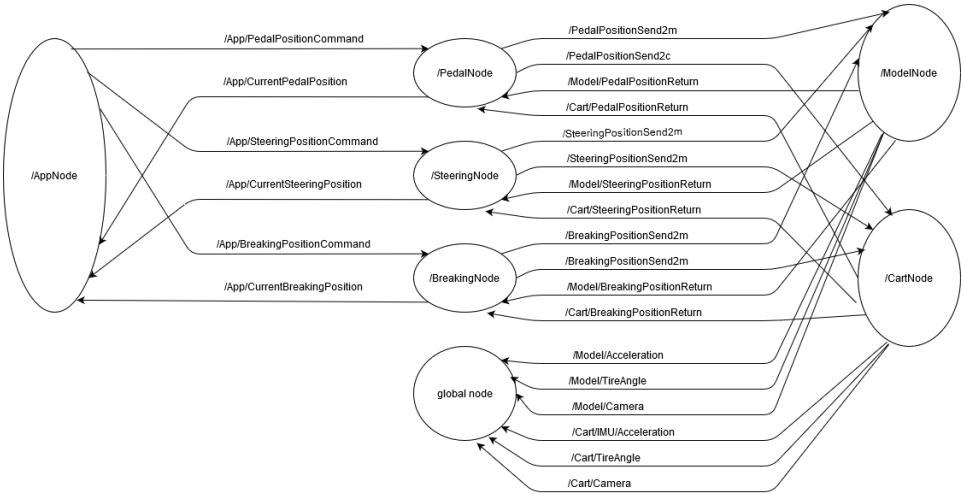


Figure 2: ROS network

MATHLAB code:

```
% Autonomous Golf Cart

rosshutdown
rosinit()

% creating all the nodes and topics
%% Nodes
node1 = ros.Node('/PedalNode');
node2 = ros.Node('/SteeringNode');
node3 = ros.Node('/BreakingNode');
node4 = ros.Node('/AppNode');
node5 = ros.Node('/ModelNode');
node6 = ros.Node('/CartNode');

%% Topics
%publisers
%app pub
pub1a = ros.Publisher(node4,'/App/
   PedalPositionCommand','geometry_msgs/Twist');
pub2a = ros.Publisher(node4,'/App/
   SteeringPositionCommand','geometry_msgs/Twist');
pub3a = ros.Publisher(node4,'/App/
   BreakingPositionCommand','geometry_msgs/Twist');
%pub to App
pub1p = ros.Publisher(node1,'/App/
   CurrentPedalPosition','geometry_msgs/Twist');
pub1s = ros.Publisher(node2,'/App/
   CurrentSteeringPosition','geometry_msgs/Twist');
pub1b = ros.Publisher(node3,'/App/
   CurrentBreakingPosition','geometry_msgs/Twist');
%pub to model
pub2pm = ros.Publisher(node1,'/PedalPositionSend2m',
   'geometry_msgs/Twist');
pub2sm = ros.Publisher(node2,'/
   SteeringPositionSend2m','geometry_msgs/Twist');
pub2bm = ros.Publisher(node3,'/
   BreakingPositionSend2m','geometry_msgs/Twist');
%pub to cart
pub2pc = ros.Publisher(node1,'/PedalPositionSend2c',
   'geometry_msgs/Twist');
pub2sc = ros.Publisher(node2,'/
```

```
   SteeringPositionSend2c','geometry_msgs/Twist');
pub2bc = ros.Publisher(node3,'/
   BreakingPositionSend2c','geometry_msgs/Twist');

%---Model pub---------
pub1m = ros.Publisher(node5,'/Model/
   PedalPositionReturn','geometry_msgs/Twist');
pub2m = ros.Publisher(node5,'/Model/
   SteeringPositionReturn','geometry_msgs/Twist');
pub3m = ros.Publisher(node5,'/Model/
   BreakingPositionReturn','geometry_msgs/Twist');
pub4m = ros.Publisher(node5,'/Model/Acceleration','
   geometry_msgs/Twist');
pub5m = ros.Publisher(node5,'/Model/TireAngle','
   geometry_msgs/Twist');
pub6m = ros.Publisher(node5,'/Model/Camera','
   geometry_msgs/Twist');
%---Cart pub----------
pub1c = ros.Publisher(node6,'/Cart/
   PedalPositionReturn','geometry_msgs/Twist');
pub2c = ros.Publisher(node6,'/Cart/
   SteeringPositionReturn','geometry_msgs/Twist');
pub3c = ros.Publisher(node6,'/Cart/
   BreakingPositionReturn','geometry_msgs/Twist');
pub4c = ros.Publisher(node6,'/Cart/IMU/Acceleration'
   ,'geometry_msgs/Twist');
pub5c = ros.Publisher(node6,'/Cart/TireAngle','
   geometry_msgs/Twist');
pub6c = ros.Publisher(node6,'/Cart/Camera','
   geometry_msgs/Twist');
%--------------------
%subscribers
%app sub
sub1a = ros.Subscriber(node4,'/App/
   CurrentPedalPosition','geometry_msgs/Twist');
sub2a = ros.Subscriber(node4,'/App/
   CurrentSteeringPosition','geometry_msgs/Twist');
sub3a = ros.Subscriber(node4,'/App/
   CurrentBreakingPosition','geometry_msgs/Twist');
%sub from App
sub1p = ros.Subscriber(node1,'/App/
   PedalPositionCommand','geometry_msgs/Twist');
sub1s = ros.Subscriber(node2,'/App/
   SteeringPositionCommand','geometry_msgs/Twist');
```

```
57 sub1b = ros.Subscriber(node3,'/App/
       BreakingPositionCommand','geometry_msgs/Twist');
58 %---sub from Model----
59 sub2p = ros.Subscriber(node1,'/Model/
       PedalPositionReturn','geometry_msgs/Twist');
60 sub2s = ros.Subscriber(node2,'/Model/
       SteeringPositionReturn','geometry_msgs/Twist');
61 sub2b = ros.Subscriber(node3,'/Model/
       BreakingPositionReturn','geometry_msgs/Twist');
62 %---sub from Cart-----
63 sub3p = ros.Subscriber(node1,'/Cart/
       PedalPositionReturn','geometry_msgs/Twist');
64 sub3s = ros.Subscriber(node2,'/Cart/
       SteeringPositionReturn','geometry_msgs/Twist');
65 sub3b = ros.Subscriber(node3,'/Cart/
       BreakingPositionReturn','geometry_msgs/Twist');
66 %--------------------------------------
67 %---Model sub---------
68 sub1m = ros.Subscriber(node5,'/PedalPositionSend2m',
       'geometry_msgs/Twist');
69 sub2m = ros.Subscriber(node5,'/
       SteeringPositionSend2m','geometry_msgs/Twist');
70 sub3m = ros.Subscriber(node5,'/
       BreakingPositionSend2m','geometry_msgs/Twist');
71 %---Cart sub----------
72 sub1c = ros.Subscriber(node6,'/PedalPositionSend2c',
       'geometry_msgs/Twist');
73 sub2c = ros.Subscriber(node6,'/
       SteeringPositionSend2c','geometry_msgs/Twist');
74 sub3c = ros.Subscriber(node6,'/
       BreakingPositionSend2c','geometry_msgs/Twist');
75 %--------------------------------------
```

2.2 Testing ROS

The following code below tests the ROS network. Before testing this code, run the ROS network code shown previously. Three ROS message are created for the three inputs and using `rosmessage(pub)` and the Data is specified as Linear.X. Linear.X is a data type of the message type 'geomerty_msgs/Twist'. The node receives the message from the input and sends it to the model node. The model node receives 20 seconds of data and plots it.

```
1 % testing
```

```
% input (app input example)
pedalpos = 12; %example
steeringpos = 23; %example
breakingpos = 34; %example

% Crate message from app and publish it
pub1amsg =rosmessage(pub1a);
pub1amsg.Linear.X = pedalpos;
send(pub1a,pub1amsg);
pub2amsg = rosmessage(pub2a);
pub2amsg.Linear.X = steeringpos;
send(pub2a,pub2amsg);
pub3amsg =rosmessage(pub3a);
pub3amsg.Linear.X = breakingpos;
send(pub3a,pub3amsg);

% get message from app
sub1pmsg = sub1p.LatestMessage;
sub1smsg = sub1s.LatestMessage;
sub1bmsg = sub1b.LatestMessage;

% set the message
pub2pmsg = sub1pmsg;
pub2smsg = sub1smsg;
pub2bmsg = sub1bmsg;

% send message to model
send(pub2pm,pub2pmsg);
send(pub2sm,pub2smsg);
send(pub2bm,pub2bmsg);

% publish to model
% start clock
tic;
i=1;
% set empty array
time=[];
p2model=[];
s2model=[];
b2model=[];
pfrommodel=[];
sfrommodel=[];
bfrommodel=[];

```

```
while(toc<20)
    % run 20 sec and model node receives the data
    p2model = sub1m.LatestMessage;
    p2ms(i) = p2model.Linear.X;
    s2model = sub2m.LatestMessage;
    s2ms(i) = s2model.Linear.X;
    b2model = sub3m.LatestMessage;
    b2ms(i) = b2model.Linear.X;

    time(i)=toc;
    i=i+1;
end
%plot the data the model recieve
figure
plot(p2ms)
figure
plot(s2ms)
figure
plot(b2ms)
```

2.3 Simulink ROS Network

The ROS Toolbox includes Simulink block that publishes and subscribes messages for designated topics. The top level of the Simulink is displayed in figure 3. This subsection will focus on the left side, creating the ROS network that sends an input through the network and onto the Simulink model of the golf cart. Within the the network subsection, figure 4 there are three subsections, the input, network, and to the model.

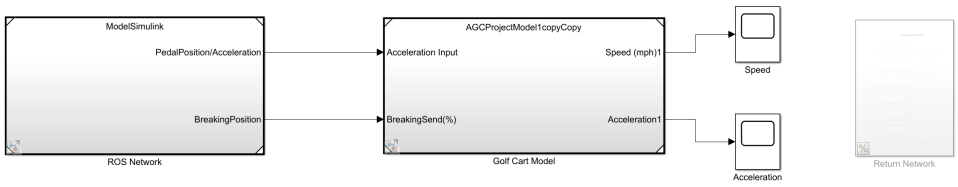


Figure 3: Simulink level 1

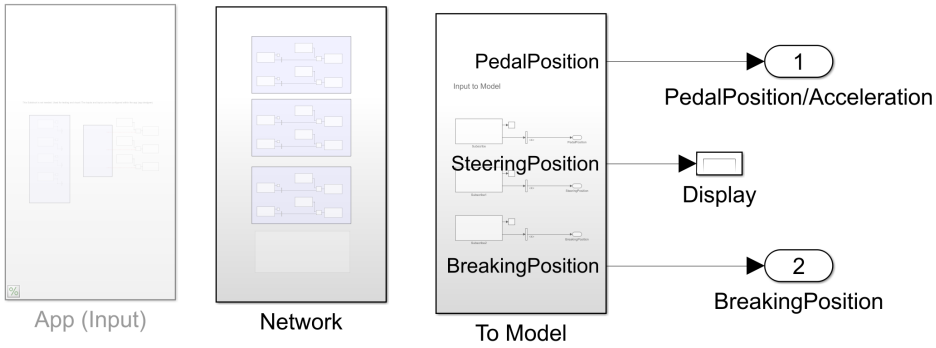


Figure 4: Simulink level 2 Ros Network

Figure 12 shows the input subsystem of the network. This subsection will be eventually replaced by the App. The input publishes the the three input signal for the pedal, steering, and braking. It is subscribed to the feedback signal of the model. Figure 13 shows the nodes of the network. In the subsystem there are six subscribe block. Three to get the message for the pedal, steering, and breaking data published from the App and similarly three for the feedback of the model. There are six publish block, three to send the message for the pedal, steering, and breaking data to the model and similarly three for the model to publish for feedback.

To create the the publisher place the Simulink block for the publisher, blank message, and the bus assignment as shown in figure 5. By first running the MATHLAB code of the golf cart ROS network from section 2.1, the topics would be loaded onto the work-space for accessibility. Then open the publish block and click select to choose the desired topic available as

in figure 6. Then open the blank message block and click select to choose the message type `geometry_msgs/Twist` as shown in figure 7. Then open the bus assignment block to select the specific data type of the message as shown in figure 8.

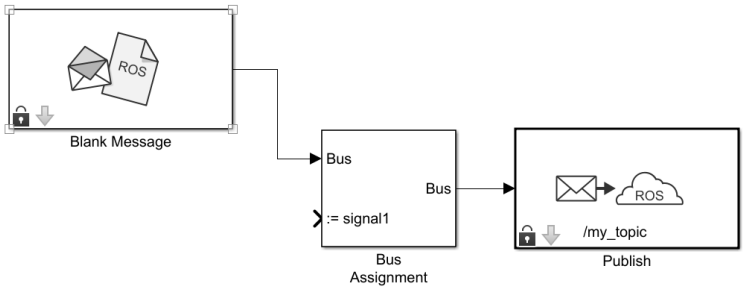


Figure 5: Publisher

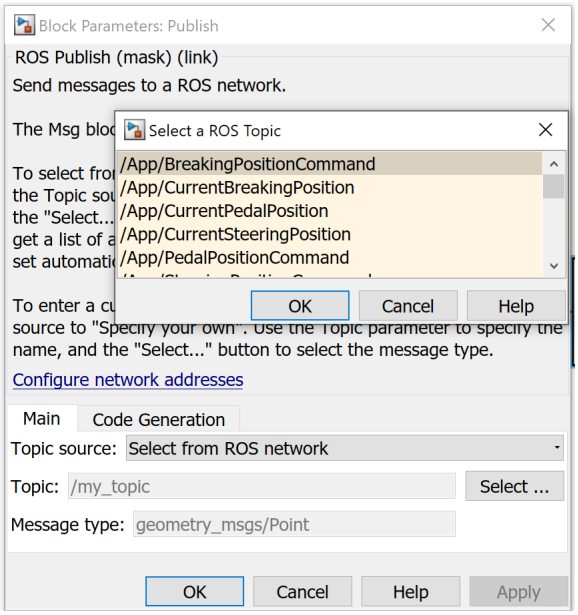


Figure 6: Publish block

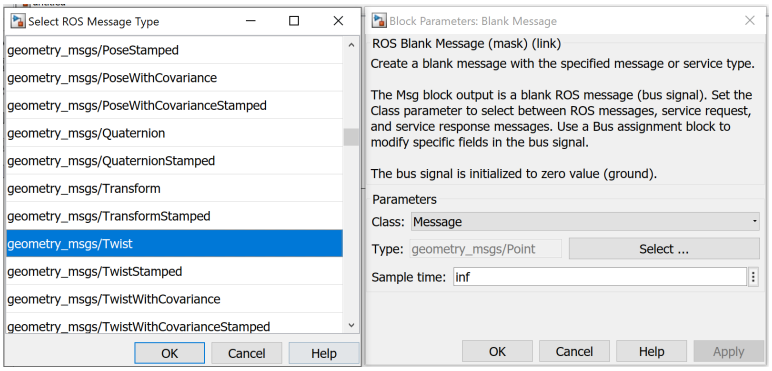


Figure 7: Blank message block

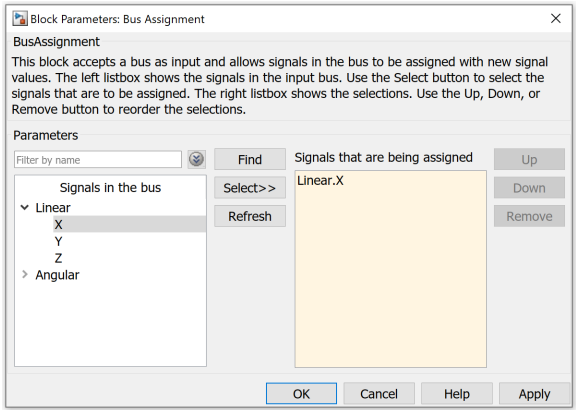


Figure 8: Bus assignment block

To create the subscriber place the subscribe block, connect the terminator block to the IsNew, and Bus selector block to the Msg. The IsNew output indicates whether the message has been received during the prior time step. For the current task, the IsNew output is not needed so it is connected the terminator. Open the subscribe block and click select to choose the desired topic available as in figure 10. Then open the bus assignment block to select the specific data type of the message as shown in figure 11.

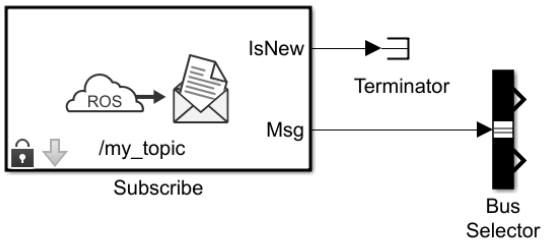


Figure 9: Subscriber

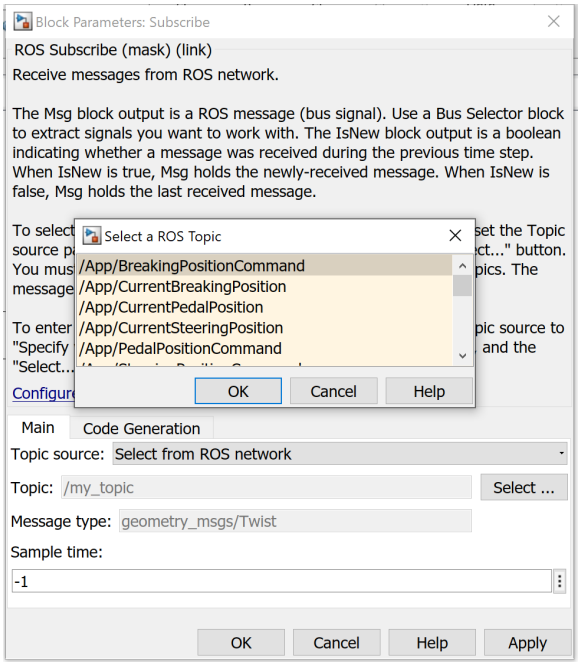


Figure 10: Subscribe block

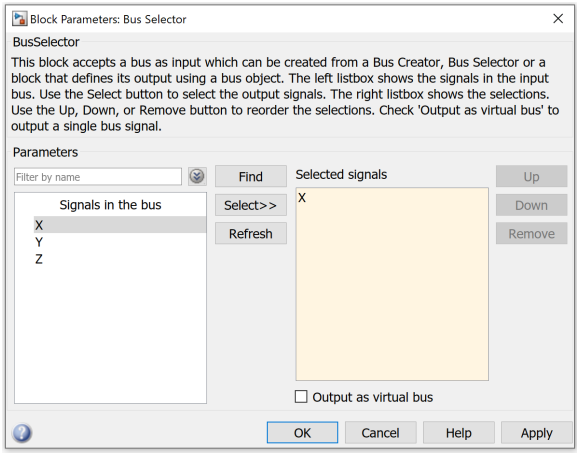


Figure 11: Bus assignment block

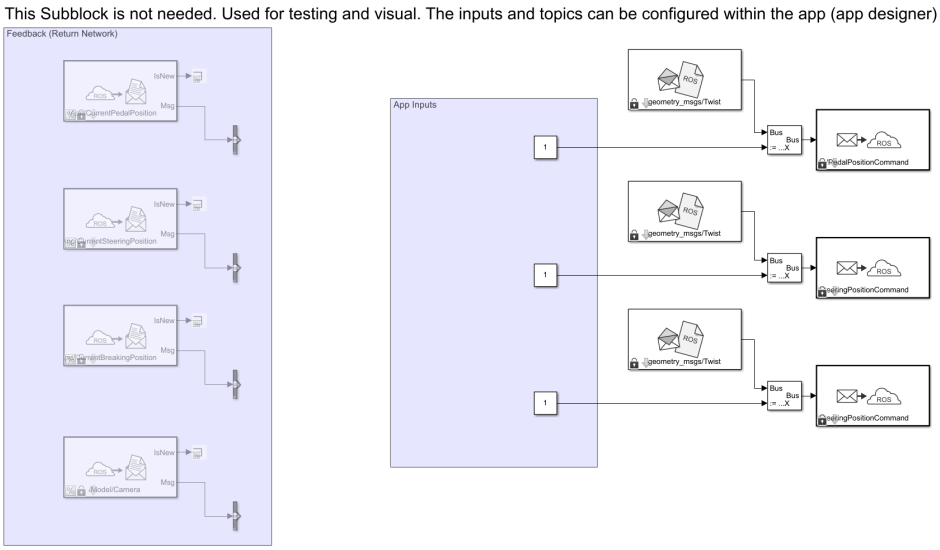


Figure 12: Simulink level 3 input subsystem

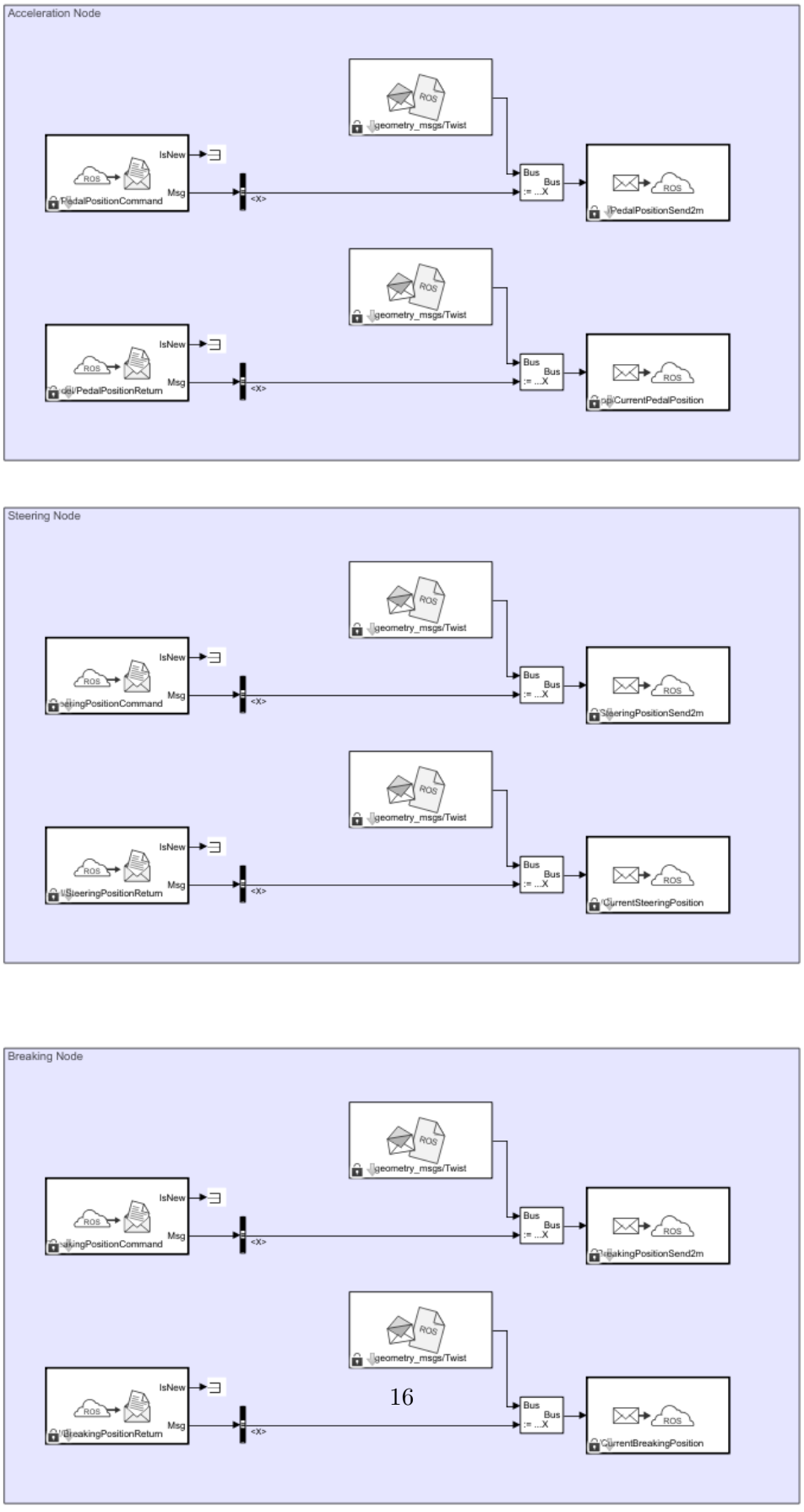


Figure 13: Simulink level 3 Network subsystem

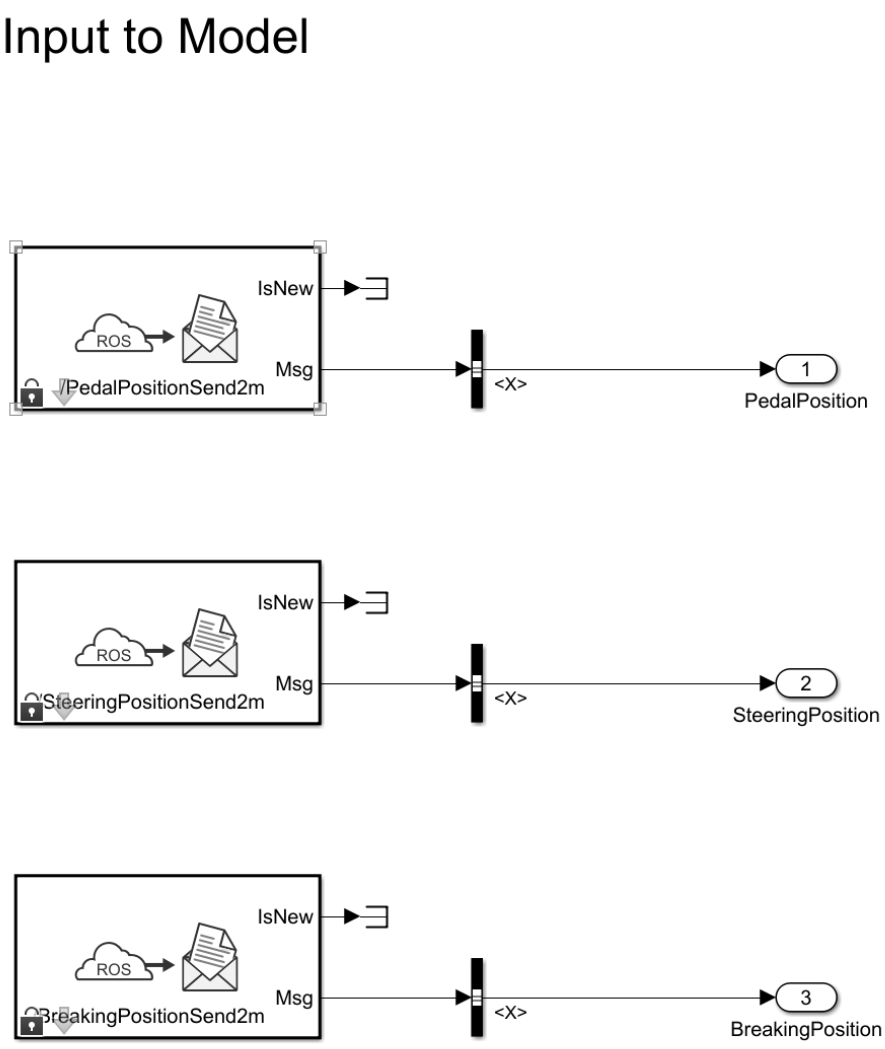


Figure 14: Simulink level 3 to model subsystem

2.4 App Design

This subsection covers the design of the App for controlling the input for the model. The App is developed with app designer from MATLAB by selecting Design App from the App tab or typing `appdesigner` in the command window. Figure 15 shows the design layout for the app. Create the app by dragging the sliders, buttons, text edit field, lamps, and axis from the app designer library onto the layout.

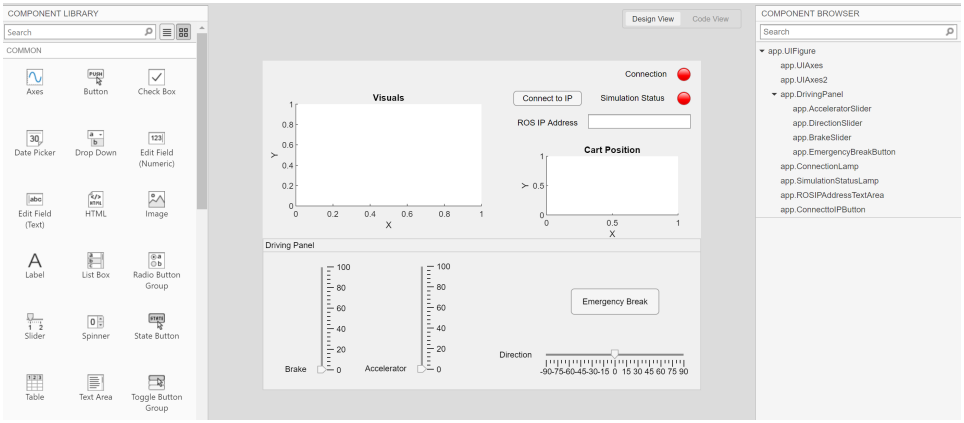


Figure 15: App layout

When the app layout is set, next switch to code view and right click each used components and select callbacks then choose to add one of the callbacks. For the sliders choose `'Add ValueChangingFcn callback'` shown in figure 16 bottom left. For the buttons and text edit field there is one callback, add that one. Within the code view, the code in gray area are automatically created by the app designer and the white spaces are to write the code for the callback. The code for the app is located in section 3.1. Add the properties and methods that would be used with the app. The `configureROSMsgs(app)` function creates the app node and topics for the app. The `ROSIPAddressTextAreaValueChanged(app , event)` function is the input where the IP address is entered for `rosinit`. The `ConnecttoIPButtonPushed(app , even)` is a push button and the ROS network is initialize when it is pressed. The `AcceleratorSliderValueChanging(app ,event)`, `BrakeSliderValueChanging(app , event)`, and `DirectionSliderValueChanging(app ,event)` publishes the message created from the slider. The `EmergencyBreakButtonPushed(app ,event)` publishes the the max breaking value and the min accelerating value when the buttong is pressed.

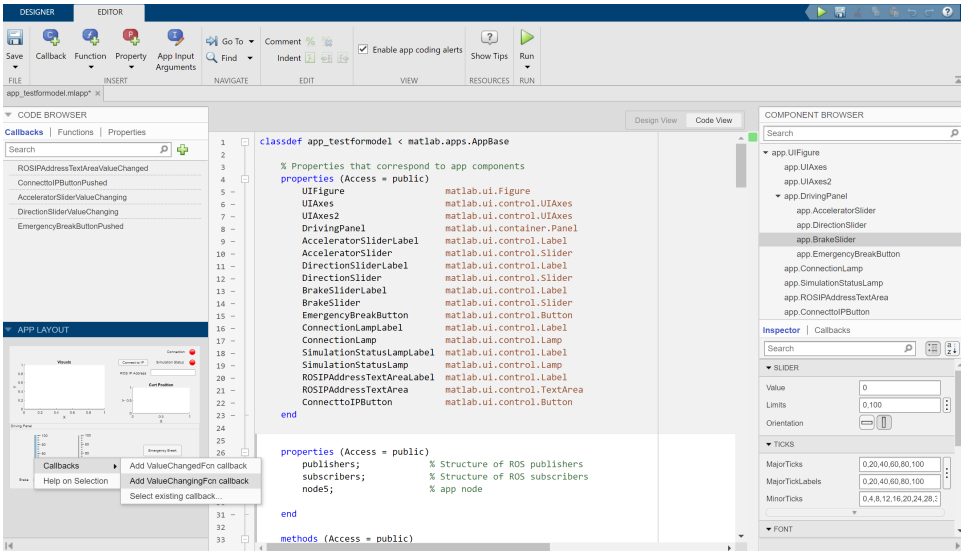


Figure 16: App add callback

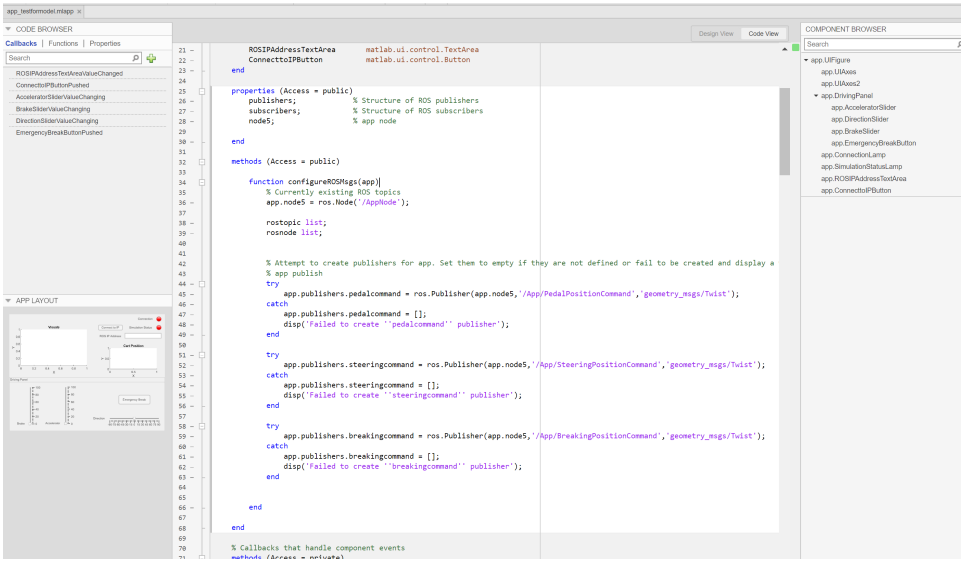


Figure 17: App code 1

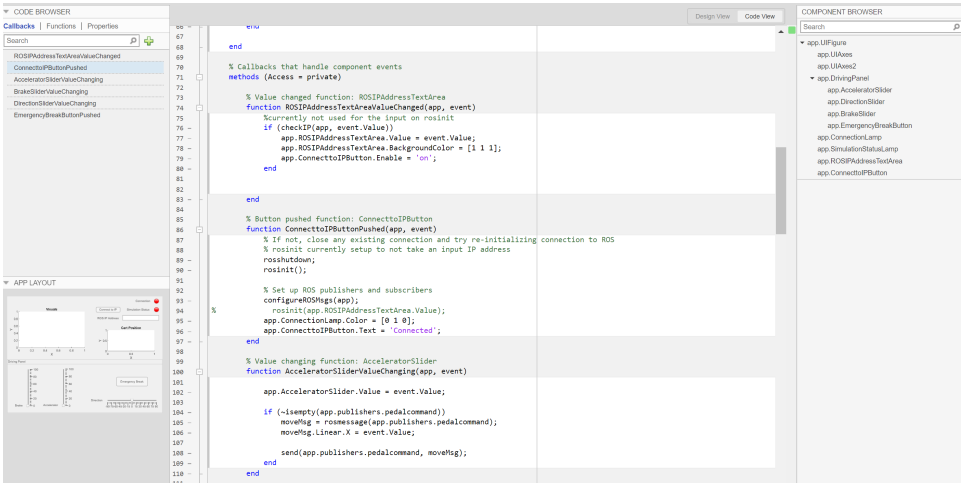


Figure 18: App code 2

2.5 Comments and proposals

This section will cover some comments, ideas and plans that have yet to be developed and should be reevaluated going forward.

- If there is an error when running the app, open it within app designer and run it.
- The current ROS network has 5 nodes and if an issue arises where the multiple nodes can't be loaded onto a raspberry pi, try combining some of the nodes.
- The current message type for all the topics are set to `geomety_msgs/Twist` and the specific data type is `Linear X`. If the data type needs to be change select the appropriate one from the message list and specify it on the publisher or subscriber. Type `rosmsg list` in the command for a list of message type. A custom message can also be created with MATLAB.
- `rosbag` is a file format for storing ROS message data. They are used primarily to log messages within the ROS network. You can use these bags for offline analysis, visualization, and storage. See the ROS Wiki page for more information about rosbags and how to create them. MATLAB does not create `rosbag` but it can open and manipulate them.
- The following page shows 6 different states of the network. The dash lines separate the hardware with the software. H is the hardware control, P is the plant of the golf cart, M is a simulation model, R is the remote or app, C is computer in autonomous mode, and the box with slash is the ROS network.
  - 1. The golf cart publishes data to the ROS network.
  - 2. The golf cart publishes data to the model for verification, to see if the model result are similar to the cart.
  - 3. The remote publishes commands to the golf cart and returns feedback
  - 4. The remote publishes to the simulation model and returns feedback.
  - 5. The autonomous mode is on, it publishes to the golf cart and returns feedback.
  - 6. The autonomous mode is on, it publishes to the simulated model and return feedback.

Network States

Tuesday, March 24, 2020 1:04 PM

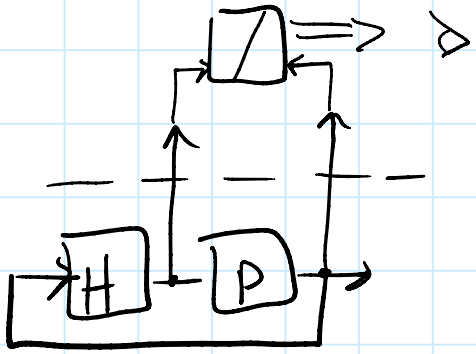
Driver \ Message Path	Cart	Model
Cart	Data Sensing	Verification
Remote	Drive by wire	Drive Sim.
Computer	Auto Mode	Auto-Test

Software

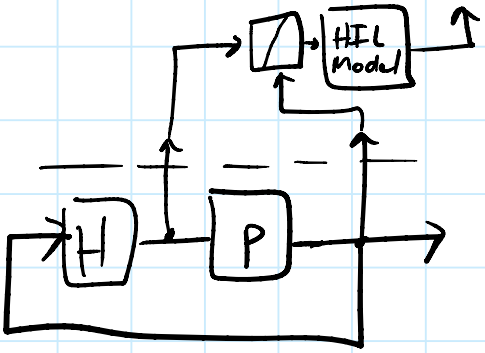
Hardware

Comm Net

1) Data sensing - Driver in Cart // Data to/from Cart

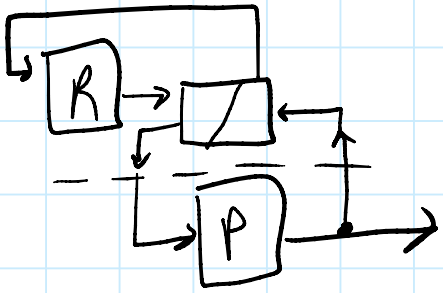


2) Verification - Driver in Cart // Data to/from Model

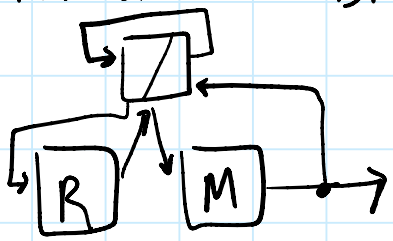


3) Drive-by-wire - Driver Remote // Data to/from Cart

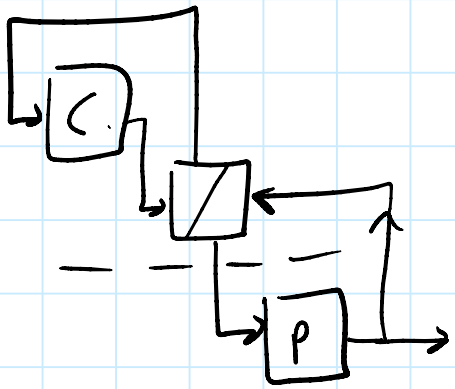
R - remote Setup



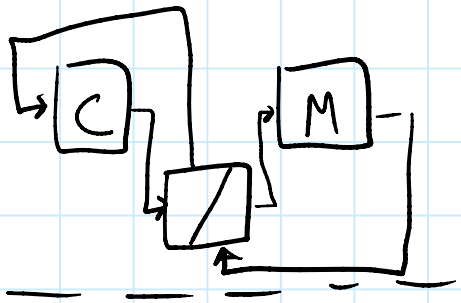
4) Drive Sim - Driver Remote // Data To/from Model



5) Auto mode - Comp Driver // Data to/from Cart



6) Auto test - Comp Driver // Data to/from Model



3 MATHLAB Code

3.1 App Code

```
classdef app_testformodel_exported < matlab.apps.
   AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        UIAxes                      matlab.ui.control
           .UIAxes
        UIAxes2                     matlab.ui.control
           .UIAxes
        DrivingPanel                matlab.ui.
           container.Panel
        AcceleratorSliderLabel      matlab.ui.control
           .Label
        AcceleratorSlider           matlab.ui.control
           .Slider
        DirectionSliderLabel        matlab.ui.control
           .Label
        DirectionSlider             matlab.ui.control
           .Slider
        BrakeSliderLabel            matlab.ui.control
           .Label
        BrakeSlider                 matlab.ui.control
           .Slider
        EmergencyBreakButton        matlab.ui.control
           .Button
        ConnectionLampLabel         matlab.ui.control
           .Label
        ConnectionLamp              matlab.ui.control
           .Lamp
        SimulationStatusLampLabel   matlab.ui.control
           .Label
        SimulationStatusLamp        matlab.ui.control
           .Lamp
        ROSIPAddressTextAreaLabel   matlab.ui.control
           .Label
        ROSIPAddressTextArea        matlab.ui.control
           .TextArea
        ConnecttoIPButton           matlab.ui.control
           .Button
```

```
23      end
24
25      properties (Access = public)
26          publishers;              % Structure of ROS
               publishers
27          subscribers;             % Structure of ROS
               subscribers
28          node5;                   % app node
29      end
30
31      methods (Access = public)
32
33          function configureROSMsgs(app)
34              % Currently existing ROS topics
35              app.node5 = ros.Node('/AppNode');
36
37              % Attempt to create publishers for app.
                   Set them to empty if they are not
                   defined or fail to be created
38              % app publish
39              try
40                  app.publishers.pedalcommand = ros.
                       Publisher(app.node5,'/App/
                       PedalPositionCommand','
                       geometry_msgs/Twist');
41              catch
42                  app.publishers.pedalcommand = [];
43              end
44
45              try
46                  app.publishers.steeringcommand = ros
                       .Publisher(app.node5,'/App/
                       SteeringPositionCommand','
                       geometry_msgs/Twist');
47              catch
48                  app.publishers.steeringcommand = [];
49              end
50
51              try
52                  app.publishers.breakingcommand = ros
                       .Publisher(app.node5,'/App/
                       BreakingPositionCommand','
                       geometry_msgs/Twist');
53              catch
```

```
54                  app.publishers.breakingcommand = [];
55              end
56
57              rostopic list;
58              rosnode list;
59
60          end
61      end
62
63      % Callbacks that handle component events
64      methods (Access = private)
65
66          % Value changed function:
               ROSIPAddressTextArea
67          function ROSIPAddressTextAreaValueChanged(
               app, event)
68              %currently not used for the input on
                   rosinit
69
70              app.ROSIPAddressTextArea.Value = event.
                   Value;
71              app.ROSIPAddressTextArea.BackgroundColor
                    = [1 1 1];
72              app.ConnecttoIPButton.Enable = 'on';
73          end
74
75          % Button pushed function: ConnecttoIPButton
76          function ConnecttoIPButtonPushed(app, event)
77              % rosinit currently setup to not take an
                    input IP address
78              rosshutdown;
79              rosinit(); % leave blank for local or
                   input ip address if app.
                   ROSIPAddressTextArea.Value is not
                   functioning
80  %             rosinit(app.ROSIPAddressTextArea.Value
    ) % displays an error
81
82              % Set up ROS publishers and subscribers
83              configureROSMsgs(app);
84
85              app.ConnectionLamp.Color = [0 1 0];
86              app.ConnecttoIPButton.Text = 'Connected'
                   ;
```

```
87          end
88
89          % Value changing function: AcceleratorSlider
90          function AcceleratorSliderValueChanging(app,
                event)
91
92              app.AcceleratorSlider.Value = event.
                    Value;
93
94              if (~isempty(app.publishers.pedalcommand
                    ))
95                  moveMsg = rosmessage(app.publishers.
                        pedalcommand);
96                  moveMsg.Linear.X = event.Value;
97
98                  send(app.publishers.pedalcommand,
                        moveMsg);
99              end
100         end
101
102         % Value changing function: BrakeSlider
103         function BrakeSliderValueChanging(app, event
                )
104
105             app.BrakeSlider.Value = event.Value;
106
107             if (~isempty(app.publishers.
                    breakingcommand))
108                 moveMsg = rosmessage(app.publishers.
                        breakingcommand);
109                 moveMsg.Linear.X = event.Value;
110
111                 send(app.publishers.breakingcommand,
                        moveMsg);
112             end
113         end
114
115         % Value changing function: DirectionSlider
116         function DirectionSliderValueChanging(app,
                event)
117
118             app.DirectionSlider.Value = event.Value;
119
```

```
120             if (~isempty(app.publishers.
                    steeringcommand))
121                 moveMsg = rosmessage(app.publishers.
                        steeringcommand);
122                 moveMsg.Linear.X = event.Value;
123
124                 send(app.publishers.steeringcommand,
                        moveMsg);
125             end
126         end
127
128         % Button pushed function:
                EmergencyBreakButton
129         function EmergencyBreakButtonPushed(app,
                event)
130             % when pressed changes breaking and
                    accelaring slider
131
132             app.BrakeSlider.Value = 100;
133             app.AcceleratorSlider.Value = 0;
134             % publish to breaking
135             moveMsg1 = rosmessage(app.publishers.
                    breakingcommand);
136             moveMsg1.Linear.X = app.BrakeSlider.
                    Value;
137             send(app.publishers.breakingcommand,
                    moveMsg1);
138             % publish to accelerator
139             moveMsg2 = rosmessage(app.publishers.
                    pedalcommand);
140             moveMsg2.Linear.X = app.
                    AcceleratorSlider.Value;
141             send(app.publishers.pedalcommand,
                    moveMsg2);
142         end
143     end
144
145     % Component initialization
146     methods (Access = private)
147
148         % Create UIFigure and components
149         function createComponents(app)
150
```

```
151             % Create UIFigure and hide until all
                    components are created
152             app.UIFigure = uifigure('Visible', 'off'
                    );
153             app.UIFigure.Position = [100 100 640
                    480];
154             app.UIFigure.Name = 'UI Figure';
155
156             % Create UIAxes
157             app.UIAxes = uiaxes(app.UIFigure);
158             title(app.UIAxes, 'Visuals')
159             xlabel(app.UIAxes, 'X')
160             ylabel(app.UIAxes, 'Y')
161             app.UIAxes.PlotBoxAspectRatio =
                    [1.8255033557047 1 1];
162             app.UIAxes.Position = [10 234 319 204];
163
164             % Create UIAxes2
165             app.UIAxes2 = uiaxes(app.UIFigure);
166             title(app.UIAxes2, 'Cart Position')
167             xlabel(app.UIAxes2, 'X')
168             ylabel(app.UIAxes2, 'Y')
169             app.UIAxes2.PlotBoxAspectRatio =
                    [2.24418604651163 1 1];
170             app.UIAxes2.Position = [376 221 240
                    141];
171
172             % Create DrivingPanel
173             app.DrivingPanel = uipanel(app.UIFigure)
                    ;
174             app.DrivingPanel.Title = 'Driving Panel'
                    ;
175             app.DrivingPanel.Position = [1 1 640
                    221];
176
177             % Create AcceleratorSliderLabel
178             app.AcceleratorSliderLabel = uilabel(app
                    .DrivingPanel);
179             app.AcceleratorSliderLabel.
                    HorizontalAlignment = 'right';
180             app.AcceleratorSliderLabel.Position =
                    [144 21 66 22];
181             app.AcceleratorSliderLabel.Text = '
                    Accelerator';
```

```
182
183             % Create AcceleratorSlider
184             app.AcceleratorSlider = uislider(app.
                    DrivingPanel);
185             app.AcceleratorSlider.Orientation = '
                    vertical';
186             app.AcceleratorSlider.ValueChangingFcn =
                     createCallbackFcn(app, @
                    AcceleratorSliderValueChanging, true)
                    ;
187             app.AcceleratorSlider.Position = [231 30
                     3 150];
188
189             % Create DirectionSliderLabel
190             app.DirectionSliderLabel = uilabel(app.
                    DrivingPanel);
191             app.DirectionSliderLabel.
                    HorizontalAlignment = 'right';
192             app.DirectionSliderLabel.Position = [340
                     41 53 22];
193             app.DirectionSliderLabel.Text = '
                    Direction';
194
195             % Create DirectionSlider
196             app.DirectionSlider = uislider(app.
                    DrivingPanel);
197             app.DirectionSlider.Limits = [-90 90];
198             app.DirectionSlider.MajorTicks = [-90
                    -75 -60 -45 -30 -15 0 15 30 45 60 75
                    90];
199             app.DirectionSlider.ValueChangingFcn =
                    createCallbackFcn(app, @
                    DirectionSliderValueChanging, true);
200             app.DirectionSlider.Position = [414 50
                    201 3];
201
202             % Create BrakeSliderLabel
203             app.BrakeSliderLabel = uilabel(app.
                    DrivingPanel);
204             app.BrakeSliderLabel.HorizontalAlignment
                     = 'right';
205             app.BrakeSliderLabel.Position = [27 20
                    37 22];
206             app.BrakeSliderLabel.Text = 'Brake';
```

```
207
208             % Create BrakeSlider
209             app.BrakeSlider = uislider(app.
                    DrivingPanel);
210             app.BrakeSlider.Orientation = 'vertical'
                    ;
211             app.BrakeSlider.ValueChangingFcn =
                    createCallbackFcn(app, @
                    BrakeSliderValueChanging, true);
212             app.BrakeSlider.Position = [85 29 3
                    150];
213
214             % Create EmergencyBreakButton
215             app.EmergencyBreakButton = uibutton(app.
                    DrivingPanel, 'push');
216             app.EmergencyBreakButton.ButtonPushedFcn
                     = createCallbackFcn(app, @
                    EmergencyBreakButtonPushed, true);
217             app.EmergencyBreakButton.Position = [450
                     110 129 38];
218             app.EmergencyBreakButton.Text = '
                    Emergency Break';
219
220             % Create ConnectionLampLabel
221             app.ConnectionLampLabel = uilabel(app.
                    UIFigure);
222             app.ConnectionLampLabel.
                    HorizontalAlignment = 'right';
223             app.ConnectionLampLabel.Position = [525
                    451 66 22];
224             app.ConnectionLampLabel.Text = '
                    Connection';
225
226             % Create ConnectionLamp
227             app.ConnectionLamp = uilamp(app.UIFigure
                    );
228             app.ConnectionLamp.Position = [606 451
                    20 20];
229             app.ConnectionLamp.Color = [1 0 0];
230
231             % Create SimulationStatusLampLabel
232             app.SimulationStatusLampLabel = uilabel(
                    app.UIFigure);
```

```
233             app.SimulationStatusLampLabel.
                    HorizontalAlignment = 'right';
234             app.SimulationStatusLampLabel.Position =
                    [489 416 102 22];
235             app.SimulationStatusLampLabel.Text = '
                    Simulation Status ';
236
237             % Create SimulationStatusLamp
238             app.SimulationStatusLamp = uilamp(app.
                    UIFigure);
239             app.SimulationStatusLamp.Position = [606
                    416 20 20];
240             app.SimulationStatusLamp.Color = [1 0
                    0];
241
242             % Create ROSIPAddressTextAreaLabel
243             app.ROSIPAddressTextAreaLabel = uilabel(
                    app.UIFigure);
244             app.ROSIPAddressTextAreaLabel.
                    HorizontalAlignment = 'right';
245             app.ROSIPAddressTextAreaLabel.Position =
                    [367 380 94 22];
246             app.ROSIPAddressTextAreaLabel.Text = '
                    ROS IP Address';
247
248             % Create ROSIPAddressTextArea
249             app.ROSIPAddressTextArea = uitextarea(
                    app.UIFigure);
250             app.ROSIPAddressTextArea.ValueChangedFcn
                    = createCallbackFcn(app, @
                    ROSIPAddressTextAreaValueChanged,
                    true);
251             app.ROSIPAddressTextArea.Position = [476
                    382 150 21];
252
253             % Create ConnecttoIPButton
254             app.ConnecttoIPButton = uibutton(app.
                    UIFigure, 'push');
255             app.ConnecttoIPButton.ButtonPushedFcn =
                    createCallbackFcn(app, @
                    ConnecttoIPButtonPushed, true);
256             app.ConnecttoIPButton.Position = [367
                    415 100 22];
```

```
257             app.ConnecttoIPButton.Text = 'Connect to
                    IP';
258
259             % Show the figure after all components
                   are created
260             app.UIFigure.Visible = 'on';
261         end
262     end
263
264     % App creation and deletion
265     methods (Access = public)
266
267         % Construct app
268         function app = app_testformodel_exported
269
270             % Create UIFigure and components
271             createComponents(app)
272
273             % Register the app with App Designer
274             registerApp(app, app.UIFigure)
275
276             if nargout == 0
277                 clear app
278             end
279         end
280
281         % Code that executes before app deletion
282         function delete(app)
283
284             % Delete UIFigure when app is deleted
285             delete(app.UIFigure)
286         end
287     end
288 end
```