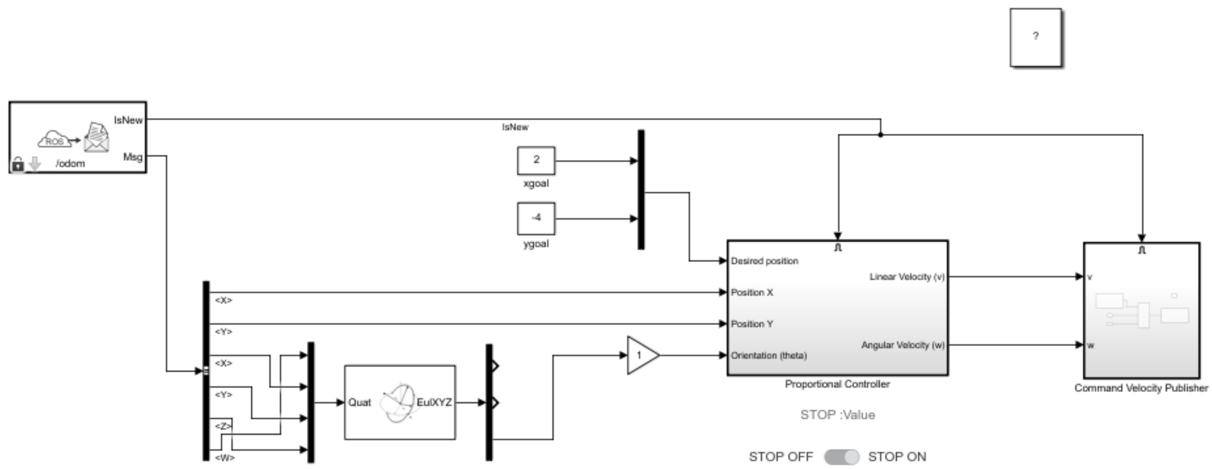
Lab 2: Mobile Robot Motion and Control

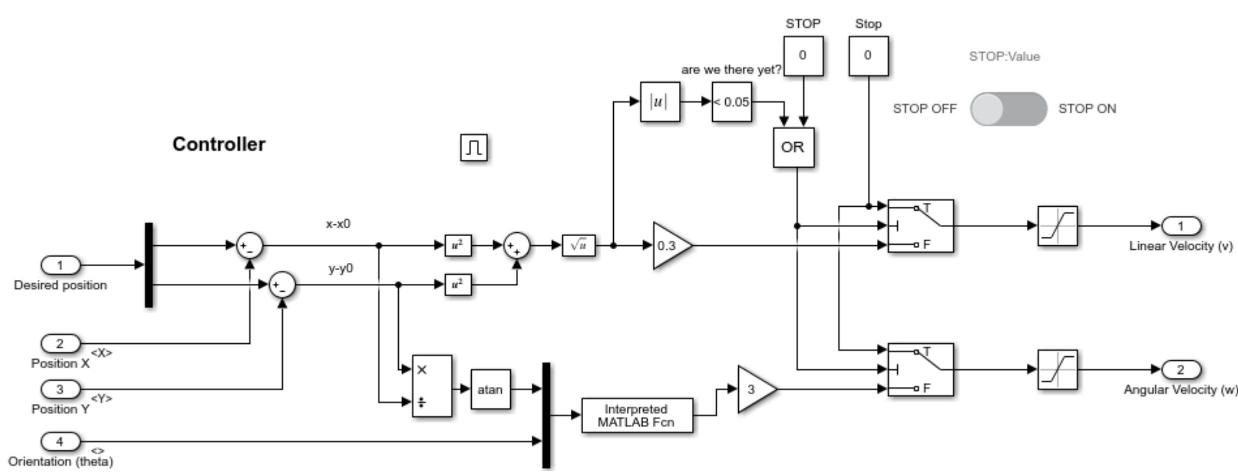
Exercises

Task 1

Moving from point A to B is achieved with a controller based on the bicycle model. The signals are extracted from the ROS network and the quaternion orientation is translated to Euler angles. Because the motion is planar, we are only interested in the rotation about the Z axis, the yaw, to control the heading of the robot. The system looks like the one shown in the picture below.

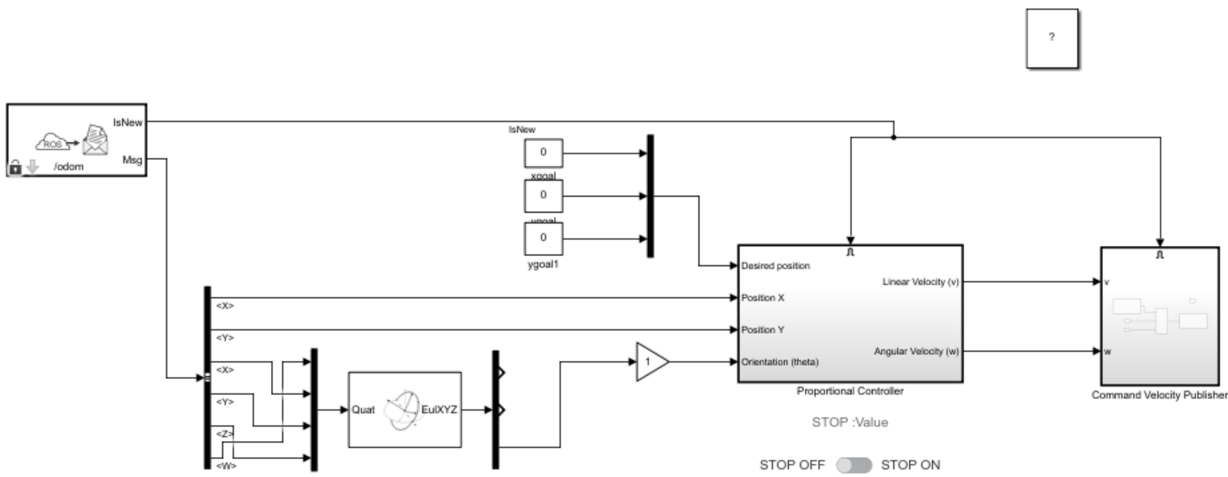


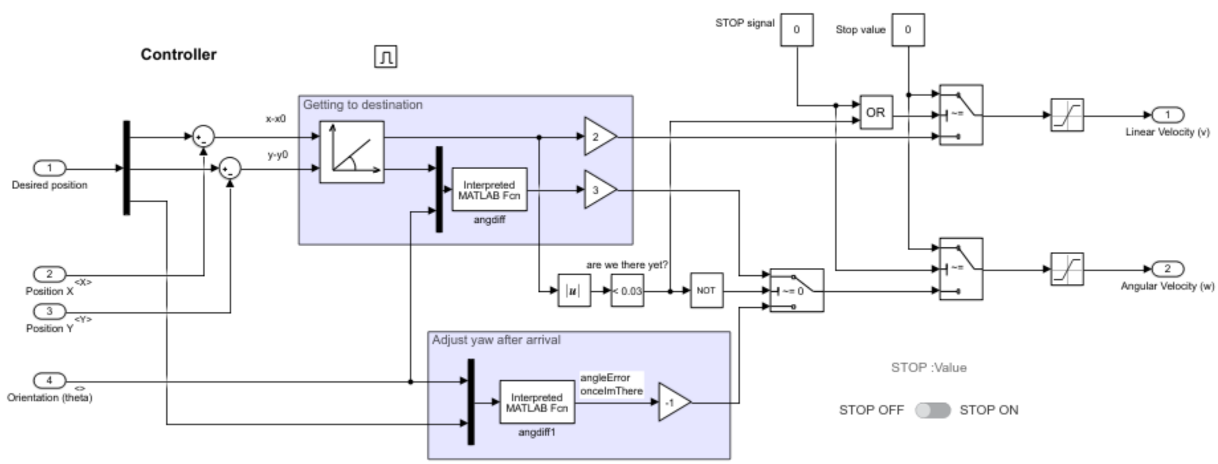
The proportional controller matches the heading of the robot with the angle between the vector joining the robot and its destination. Both the speed and the turning rate are controlled with a gain proportional to the error. Saturation boundaries were included in the speed to avoid instability in the system. If the speed is to high, which can happen if the destination is too far away from the robot, Turtlebot will start to drift and lose control over its position. An arrival condition is also placed so the robot won't overpass its destination.



Task 2

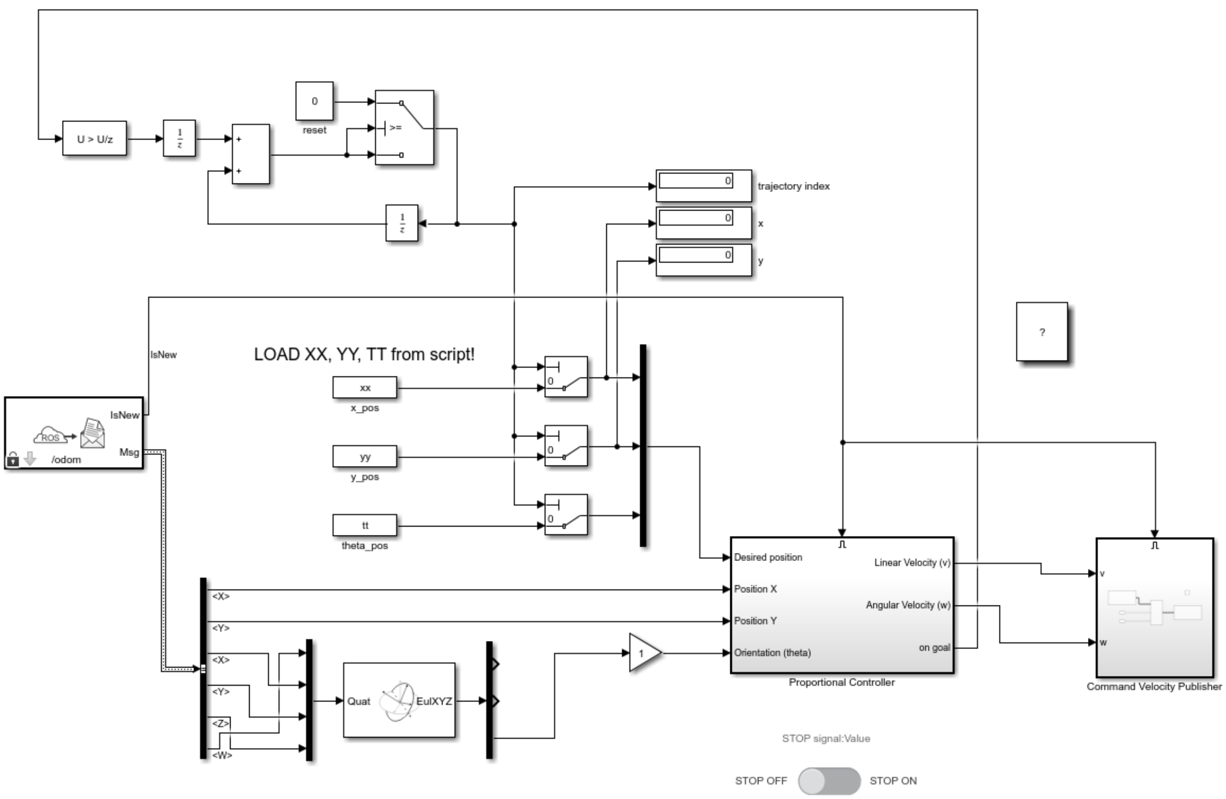
Moving from pose A to B is achieved with a modification of the previous case. The algorithm depicted by Peter Corke in his book for a biclycle model shows issues once the robot reaches its destination, because Turtlebot can yaw . For simplicity, an extra controller will adjust the yaw of the robot once it reaches its destination to match the requested pose. The model and the controller are shown in the picture below.





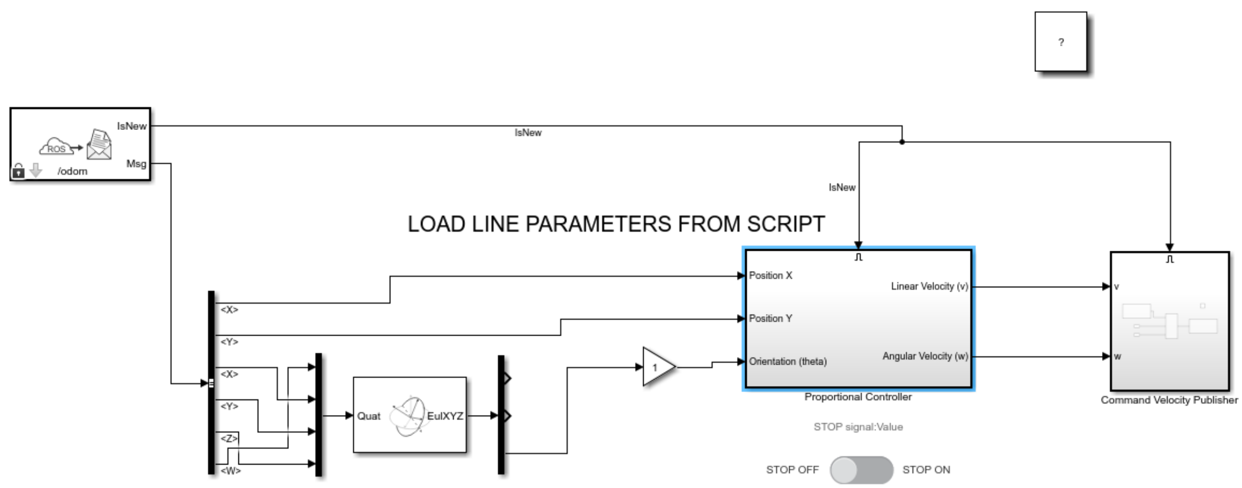
Task 3

For trajectory following, the same strategy as before is used. This time, once the robot reaches its destination, a new target will be loaded. For running the required trajectories, these are first sampled in a script file. After allocating the variables by running a section of the code, the following system will make Turtlebot follow the required points sequentially. The controller is the same as in the previous tasks, only now with a logical output which indicates with a Boolean value if the robot has reached its goal.

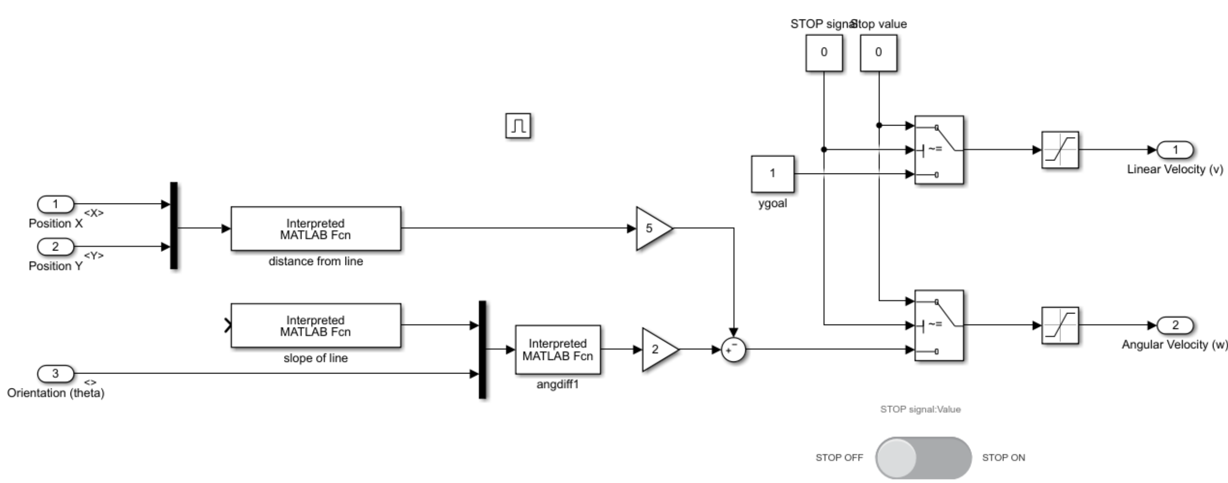


Task 4

Following a line could be achieved by the same technique as the previous task. However for the purpose of this exercise the model proposed by Peter Corke will be applied. The parameters for the line must be run from a matlab script to load the variables with their adequate values, then the Simulink model can be run to make the robot follow the line.

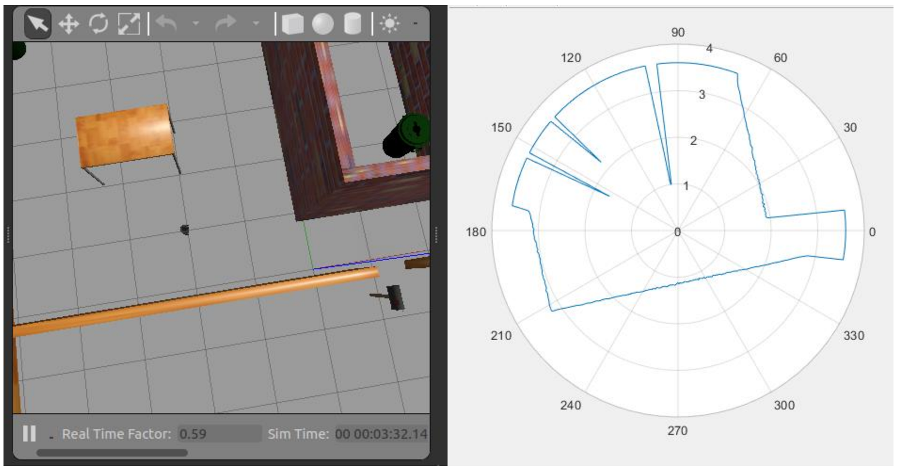


The controller adjusts the heading of the robot to both approach and then math the line slope. The speed is set to a constant value for the whole period.

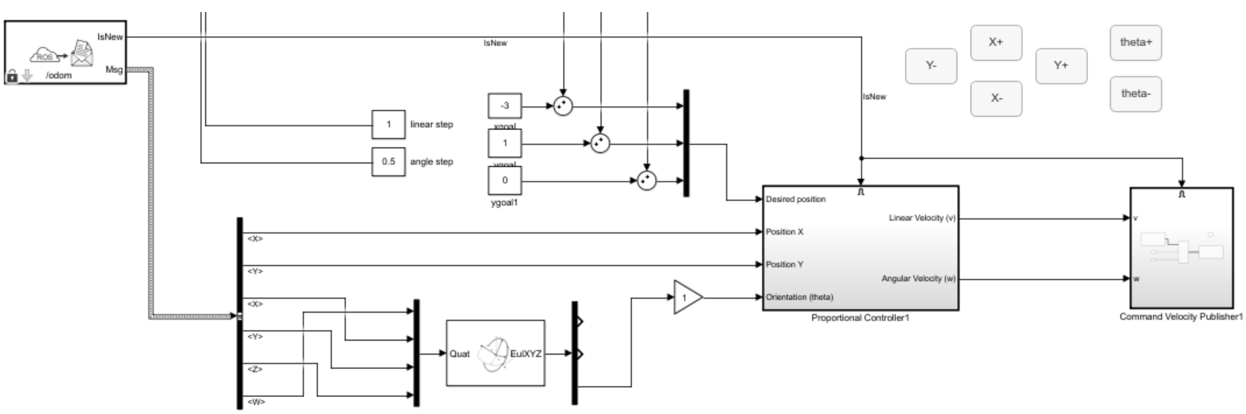


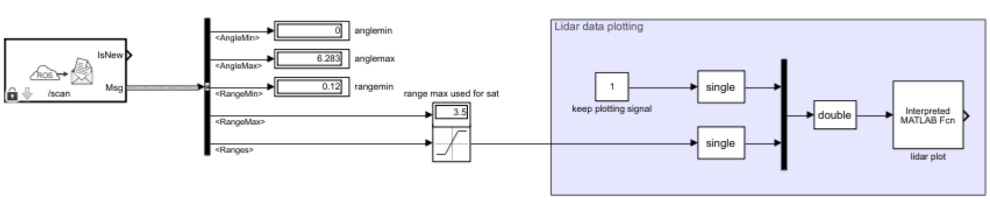
Task 5

Plotting the LIDAR signal was achieved by an external function that is called from the Simulink environment. It is observed that the LIDAR has a maximum detection range of 3.5 meters. When the robot is very close to a surface, the minimum distance the LIDAR shows is of 0.12 meters. Some buttons were added to the model to make it easier to navigate the environment.



The most significant parts of the model are shown below.





Matlab script for tasks 3 & 4

Use Run section on the following snippets before running the Simulink file to load the different trajectories.

```
%% square
square = [...
    1    0    0;...
    1    1    0;...
    0    1    0;...
    0    0    0]
xx = square(:,1)
yy = square(:,2)
tt = square(:,3)
lim = length(xx)

%% letter S
letter_S = [...
    1    0    0;...
    1    1    0;...
    0    1    0;...
    0    2    0;...
    1    2    0;...
    0    0    0]
xx = letter_S(:,1)
yy = letter_S(:,2)
tt = letter_S(:,3)
lim = length(xx)

%% sinusoidal
xx = 0:0.5:2*3.14
yy = sin(0:0.5:2*3.14)
tt = zeros(size(xx))
lim = length(xx)


%% Following line 1
a = 1
b = -1
c = 1

%% Following line 2
a = 1
b = -2
c = 4
```

Matlab script for tasks 5

This function is called every iteration loop from within the Simulink environment to plot the LIDAR signal.

```
function a = plot_lidar(control, ranges)
    a = 1;
    if control == 1
        clf
        rng = ranges
        ang = linspace(0, 2*pi, numel(ranges))
        polarplot(ang, rng)
    end
end
```