Problem 1>

1> Set up Ros Kinetic with gazebo
2> Created a Robot URDF to add to wheels and laserscan sensor.
3> Created a urdf parser to link to ROS
4> Set up catkin workspace
5> Update package details in CMAKELIST and package.xml
6> Code the keyboard control behaviour in the pipeline.
7> Create launch files in the launch folder to link the codes in the scripts folder, urdf parser in the src file and the gazebo world.

Keyboard map:
u :left rotate
I :forward movement
o :right rotate
k :stop
j : left turn
l : right turn
m : left rotate
, :move backwards
. : rotate right
q : to increase speed
z : decrease speed
Ctrl-C to quit

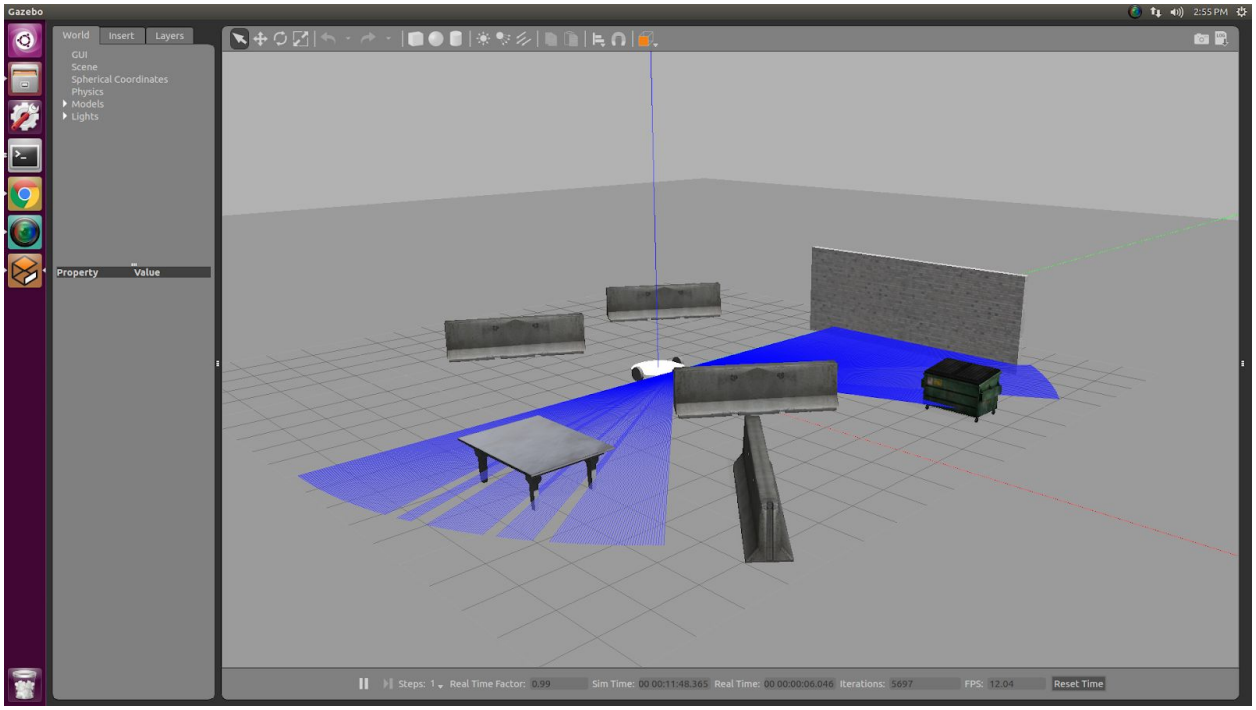
Following is the pseudocode :

1. Initialize the ros node.
2. Write a function to read inputs from keyboards using sys,select,termios and tty
3. Create a dictionary to map keyboard buttons with x and y movement values.
4. Create a dictionary to map keyboard input keys to increase and decrease the speed.
5. Create a publisher of type “twist” to publish commands to the robot to give linear and angular speeds.
6. Put the function in try and catch to handle exceptions where keyboard inputs are not decoded due to queue.

Issues faced:

1. Was trying to make one launch file which can invoke urdf, ros code and gazebo. Solved the same by creating separate launch files for the three.
2. Creating the model in gazebo world created a problem as the world would explode upon as the robot starts moving. Solved this by creating the world in the form of a model and importing the model.

Images during development (Final demo video attached in zip):



Setting up the robot urdf and the gazebo world

Problem 2 >

1> Set up Ros Kinetic with gazebo
2> Created a Robot URDF to add to wheels and laserscan sensor.
3> Created a urdf parser to link to ROS
4> Set up catkin workspace
5> Update package details in CMAKELIST and package.xml
6> Code the collision avoidance and wandering behaviour in the pipeline.
7> Create launch files to link the code in the scripts, urdf parser in the src file and the gazebo world.

Following is the pseudocode :

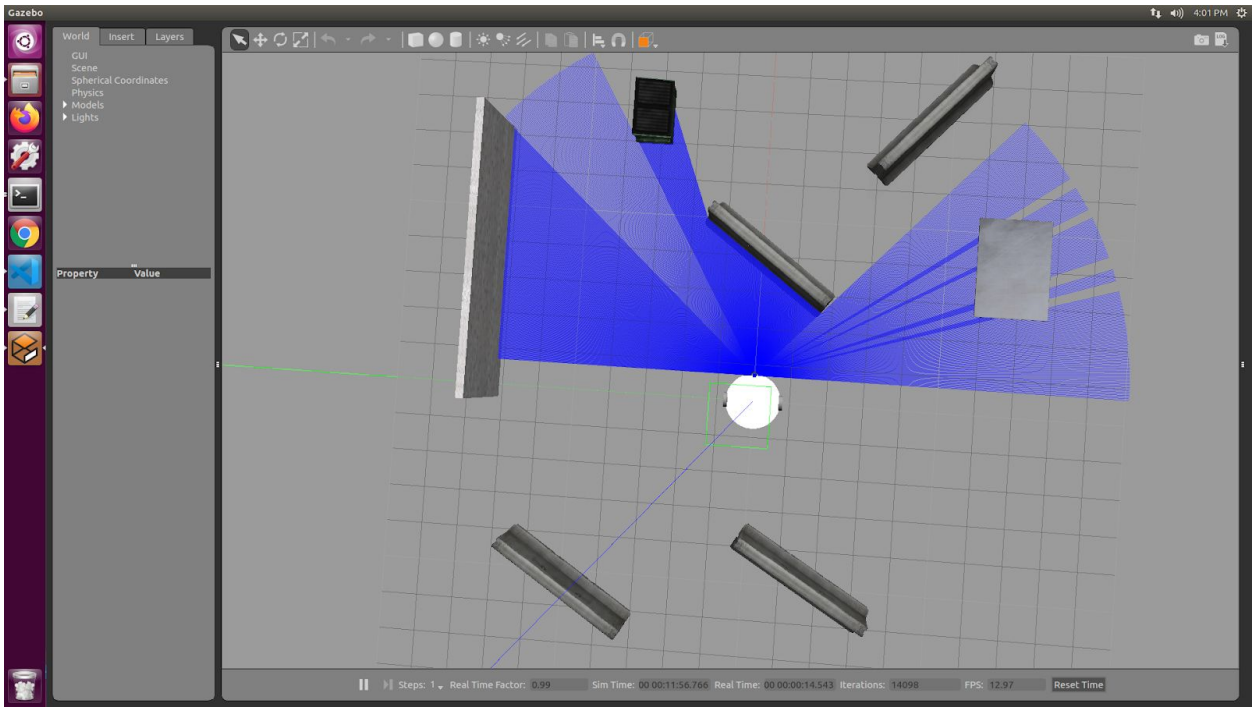
1. Initialize the ros node.
2. Initialize the ros subscriber to read the laserscan input
3. Initialize the ros publisher to publish the twist commands.
4. Set the publisher-subscriber frequency to 10 hz.
5. Set a threshold distance at which robot should turn to avoid collision

6. If the object is within the threshold distance, the robot will turn by giving twist.angular command. If the object is beyond the threshold distance, the robot will continue to move forward.

Issues faced:

1. The laserscan data returns a range of data values. I had to identify the data node that I had to use for collision avoidance behaviour.

Images during development (Final demo video attached in zip):



After setting up Laserscan on my robot