Innovation Centre for Education

Student Exercise Guide

Contents

Exercise 1. Time Domain and frequency Domain representation using Python

Estimated time:

60.00 minutes

What this exercise is about:

This exercise explains how to represent time domain and frequency domain signal using python.

What you should be able to do:

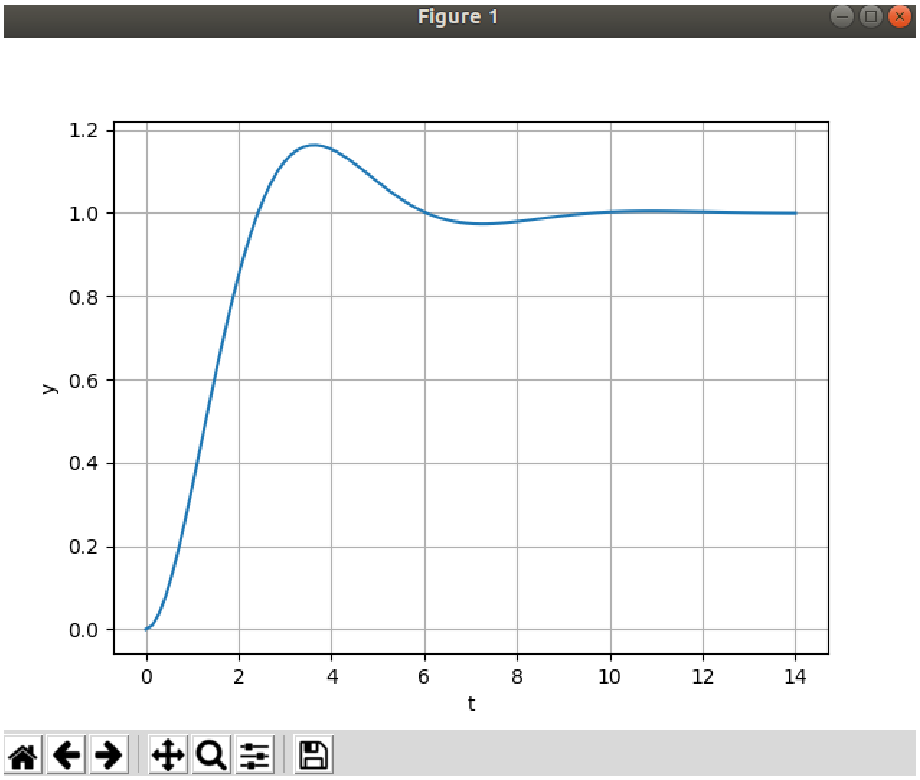
- Write a code to represent signal in time domain.
- Write a code to represent signal in frequency domain.

Instructor exercise overview:

Python code: Time domain representation of the signal

```
from control  import  *

import matplotlib.pyplot as plt

import pylab as pl

g = tf (1, [1,1,1])

t,y = step_response (g)

plt.plot(t,y)

plt.xlabel('t')

plt.ylabel('y')

plt.grid()

pl.show()
```

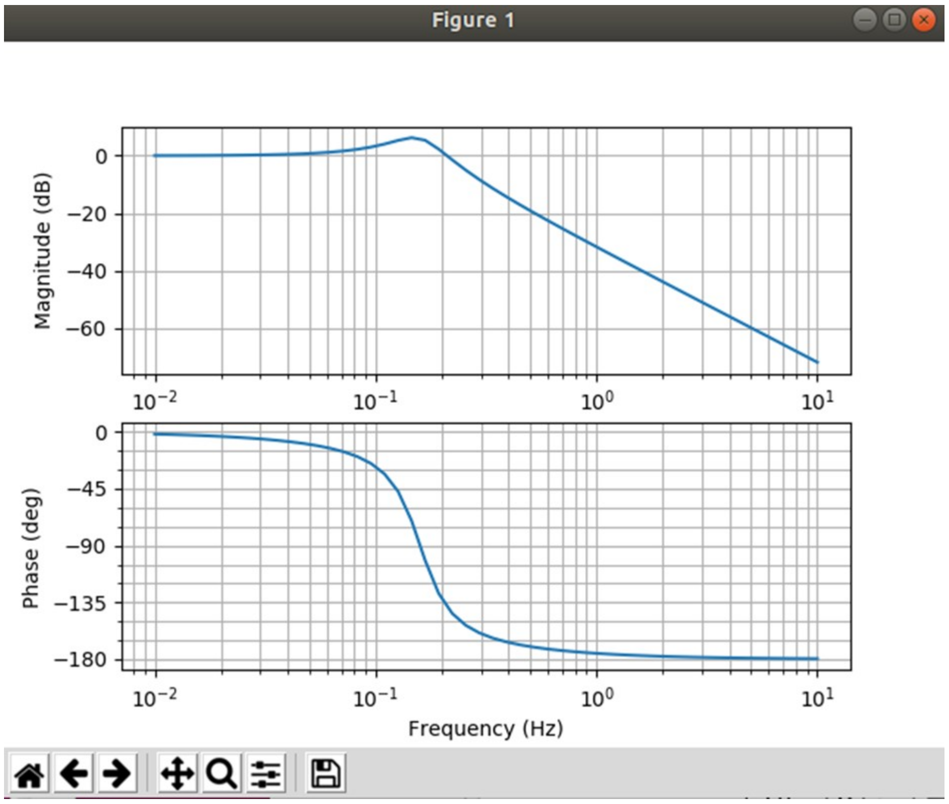
Result:



Python code: Frequency domain representation of the signal

```
from control  import  *
import matplotlib.pyplot as plt
from control.matlab import c2d
import pylab as pl
g = tf ([1], [1,0.5,1])
bode_plot(g, dB=True);
pl.show()
```

Result:



End of Exercise.

Exercise 2. Pendulum Simulation using Python: example of a system

Estimated time:

90.00 minutes

What this exercise is about:

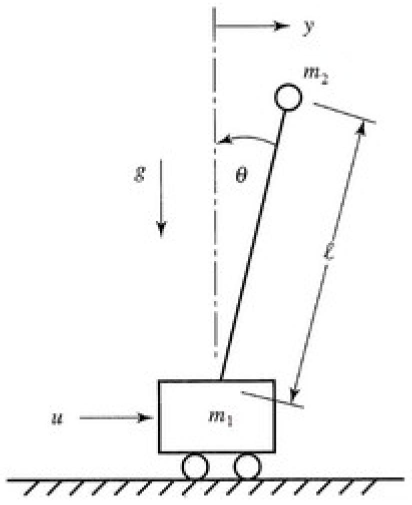
Designing a model predictive controller for an inverted pendulum system with an adjustable cart.

What you should be able to do:

- Design a model predictive controller for an inverted pendulum system with an adjustable cart.
- Demonstrate that the cart can perform a sequence of moves to maneuver from position *y*=-1.0 to *y*=0.0 within 6.2 seconds.

Introduction:

Design a model predictive controller for an inverted pendulum system with an adjustable cart. Demonstrate that the cart can perform a sequence of moves to maneuver from position *y*=-1.0 to *y*=0.0 within 6.2 seconds. Verify that *v*, θ, and *q* are zero before and after the maneuver.



The inverted pendulum is described by the following dynamic equations:

$$\begin{bmatrix}\dot{y}\\ \dot{v}\\ \dot{\theta}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -\epsilon & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} \begin{bmatrix}y\\ v\\ \theta\\ q\end{bmatrix} + \begin{bmatrix}0\\ 1\\ 0\\ -1\end{bmatrix} u$$

where *u* is the force applied to the cart, ε is *m2/(m1+m2)*, y is the position of the cart, *v* is the velocity of the cart, θ is the angle of the pendulum relative to the cart, *m1*=10, *m2*=1, and *q* is the rate of angle change. Tune the controller to minimize the use of force applied to the cart either in the forward or reverse direction (i.e. minimize fuel consumed to perform the maneuver). Explain the tuning and the optimal solution with appropriate plots that demonstrate that the solution is optimal.

Instructor exercise overview:

Python code:

```
# Contributed by Everton Colling
import matplotlib.animation as animation
import numpy as np
from gekko import GEKKO
#Defining a model
m = GEKKO()
#################################
#Weight of item
m2 = 1
#################################
#Defining the time, we will go beyond the 6.2s
#to check if the objective was achieved
m.time = np.linspace(0,8,100)
end_loc = int(100.0*6.2/8.0)
#Parameters
m1a = m.Param(value=10)
m2a = m.Param(value=m2)
final = np.zeros(len(m.time))
for i in range(len(m.time)):
    if m.time[i] < 6.2:
        final[i] = 0
    else:
        final[i] = 1
final = m.Param(value=final)
```

```
#MV
ua = m.Var(value=0)
#State Variables
theta_a = m.Var(value=0)
qa = m.Var(value=0)
ya = m.Var(value=-1)
va = m.Var(value=0)
#Intermediates
epsilon = m.Intermediate(m2a/(m1a+m2a))

#Defining the State Space Model
m.Equation(ya.dt() == va)
m.Equation(va.dt() == -epsilon*theta_a + ua)
m.Equation(theta_a.dt() == qa)
m.Equation(qa.dt() == theta_a -ua)

#Definine the Objectives
#Make all the state variables be zero at time >= 6.2
m.Obj(final*ya**2)
m.Obj(final*va**2)
m.Obj(final*theta_a**2)
m.Obj(final*qa**2)

m.fix(ya,pos=end_loc,val=0.0)
m.fix(va,pos=end_loc,val=0.0)
m.fix(theta_a,pos=end_loc,val=0.0)
m.fix(qa,pos=end_loc,val=0.0)
#Try to minimize change of MV over all horizon
m.Obj(0.001*ua**2)

m.options.IMODE = 6 #MPC
m.solve() #(disp=False)

#Plotting the results
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12,10))

plt.subplot(221)
plt.plot(m.time,ua.value,'m',lw=2)
plt.legend([r'$u$'],loc=1)
plt.ylabel('Force')
plt.xlabel('Time')
plt.xlim(m.time[0],m.time[-1])

plt.subplot(222)
plt.plot(m.time,va.value,'g',lw=2)
plt.legend([r'$v$'],loc=1)
plt.ylabel('Velocity')
plt.xlabel('Time')
plt.xlim(m.time[0],m.time[-1])

plt.subplot(223)
plt.plot(m.time,ya.value,'r',lw=2)
plt.legend([r'$y$'],loc=1)
plt.ylabel('Position')
plt.xlabel('Time')
plt.xlim(m.time[0],m.time[-1])

plt.subplot(224)
plt.plot(m.time,theta_a.value,'y',lw=2)
plt.plot(m.time,qa.value,'c',lw=2)
plt.legend([r'$\theta$',r'$q$'],loc=1)
plt.ylabel('Angle')
plt.xlabel('Time')
plt.xlim(m.time[0],m.time[-1])

plt.rcParams['animation.html'] = 'html5'

x1 = ya.value
y1 = np.zeros(len(m.time))
```

```
#suppose that l = 1
x2 = 1*np.sin(theta_a.value)+x1
x2b = 1.05*np.sin(theta_a.value)+x1
y2 = 1*np.cos(theta_a.value)-y1
y2b = 1.05*np.cos(theta_a.value)-y1

fig = plt.figure(figsize=(8,6.4))
ax = fig.add_subplot(111,autoscale_on=False,\
                     xlim=(-1.5,0.5),ylim=(-0.4,1.2))
ax.set_xlabel('position')
ax.get_yaxis().set_visible(False)

crane_rail, = ax.plot([-1.5,0.5],[-0.2,-0.2],'k-',lw=4)
start, = ax.plot([-1,-1],[-1.5,1.5],'k:',lw=2)
objective, = ax.plot([0,0],[-0.5,1.5],'k:',lw=2)
mass1, = ax.plot([],[],linestyle='None',marker='s',\
                 markersize=40,markeredgecolor='k',\
                 color='orange',markeredgewidth=2)
mass2, = ax.plot([],[],linestyle='None',marker='o',\
                 markersize=20,markeredgecolor='k',\
                 color='orange',markeredgewidth=2)
line, = ax.plot([],[],'o-',color='orange',lw=4,\
                markersize=6,markeredgecolor='k',\
                markerfacecolor='k')
time_template = 'time = %.1fs'
time_text = ax.text(0.05,0.9,'',transform=ax.transAxes)
start_text = ax.text(-1.06,-0.3,'start',ha='right')
end_text = ax.text(0.06,-0.3,'objective',ha='left')

def init():
    mass1.set_data([],[])
    mass2.set_data([],[])
    line.set_data([],[])
    time_text.set_text('')
    return line, mass1, mass2, time_text

def animate(i):
    mass1.set_data([x1[i]],[y1[i]-0.1])
    mass2.set_data([x2b[i]],[y2b[i]])
    line.set_data([x1[i],x2[i]],[y1[i],y2[i]])
    time_text.set_text(time_template % m.time[i])
    return line, mass1, mass2, time_text

ani_a = animation.FuncAnimation(fig, animate, \
         np.arange(1,len(m.time)), \
         interval=40,blit=False,init_func=init)

# requires ffmpeg to save mp4 file
#  available from https://ffmpeg.zeranoe.com/builds/
#  add ffmpeg.exe to path such as C:\ffmpeg\bin\ in
#  environment variables

#ani_a.save('Pendulum_Control.mp4',fps=30)

plt.show()
```

Result:

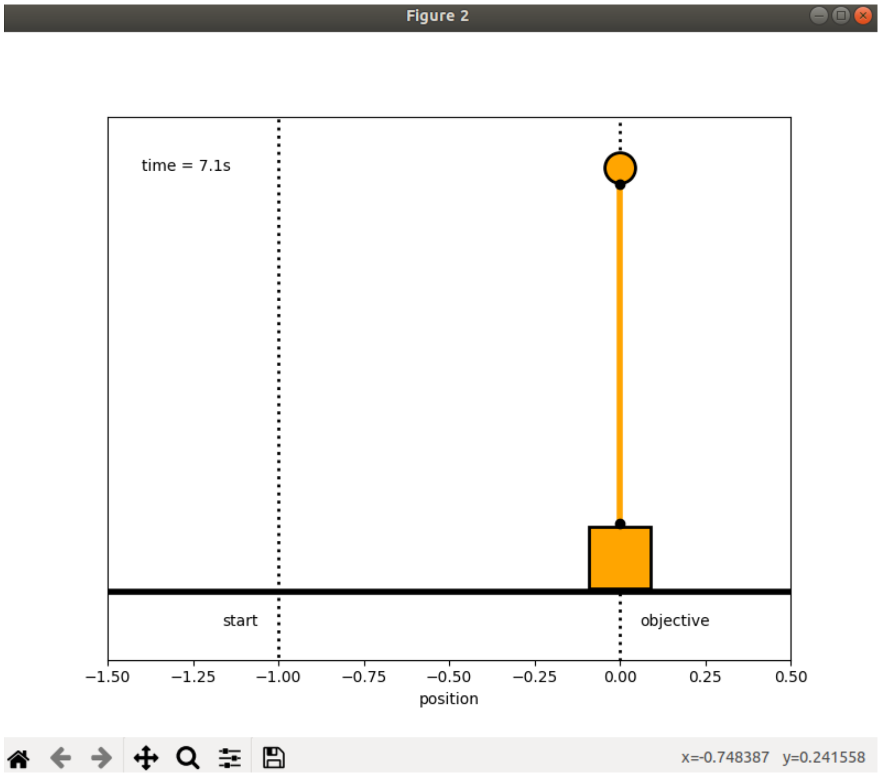


Figure 2
time = 7.1s
start
objective
−1.50
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
position
x=-0.748387 y=0.241558

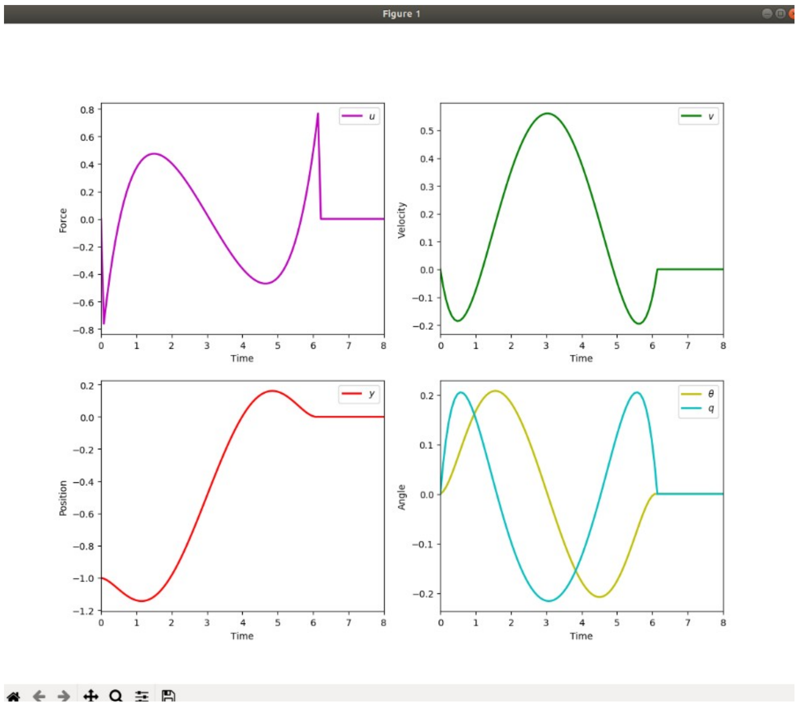
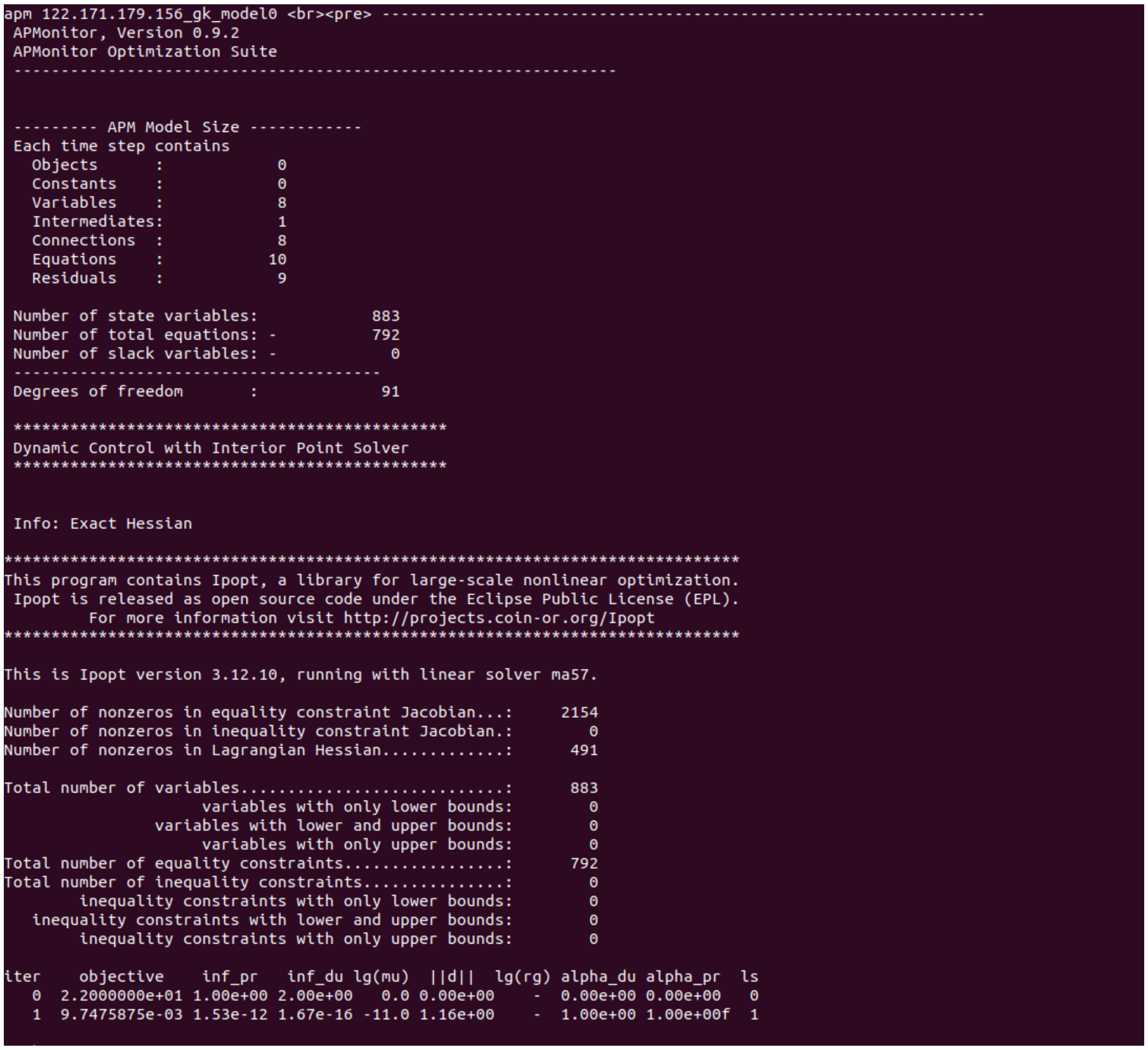


Figure 1
u
Force
Time
v
Velocity
y
Position
θ
q
Angle



```
apm 122.171.179.156_gk_model0 <br><pre> ----------------------------------------------------------------
 APMonitor, Version 0.9.2
 APMonitor Optimization Suite
 ----------------------------------------------------------------


 --------- APM Model Size ------------
 Each time step contains
   Objects      :            0
   Constants    :            0
   Variables    :            8
   Intermediates:            1
   Connections  :            8
   Equations    :           10
   Residuals    :            9

 Number of state variables:            883
 Number of total equations: -          792
 Number of slack variables: -            0
 ---------------------------------------
 Degrees of freedom       :             91

 **********************************************
 Dynamic Control with Interior Point Solver
 **********************************************


 Info: Exact Hessian

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.10, running with linear solver ma57.

Number of nonzeros in equality constraint Jacobian...:     2154
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:      491

Total number of variables............................:      883
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:      792
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  2.2000000e+01 1.00e+00 2.00e+00   0.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  9.7475875e-03 1.53e-12 1.67e-16 -11.0 1.16e+00    -  1.00e+00 1.00e+00f  1
```

End of Exercise.

Exercise 3. 8 queens problem using Python

Estimated time:

90.00 minutes

What this exercise is about:

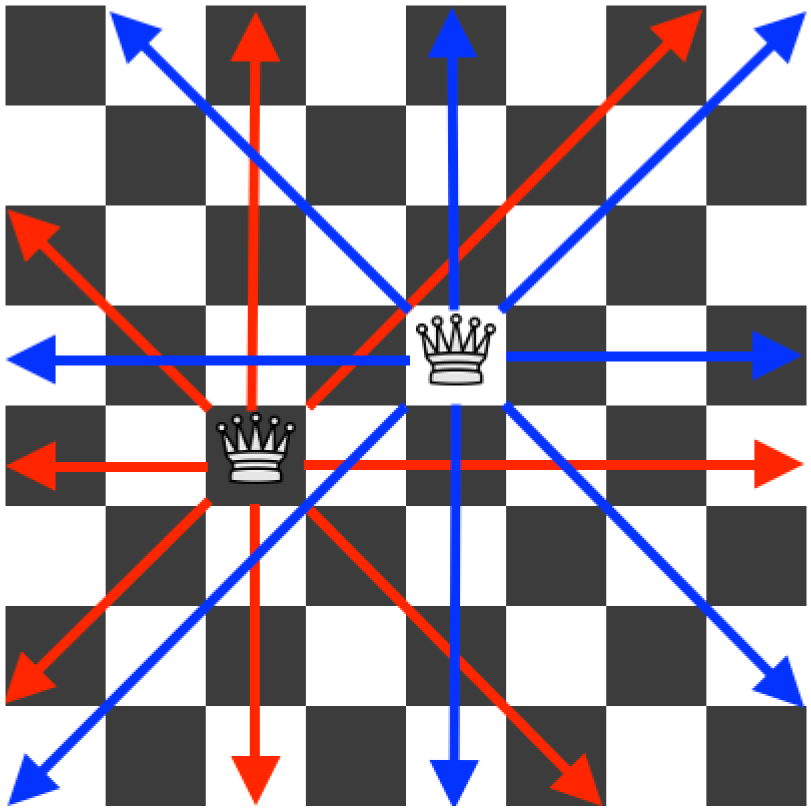
This exercise is about the eight queens puzzle, or the eight queens problem, asks how to place eight queens on a chessboard without attacking each other.

What you should be able to do:

Generate a solution to the problem by scanning each row of the board and placing one queen per column, while checking at every step, that no two queens are in the line of attack of the other

Introduction:

The eight queens puzzle, or the eight queens problem, asks how to place eight queens on a chessboard without attacking each other. In the below figure, you can see two queens with their attack patterns:



We can generate a solution to the problem by scanning each row of the board and placing one queen per column, while checking at every step, that no two queens are in the line of attack of the other. A brute force approach to the problem will be to generate all possible combinations of the eight queens on the chessboard and reject the invalid states. How many combinations of 8 queens on a 64 cells chessboard are possible ?

The combinations formula is

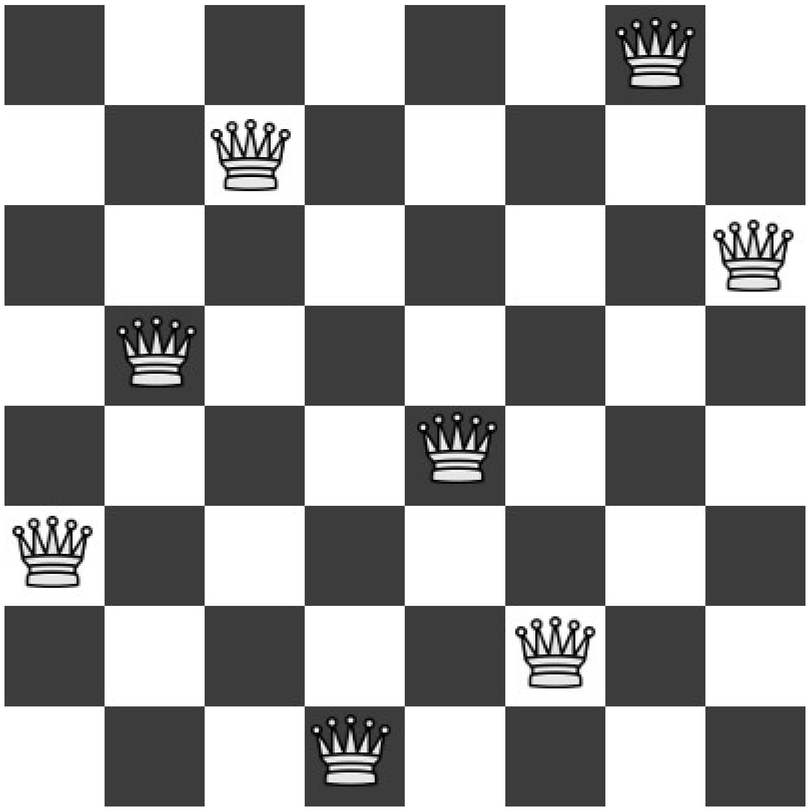
$$C(n,k) =_n C_k = \frac{n!}{k! \cdot (n-k)!}$$

which, for our particular case is:

$$C(64,8) =_{64} C_8 = \frac{64!}{8! \cdot (64-8)!} = 4,426,165,368$$

Clearly, the brute force approach is not practical!

We can further reduce the number of potential solutions if we observe that a valid solution can have only one queen per row, which means that we can represent the board as an array of eight elements, where each entry represents the column position of the queen from a particular row. Take as an example the next solution of the problem:



The queens positions on the above board, can be represented as the occupied positions of a two dimensional 8x8 array: [0, 6], [1, 2], [2, 7], [3, 1], [4, 4], [5, 0], [6, 5], [7, 3]. Or, as described above, we can use a one dimensional 8 elements array: [6, 2, 7, 1, 4, 0, 5, 3].

If we look closely at the example solution [6, 2, 7, 1, 4, 0, 5, 3], we note that a potential solution to the eight queens puzzle can be constructed by generating all possible permutations of an array of eight numbers, [0, 1, 2, 3, 4, 5, 6, 7], and rejecting the invalid states (the ones in which any two queens can attack each other). The number of all permutations of *n* unique objects is *n!*, which for our particular case is:

n!=40,320

which is more reasonable than the previous *4,426,165,368* situations to analyze for the brute force approach.

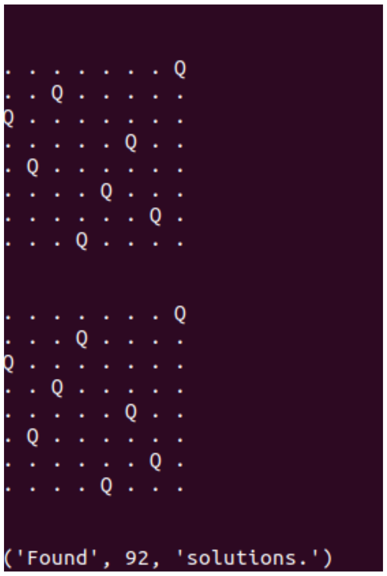
A slightly more efficient solution to the puzzle uses a recursive approach: assume that we've already generated all possible ways to place *k* queens on the first *k* rows. In order to generate the valid positions for the *k+1* queen we place a queen on all columns of row *k+1* and we reject the invalid states. We do the above steps until all eight queens are placed on the board. This approach will generate all 92 distinct solutions for the eight queens puzzle.

Instructor exercise overview:

Python code:

```
"""The n queens puzzle."""
class NQueens:
    """Generate all valid solutions for the n queens puzzle"""
    def __init__(self, size):
        # Store the puzzle (problem) size and the number of valid solutions
        self.size = size
        self.solutions = 0
        self.solve()

    def solve(self):
        """Solve the n queens puzzle and print the number of solutions"""
        positions = [-1] * self.size
        self.put_queen(positions, 0)
        print("Found", self.solutions, "solutions.")

    def put_queen(self, positions, target_row):
        """
        Try to place a queen on target_row by checking all N possible cases.
        If a valid place is found the function calls itself trying to place a
queen
        on the next row until all N queens are placed on the NxN board.
        """
        # Base (stop) case - all N rows are occupied
        if target_row == self.size:
            self.show_full_board(positions)
            # self.show_short_board(positions)
            self.solutions += 1
        else:
            # For all N columns positions try to place a queen
            for column in range(self.size):
                # Reject all invalid positions
                if self.check_place(positions, target_row, column):
                    positions[target_row] = column
                    self.put_queen(positions, target_row + 1)



    def check_place(self, positions, ocuppied_rows, column):
        """
        Check if a given position is under attack from any of
        the previously placed queens (check column and diagonal positions)
        """
        for i in range(ocuppied_rows):
            if positions[i] == column or \ i
                positions[i] - == column -   ocuppied_rows or \
                positions[i] + i == column + ocuppied_rows:

                return False
        return True

    def show_full_board(self, positions):
        """Show the full NxN board""" for
        row in range(self.size):
            line = ""
            for column in range(self.size):
                if positions[row] == column:
                    line += "Q "
                else:
                    line += ". "
            print(line)
        print("\n")

    def show_short_board(self, positions):
        """
        Show the queens positions on the board in compressed form,
        each number represent the occupied column position  in the corresponding
row.
        """
        line = ""
        for i in range(self.size):
            line += str(positions[i]) + " "
        print(line)

def main():
    """Initialize and solve the n queens puzzle"""
    NQueens(8)

if   name___== "  main  ":
    # execute only if run as a script
    main()
```

Result:



```
. . . . . . . Q
. . Q . . . . .
Q . . . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . Q . . .
. . . . . . Q .
. . . Q . . . .


. . . . . . . Q
. . . Q . . . .
Q . . . . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . . Q .
. . . . Q . . .


('Found', 92, 'solutions.')
```

End of Exercise.

Exercise 4. Search Algorithms using python

Estimated time:

90.00 minutes

What this exercise is about:

Taking a list and key as input and finds the index of the key in the list using linear search.

What you should be able to do:

Write a program that takes a list and key as input and finds the index of the key in the list using linear search.

Introduction:

- Create a function linear search that takes a list and key as arguments
- A loop iterates through the list and when an item matching the key is found, the corresponding index is returned.
- If no such item is found, -1 is returned.

Instructor exercise overview:

Python code:

```
def linear_search(alist, key):
    """Return index of key in alist. Return -1 if key not present."""
    for i in range(len(alist)):
        if alist[i] == key:
            return i
    return -1



alist = input('Enter the list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = linear_search(alist, key)
if index < 0:
    print('{} was not found.'.format(key))
else:
    print('{} was found at index {}.'.format(key, index))
```

Result:

```
Enter the list of numbers: 3 4 5
The number to search for: 2
2 was not found.

Enter the list of numbers: 5 3 2 4 6
The number to search for: 3
3 was found at index 1.
```

End of Exercise.

Exercise 5. Hill Climbing using python

Estimated time:

90.00 minutes

What this exercise is about:

Travelling salesman problem solution using randomized hill climbing and simulated annealing.

What you should be able to do:

Write a program implements two search strategies for N cities Travelling Salesman Problem with cities being numbered from 0 to N-1.

Introduction:

Travelling Salesman Problem solution using Randomized hill climbing and Simulated Annealing This program implements two search strategies for N cities Travelling Salesman Problem with cities being numbered from 0 to N-1. This program uses three different cost functions to calculate the cost of the tour. The program takes the following inputs from the command line :

- Number of cities
- MEB (The Maximum number of states to be searched by the algorithm)
- Enter the cost function either c1 c2 or c3. *Note Please enter c1 for cost function c1 and so on
- Enter the seed
- Enter the search strategy. Enter 1 for simple search(randomized hill climbing) and enter 2 for sophisticated search (Simulated Annealing)

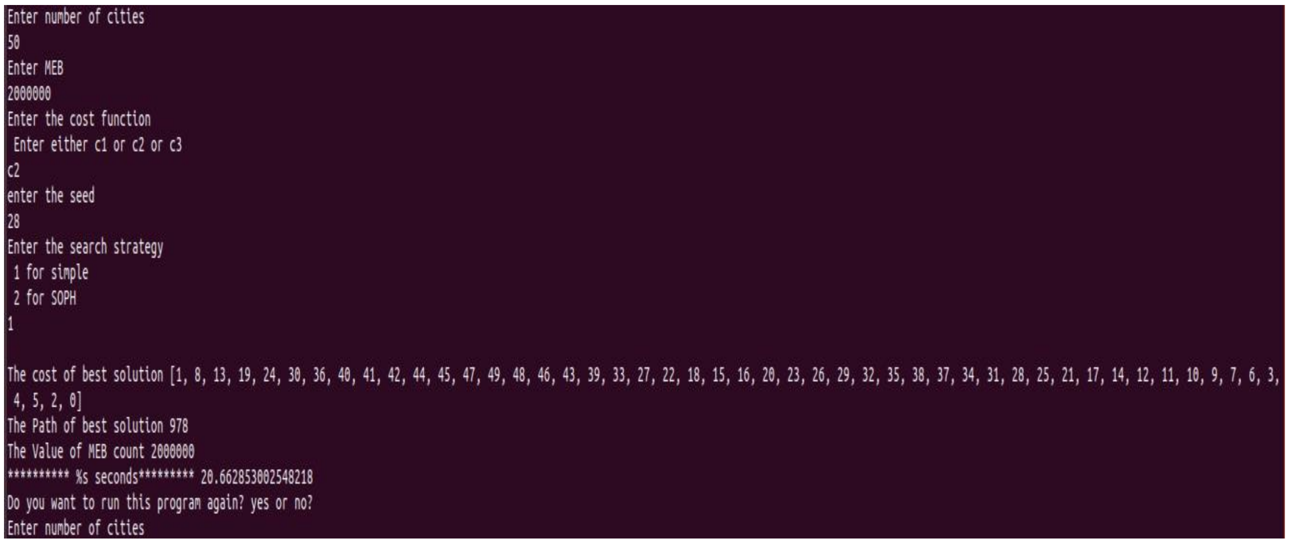
Instructor exercise overview:

Python code:

```
import random
import math
import time

#Three cost functions are described in the project guidelines

def cost1(x,y):
    if x==y:
        return 0
    elif x<3 and y<3:
        return 1
    elif x<3:
        return 200
    elif y<3:
        return 200
    elif (x%7)==(y%7):
        return 2
    else:
        return abs(x-y)+3
    return

def cost2(x,y):
    if x==y:
        return 0
    elif (x+y)<10:
        return abs(x-y)+4
    elif [(x+y)%11]==0:
        return 3
    else:
        return abs(x-y)**2+10
    return

def cost3(x,y):
    if x==y:
        return 0
    else:
        return (x+y)**2
    return

#function that returns a random path given a total number of cities
def random_path(no_cities,seed1):
    tour=list(range(no_cities))
    random.seed(seed1)
    random.shuffle(tour)
    return tour



#Given a list of cities, calculates the cost of the tour
def tour_cost(tours,cost_fun):
    total_cost =0
    cost_i=0
    n=len(tours)
```

```
        for i,city in enumerate(tours):
            if i==n-1:
                if(cost_fun=="c1"):
                    cost_i = cost1(tours[i],tours[0])

                if(cost_fun=="c2"):
                    cost_i = cost2(tours[i],tours[0])



                if(cost_fun=="c3"):
                    cost_i = cost3(tours[i],tours[0])



                total_cost=total_cost+cost_i

            else:
                if(cost_fun=="c1"):
                    cost_i = cost1(tours[i],tours[i+1])

                if(cost_fun=="c2"):
                    cost_i = cost2(tours[i],tours[i+1])

                if(cost_fun=="c3"):
                    cost_i = cost3(tours[i],tours[i+1])



                total_cost=total_cost+cost_i

        return total_cost

    # mutation operator that swaps two cities randomly to  create a new path

    def mutation_operator(tours):
        r1= list(range(len(tours)))
        r2= list(range(len(tours)))
        random.shuffle(r1)
        random.shuffle(r2)
        for i in r1:
            for j in r2:
                if i < j:
                    next_state =tours[:]
                    next_state[i],next_state[j]=tours[j],tours[i]
                    yield next_state

    #probabilistically choosing a neighbour
    def  Probability_acceptance(prev_score,next_score,temperature):
        if next_score < prev_score:
            return 1.0
        elif temperature == 0:
            return 0.0
        else:
            return math.exp( -abs(next_score-prev_score)/temperature  )

    #The cooling schedule based on  kirkpatrick model
    def cooling_schedule(start_temp,cooling_constant):
        T=start_temp
        while True:
            yield T
            T= cooling_constant*T

    #This function implements randomized hill climbing for TSP
    def randomized_hill_climbing(no_cities,cost_func,MEB,seed1):

        best_path=random_path(no_cities,seed1)
        best_cost = tour_cost(best_path,cost_func)
        evaluations_count=1
        while evaluations_count < MEB:
            for next_city in mutation_operator(best_path):
                if evaluations_count == MEB:
                    break
                str1 = ''.join(str(e) for e in next_city)
                #Skip calculating the cost of repeated paths
                if str1 in dict:
                    evaluations_count+=1
                    continue

                next_tCost=tour_cost(next_city,cost_func)
                #store it in the dictionary
                dict[str1] = next_tCost
                evaluations_count+=1

                #selecting the path with lowest cost
                if next_tCost < best_cost:
                    best_path=next_city
                    best_cost=next_tCost

        return best_cost,best_path,evaluations_count
    #This function implements simulated annealing for TSP
    def simulated_annealing(no_cities,cost_func,MEB,seed1):

        start_temp=70
        cooling_constant=0.9995
        best_path = None
        best_cost = None
        current_path=random_path(int(no_cities),seed1)
        current_cost=tour_cost(current_path,cost_func)

        if best_path is None or   current_cost < best_cost:
            best_cost =  current_cost
            best_path = current_path

        num_evaluations=1
        temp_schedule=cooling_schedule(start_temp,cooling_constant)
        for temperature in  temp_schedule:
            flag = False
            #examinning moves around our current path
            for next_path in mutation_operator(current_path):
                if num_evaluations == MEB:
                    #print "reached meb"
                    flag=True
                    break

                next_cost=tour_cost(next_path,cost_func)

                if best_path is None or  next_cost < best_cost:
                    best_cost =  next_cost
                    best_path = next_path

                num_evaluations+=1
                p=Probability_acceptance(current_cost,next_cost,temperature)
                if random.random() < p:
                    current_path=next_path
                    current_cost=next_cost
                    break

            if flag:
                break

        return best_path,best_cost,num_evaluations

    keeprunning=True
    while keeprunning:

        no_cities=int(input("Enter number of cities \n"))
        MEB=int(input("Enter MEB \n"))
        dict={}
        cost_func=input("Enter the cost function \n Enter either  c1 or c2 or c3
    \n")
        seed1=int(input("enter the seed \n"))
        search_strat=int(input("Enter the search strategy \n 1 for  simple \n 2 for
    SOPH \n"))
        start_time=time.time()
        if(search_strat==1):
            print("This is the output of randomized hill climbing - Simple Search
    \n", file=open("2runs.txt", "a"))

    best_path,best_cost,num_evaluations=randomized_hill_climbing(no_cities,cost_
    func,MEB,seed1)
        elif(search_strat==2):
            print("This is the output of simulated annealing - Sophisticated Search
    \n", file=open("2runs.txt", "a"))

    best_path,best_cost,num_evaluations=simulated_annealing(no_cities,cost_func,
    MEB,seed1)

        else:
            print("Please enter a valid option either 1 or 2 !!")
            break

        print ("The cost of best solution",best_cost)
        print ("The Path of best solution",best_path)
        print ("The Value of MEB count",num_evaluations)
        print("********** %s seconds*********",(time.time()-start_time))
        print("The cost of best Solution",best_cost, file=open("2runs.txt", "a"))
        print("The path of best solution",best_path, file=open("2runs.txt", "a"))
        print("Value of MEB count is ",num_evaluations, file=open("2runs.txt",
    "a"))
        print("**********     %s     seconds*********",(time.time()-start_time),
    file=open("2runs.txt", "a"))
        stop=input("Do you want to run this program again? yes or no?\n")
        if stop=="no":
            keeprunning=False
            break
```

Result:



```
Enter number of cities
50
Enter MEB
2000000
Enter the cost function
 Enter either c1 or c2 or c3
c2
enter the seed
28
Enter the search strategy
 1 for simple
 2 for SOPH
1

The cost of best solution [1, 8, 13, 19, 24, 30, 36, 40, 41, 42, 44, 45, 47, 49, 48, 46, 43, 39, 33, 27, 22, 18, 15, 16, 20, 23, 26, 29, 32, 35, 38, 37, 34, 31, 28, 25, 21, 17, 14, 12, 11, 10, 9, 7, 6, 3,
 4, 5, 2, 0]
The Path of best solution 978
The Value of MEB count 2000000
********** %s seconds********* 20.662853002548218
Do you want to run this program again? yes or no?
Enter number of cities
```

This is the output of randomized hill climbing - Simple Search

The cost of best Solution [3, 0, 2, 1]

The path of best solution 402 Value

of MEB count is 2

********** %s seconds********* 0.0009515285491943359

This is the output of randomized hill climbing - Simple Search

The cost of best Solution [1, 8, 13, 19, 24, 30, 36, 40, 41, 42, 44, 45, 47, 49, 48, 46, 43, 39, 33, 27, 22, 18, 15, 16, 20, 23, 26, 29, 32, 35, 38, 37, 34, 31, 28, 25, 21, 17, 14, 12, 11, 10, 9, 7, 6, 3, 4, 5, 2, 0]

The path of best solution 978

Value of MEB count is 2000000

********** %s seconds********* 20.66306781768799

End of Exercise.

Exercise 6. Reinforcement learning using python

Estimated time:

90.00 minutes

What this exercise is about:

Demonstrating how to implement a basic Reinforcement Learning algorithm - the Q-Learning technique.

What you should be able to do:

Demonstrate how to implement a basic Reinforcement Learning algorithm which is called the Q-Learning technique.

Introduction:

Reinforcement Learning is a type of Machine Learning paradigm in which a learning algorithm is trained not on preset data but rather based on a feedback system. These algorithms eliminate the cost of collecting and cleaning the data.

Here we attempt to teach a robot to reach its destination using the Q-learning technique.

Instructor exercise overview:

Step 1: Importing the required libraries.

Step 2: Defining and visualising the graph.

Step 3: Defining the reward the system for the robot.

Step 4: Defining some utility functions to be used in the training.

Step 5: Training and evaluating the bot using the Q-Matrix.

Python code:

```
import numpy as np

import pylab as pl

import networkx as nx

edges = [(0, 1), (1, 5), (5, 6), (5, 4), (1, 2),

            (1, 3), (9, 10), (2, 4), (0, 6), (6, 7),

            (8, 9), (7, 8), (1, 7), (3, 9)]

goal = 10

G = nx.Graph()

G.add_edges_from(edges)

pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos)

nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
pl.show()
MATRIX_SIZE = 11
M = np.matrix(np.ones(shape =(MATRIX_SIZE, MATRIX_SIZE)))
M *= -1
for point in edges:
      print(point)
      if point[1] == goal:
            M[point] = 100
      else:
            M[point] = 0

      if point[0] == goal:
            M[point[::-1]] = 100
      else:
            M[point[::-1]]= 0
            # reverse of point

M[goal, goal]= 100
print(M)
# add goal point round trip
# Q matrix
Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))

# Gamma (learning parameter).
gamma = 0.8

# Initial state. (Usually to be chosen at random)
initial_state = 1

# This function returns all available actions in the state given as an argument
```

```
def available_actions(state):
    current_state_row = M[state,]
    available_action = np.where(current_state_row >= 0)[1]
    return available_action

# Get available actions in the current state
available_action = available_actions(initial_state)

# This function chooses at random which action to be performed  within the range
# of all the available actions.
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_action,1))
    return next_action

# Sample next action to be performed
action = sample_next_action(available_action)

# This function updates the Q matrix according to the path selected and the  Q
# learning algorithm
def update(current_state, action, gamma):

    max_index = np.where(Q[action,] == np.max(Q[action,]))[1]

    if max_index.shape[0] > 1:
        max_index = int(np.random.choice(max_index, size = 1))
    else:
        max_index = int(max_index)
    max_value = Q[action, max_index]

    # Q learning formula
    Q[current_state, action] = M[current_state, action] + gamma  * max_value
    if (np.max(Q) > 0):
      return(np.sum(Q / np.max(Q)*100))
    else:
      return (0)
# Updates the Q-Matrix according to the path chosen


# Update Q matrix
update(initial_state,action,gamma)
#--------------------------------------------------------------------------------
-
# Training
scores = []
# Train over 100 iterations. (Re-iterate the process above).
for i in range(1000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_action = available_actions(current_state)
    action = sample_next_action(available_action)
    score = update(current_state,action,gamma)
    scores.append(score)
# Normalize the "trained" Q matrix
print("Trained Q matrix:")
print(Q/np.max(Q)*100)


#--------------------------------------------------------------------------------
-
# Testing
# Goal state = 5
# Best sequence path starting from 2 -> 2, 3, 1, 5


current_state = 0
steps = [current_state]
```

```
while current_state != 10:
   next_step_index = np.where(Q[current_state,] == np.max(Q[current_state,]))[1]

    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index,  size = 1))
    else:
        next_step_index = int(next_step_index)

    steps.append(next_step_index)
    current_state = next_step_index
# Print selected sequence of steps
print("Most efficient path:")
print(steps)
pl.plot(scores)
pl.xlabel('No of iterations')
pl.ylabel('Reward gained')
pl.show()
#------------------------------------------------------------------------------
#                                OUTPUT
#------------------------------------------------------------------------------
#
# Trained Q matrix:
#[[   0.     0.     0.     0.    80.     0. ]
# [   0.     0.     0.    64.     0.   100. ]
# [   0.     0.     0.    64.     0.     0. ]
# [   0.    80.    51.2    0.    80.     0. ]
# [   0.    80.    51.2    0.     0.   100. ]
# [   0.    80.     0.     0.    80.   100. ]]
#
# Selected path:
# [2, 3, 1, 5]#
```

Results:

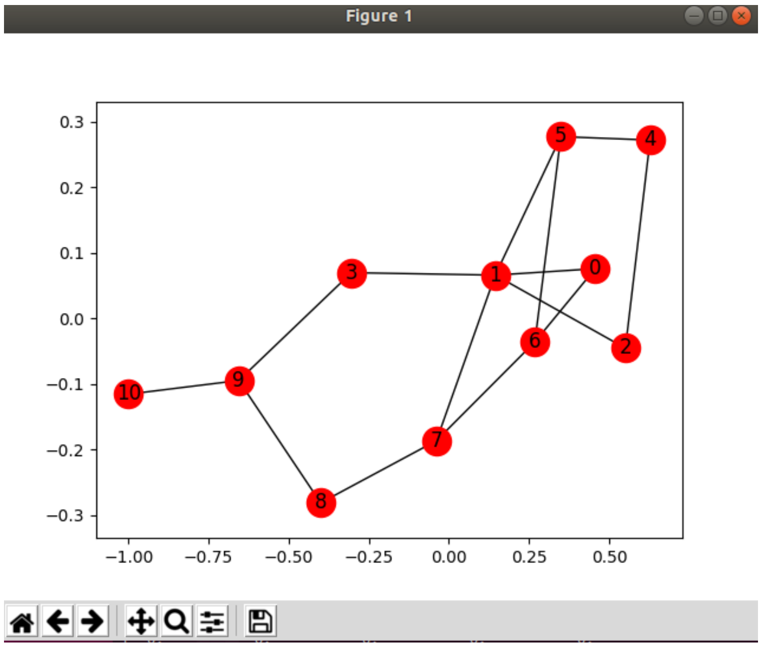
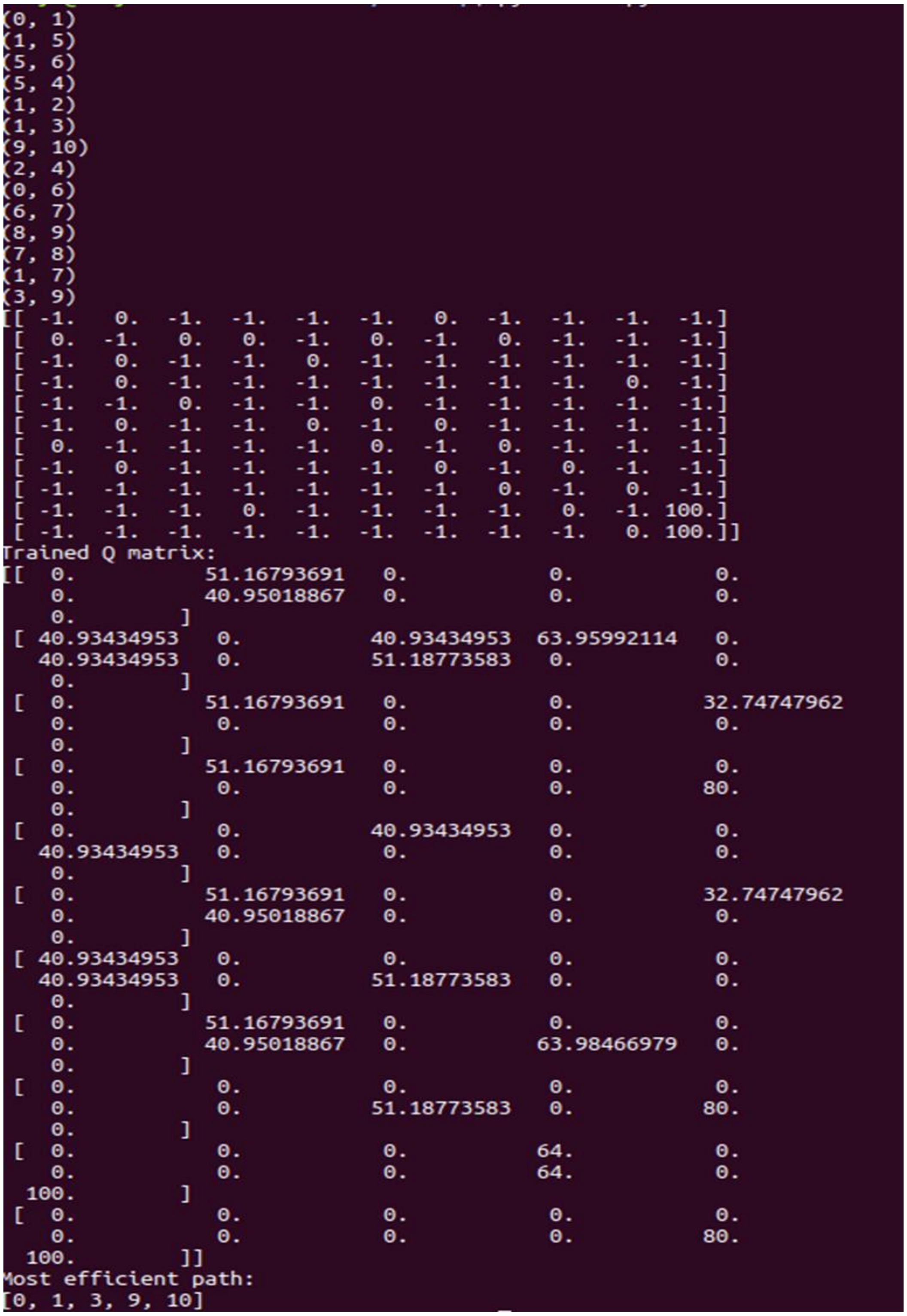
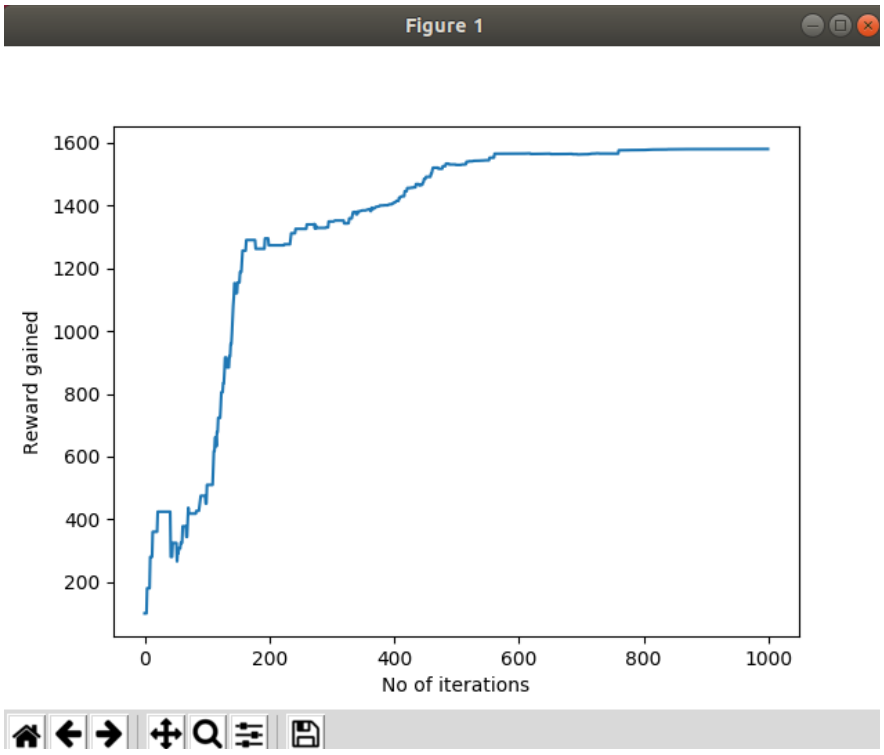


Figure 1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0
1
2
3
4
5
6
7
8
9
10



```
(0, 1)
(1, 5)
(5, 6)
(5, 4)
(1, 2)
(1, 3)
(9, 10)
(2, 4)
(0, 6)
(6, 7)
(8, 9)
(7, 8)
(1, 7)
(3, 9)
[[ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.   0.   0.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.]
 [ -1.  -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.]
 [ -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.   0.  -1. 100.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0. 100.]]
Trained Q matrix:
[[  0.          51.16793691   0.           0.           0.
    0.          40.95018867   0.           0.           0.
    0.        ]
 [ 40.93434953   0.          40.93434953  63.95992114   0.
   40.93434953   0.          51.18773583   0.           0.
    0.        ]
 [  0.          51.16793691   0.           0.          32.74747962
    0.           0.           0.           0.           0.
    0.        ]
 [  0.          51.16793691   0.           0.           0.
    0.           0.           0.           0.          80.
    0.        ]
 [  0.           0.          40.93434953   0.           0.
   40.93434953   0.           0.           0.           0.
    0.        ]
 [  0.          51.16793691   0.           0.          32.74747962
    0.          40.95018867   0.           0.           0.
    0.        ]
 [ 40.93434953   0.           0.           0.           0.
   40.93434953   0.          51.18773583   0.           0.
    0.        ]
 [  0.          51.16793691   0.           0.           0.
    0.          40.95018867   0.          63.98466979   0.
    0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.          51.18773583   0.          80.
    0.        ]
 [  0.           0.           0.          64.           0.
    0.           0.           0.          64.           0.
  100.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.          80.
  100.        ]]
Most efficient path:
[0, 1, 3, 9, 10]
```



End of Exercise.

Exercise 7. Simple Neural Network concept using python

Estimated time:

90.00 minutes

What this exercise is about:

Simulating a basic neural network.

What you should be able to do:

Simulate a basic neural network.

Introduction:

Here is a table that shows the problem.

	Input			Output
Training data 1	0	0	1	0
Training data 2	1	1	1	1
Training data 3	1	0	1	1
Training data 4	0	1	1	0
New Situation	1	0	0	**?**

We are going to train the neural network such that it can predict the correct output value when provided with a new set of data.

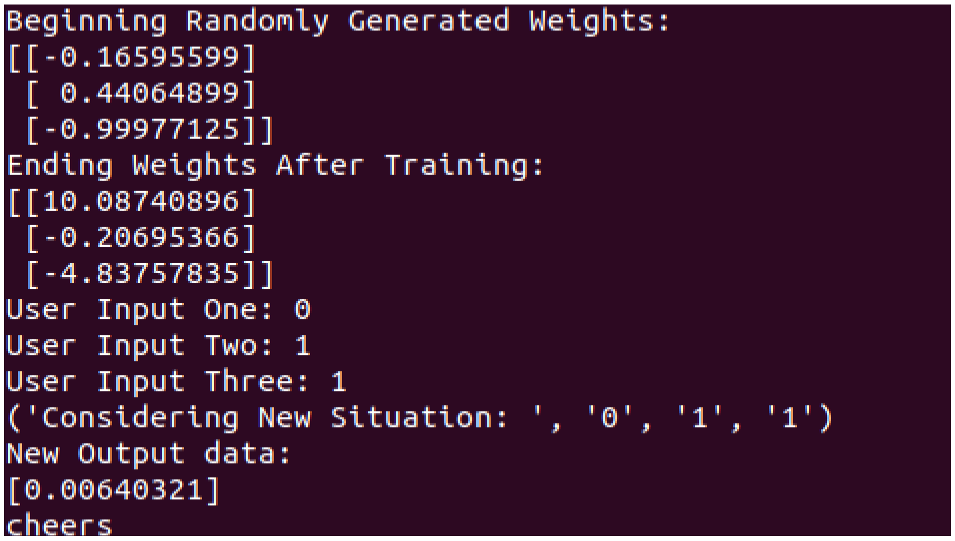
As you can see on the table, the value of the output is always equal to the first value in the input section. Therefore, we expect the value of the output (**?**) to be 1.

Instructor exercise overview:

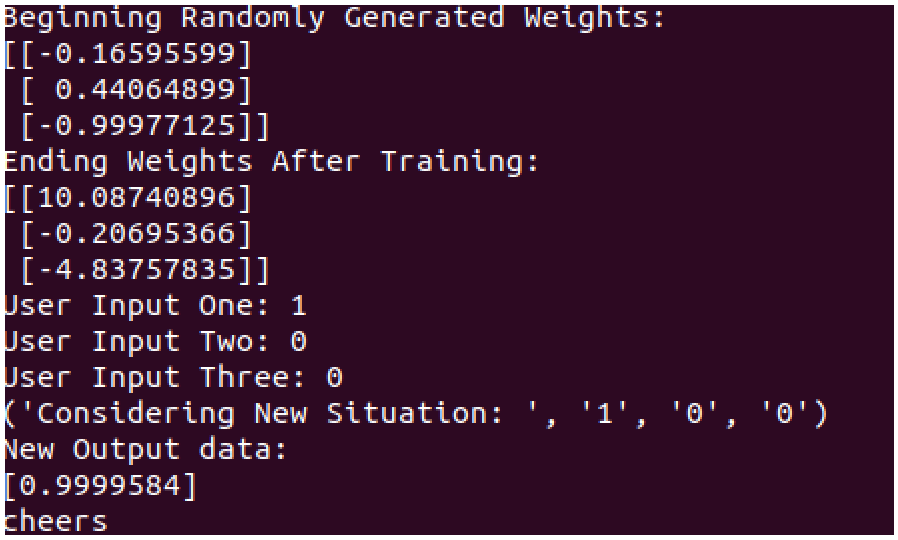
Python code:

```
import numpy as np


class NeuralNetwork():


    def __init  (self):
        # seeding for random number generation
        np.random.seed(1)


        #converting weights to a 3 by 1 matrix with values from -1 to 1 and
mean of 0
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1
    def sigmoid(self, x):
        #applying the sigmoid function
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        #computing derivative to the Sigmoid function
        return x * (1 - x)

    def train(self, training_inputs, training_outputs, training_iterations):

        #training the model to make accurate predictions while  adjusting
weights continually
        for iteration in range(training_iterations):
            #siphon the training data via  the neuron
            output = self.think(training_inputs)

            #computing error rate for back-propagation
            error = training_outputs - output

            #performing weight adjustments
            adjustments = np.dot(training_inputs.T, error  *
self.sigmoid_derivative(output))

            self.synaptic_weights += adjustments

    def think(self, inputs):
        #passing the inputs via the neuron to get output
        #converting values to floats

        inputs = inputs.astype(float)
        output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
        return output
if   name___== "  main  ":

    #initializing the neuron class
    neural_network = NeuralNetwork()

    print("Beginning Randomly Generated Weights: ")
    print(neural_network.synaptic_weights)

    #training data consisting of 4 examples--3 input values and 1 output
    training_inputs = np.array([[0,0,1],
                                [1,1,1],
                                [1,0,1],
                                [0,1,1]])

    training_outputs = np.array([[0,1,1,0]]).T

    #training taking place
    neural_network.train(training_inputs, training_outputs, 15000)

    print("Ending Weights After Training: ")
    print(neural_network.synaptic_weights)

    user_input_one = str(input("User Input One: "))
    user_input_two = str(input("User Input Two: "))
    user_input_three = str(input("User Input Three: "))

    print("Considering New Situation: ", user_input_one, user_input_two,
user_input_three)
    print("New Output data: ")
    print(neural_network.think(np.array([user_input_one, user_input_two,
user_input_three])))
    print("cheers")
```

Results:



```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
User Input One: 0
User Input Two: 1
User Input Three: 1
('Considering New Situation: ', '0', '1', '1')
New Output data:
[0.00640321]
cheers
```



```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
User Input One: 1
User Input Two: 0
User Input Three: 0
('Considering New Situation: ', '1', '0', '0')
New Output data:
[0.9999584]
cheers
```

End of Exercise.

Exercise 8. Compare various learning strategies for MLP classifier

Estimated time:

90.00 minutes

What this exercise is about:

Comparing various learning strategies for MLP classifier for various datasets.

What you should be able to do:

Compare various learning strategies for MLP classifier for various datasets.

Introduction:

This example visualizes some training loss curves for different stochastic learning strategies, including SGD and Adam. Because of time-constraints, we use several small datasets, for which L-BFGS might be more suitable. The general trend shown in these examples seems to carry over to large datasets, however.

Note that those results can be highly dependent on the value of `learning_rate_init`.

Instructor exercise overview:

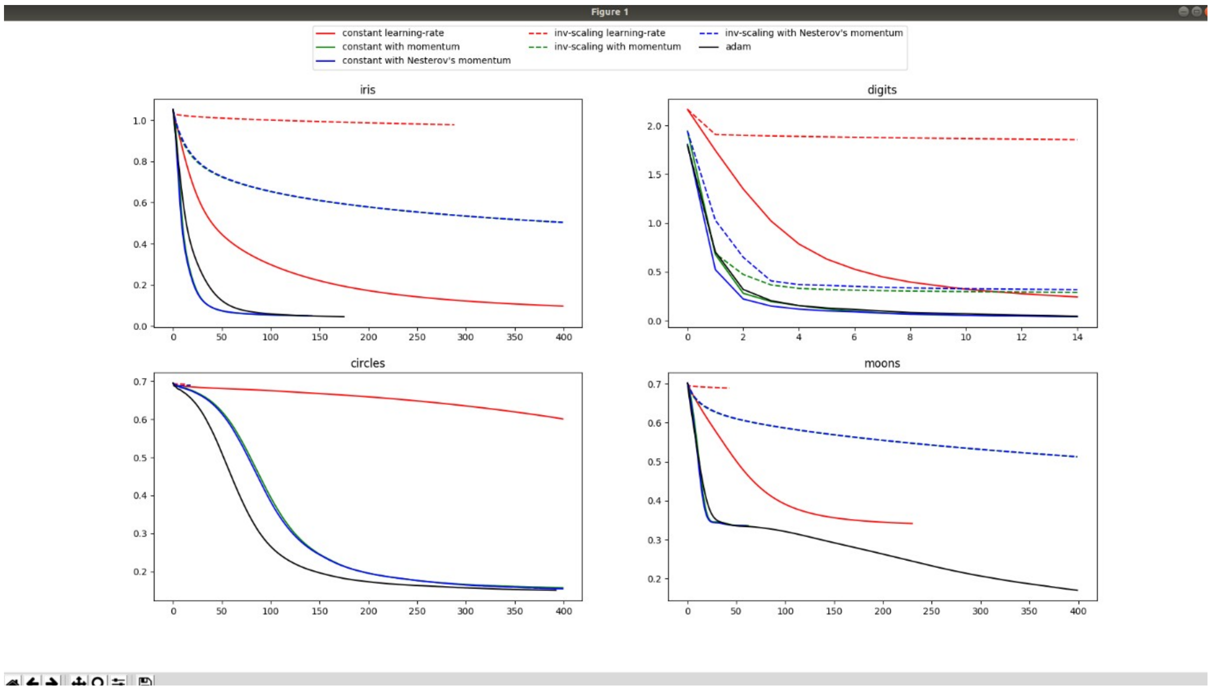
Python code:

```
"""
========================================================
Compare Stochastic learning strategies for MLPClassifier
========================================================

This example visualizes some training loss curves for different stochastic
learning strategies, including SGD and Adam. Because of time-constraints, we
use several small datasets, for which L-BFGS might be more suitable. The
general trend shown in these examples seems to carry over to larger datasets,
however.

Note that those results can be highly dependent on the value of
``learning_rate_init``.
"""

print(  doc  )

import warnings
import matplotlib.pyplot as plt

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn import datasets
from sklearn.exceptions import ConvergenceWarning

# different learning rate schedules and momentum parameters
params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum':  0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum':  .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum':  .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum':  0,
           'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum':  .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum':  .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2},
          {'solver': 'adam', 'learning_rate_init': 0.01}]

labels = ["constant learning-rate", "constant with momentum",
          "constant with Nesterov's momentum",
          "inv-scaling learning-rate", "inv-scaling with momentum",
          "inv-scaling with Nesterov's momentum", "adam"]

plot_args = [{'c': 'red', 'linestyle': '-'},
             {'c': 'green', 'linestyle': '-'},
             {'c': 'blue', 'linestyle': '-'},
             {'c': 'red', 'linestyle': '--'},
             {'c': 'green', 'linestyle': '--'},
             {'c': 'blue', 'linestyle': '--'},
             {'c': 'black', 'linestyle': '-'}]
```

```
def plot_on_dataset(X, y, ax, name):
    # for each dataset, plot learning for each learning  strategy
    print("\nlearning on dataset %s" % name)
    ax.set_title(name)

    X = MinMaxScaler().fit_transform(X)
    mlps = []
    if name == "digits":
        # digits is larger but converges fairly quickly
        max_iter = 15
    else:
        max_iter = 400

    for label, param in zip(labels, params):
        print("training: %s" % label)
        mlp = MLPClassifier(random_state=0,
                            max_iter=max_iter, **param)

        # some parameter combinations will not converge  as can be seen on the
        # plots so they are ignored here
        with warnings.catch_warnings():
            warnings.filterwarnings("ignore", category=ConvergenceWarning,
                                    module="sklearn")
            mlp.fit(X, y)

        mlps.append(mlp)
        print("Training set  score: %f" % mlp.score(X, y))
        print("Training set  loss: %f" % mlp.loss_)
    for mlp, label, args in zip(mlps, labels, plot_args):
        ax.plot(mlp.loss_curve_, label=label, **args)


fig, axes = plt.subplots(2, 2, figsize=(15, 10))
```

```
# load / generate some toy datasets
iris = datasets.load_iris()
X_digits, y_digits = datasets.load_digits(return_X_y=True)
data_sets = [(iris.data, iris.target),
             (X_digits, y_digits),
             datasets.make_circles(noise=0.2, factor=0.5, random_state=1),
             datasets.make_moons(noise=0.3, random_state=0)]

for ax, data, name in zip(axes.ravel(), data_sets, ['iris', 'digits',
                                                    'circles', 'moons']):
    plot_on_dataset(*data, ax=ax, name=name)

fig.legend(ax.get_lines(), labels, ncol=3, loc="upper center")
plt.show()
```

Result:



End of Exercise.

Exercise 9. Kalman Filtering using python

Estimated time:

60.00 minutes

What this exercise is about:

Implementing a Linear Kalman Filter.

What you should be able to do:

Implement a Linear Kalman Filter.

Introduction:

- Kalman Filters are discrete. That is, they rely on measurement samples taken between repeated but constant periods of time. Although you can approximate it fairly well, you don't know what happens between the samples.
- Kalman Filters are recursive. This means its prediction of the future relies on the state of the present (position, velocity, acceleration, etc) as well as a guess about what any controllable parts tried to do to affect the situation (such as a rudder or steering differential).
- Kalman Filters work by making a prediction of the future, getting a measurement from reality, comparing the two, moderating this difference, and adjusting its estimate with this moderated value.
- The more you understand the mathematical model of your situation, the more accurate the Kalman filter's results will be.
- If your model is completely consistent with what's actually happening, the Kalman filter's estimate will eventually converge with what's actually happening.

When you start up a Kalman Filter, these are the things it expects:

- The mathematical model of the system, represented by matrices A, B, and H.
- An initial estimate about the complete state of the system, given as a vector x.
- An initial estimate about the error of the system, given as a matrix P.
- Estimates about the general process and measurement error of the system, represented by matrices Q and R.

During each time step, you are expected to give it the following information:

- A vector containing the most current control state (vector "u"). This is the system's guess as to what it did to affect the situation (such as steering commands).
- A vector containing the most current measurements that can be used to calculate the state (vector "z").

After the calculations, you get the following information:

- The most current estimate of the true state of the system.
- The most current estimate of the overall error of the system.

The Equations

NOTE: The equations are here for exposition and reference. You aren't expected to understand the equations on the first read.

The Kalman Filter is like a function in a programming language: it's a process of sequential equations with inputs, constants, and outputs. Here I've color-coded the filter equations to illustrate which parts are which. If you are using the Kalman Filter like a black box, you can ignore the gray intermediary variables.

BLUE = inputs ORANGE = outputs BLACK = constants GRAY = intermediary variables

Information: State Prediction **Information:** (Predict where we're going to be)	**Information:** $\mathbf{x}_{predicted} = \mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{u}_{n}$
Information: Covariance Prediction **Information:** (Predict how much error)	**Information:** $\mathbf{P}_{predicted} = \mathbf{A}\mathbf{P}_{n-1}\mathbf{A}^{\mathbf{T}} +$
Information: Innovation **Information:** (Compare reality against prediction)	**Information:** $\tilde{\mathbf{y}} = \mathbf{z}_n - \mathbf{H}\mathbf{x}_{predicted}$
Information: Innovation Covariance **Information:** (Compare real error against prediction)	**Information:** $\mathbf{S} = \mathbf{H}\mathbf{P}_{predicted}\mathbf{H}^{\mathbf{T}} + \mathbf{R}$
Information: Kalman Gain **Information:** (Moderate the prediction)	**Information:** $\mathbf{K} = \mathbf{P}_{predicted}\mathbf{H}^{\mathbf{T}}\mathbf{S}^{-1}$
Information: State Update	**Information:** $\mathbf{x}_n = \mathbf{x}_{predicted} + \mathbf{K}\tilde{\mathbf{y}}$

Information: (New estimate of where we are)	
Information: Covariance Update **Information:** (New estimate of error)	**Information:** $\mathbf{P}_n = (I - \mathbf{K}\mathbf{H})\mathbf{P}_{predicted}$

Inputs:

Un = Control vector. This indicates the magnitude of any control system's or user's control on the situation.

Zn = Measurement vector. This contains the real-world measurement we received in this time step.

Outputs:

Xn = Newest estimate of the current "true" state.

Pn = Newest estimate of the average error for each part of the state.

Constants:

A = State transition matrix. Basically, multiply state by this and add control factors, and you get a prediction of the state for the next time step.

B = Control matrix. This is used to define linear equations for any control factors.

H = Observation matrix. Multiply a state vector by H to translate it to a measurement vector.

Q = Estimated process error covariance. Finding precise values for Q and R are beyond the scope of this guide.

R = Estimated measurement error covariance. Finding precise values for Q and R are beyond the scope of this guide.

To program a Kalman Filter class, your constructor should have all the constant matrices, as well as an initial estimate of the state (x) and error (P). The step function should have the inputs (measurement and control vectors). In my version, you access outputs through "getter" functions.

We will take an example of Single- Variable

Situation

We will attempt to measure a constant DC voltage with a noisy voltmeter. We will use the Kalman filter to filter out the noise and converge toward the true value.

The state transition equation:

$$V_n = V_{n-1} + w_n$$

Vn = The current voltage.

Vn-1 = The voltage last time.

Wn = Random noise (measurement error).

Since the voltage never changes, it is a very simple equation. The objective of the Kalman filter is to mitigate the influence of Wn in this equation.

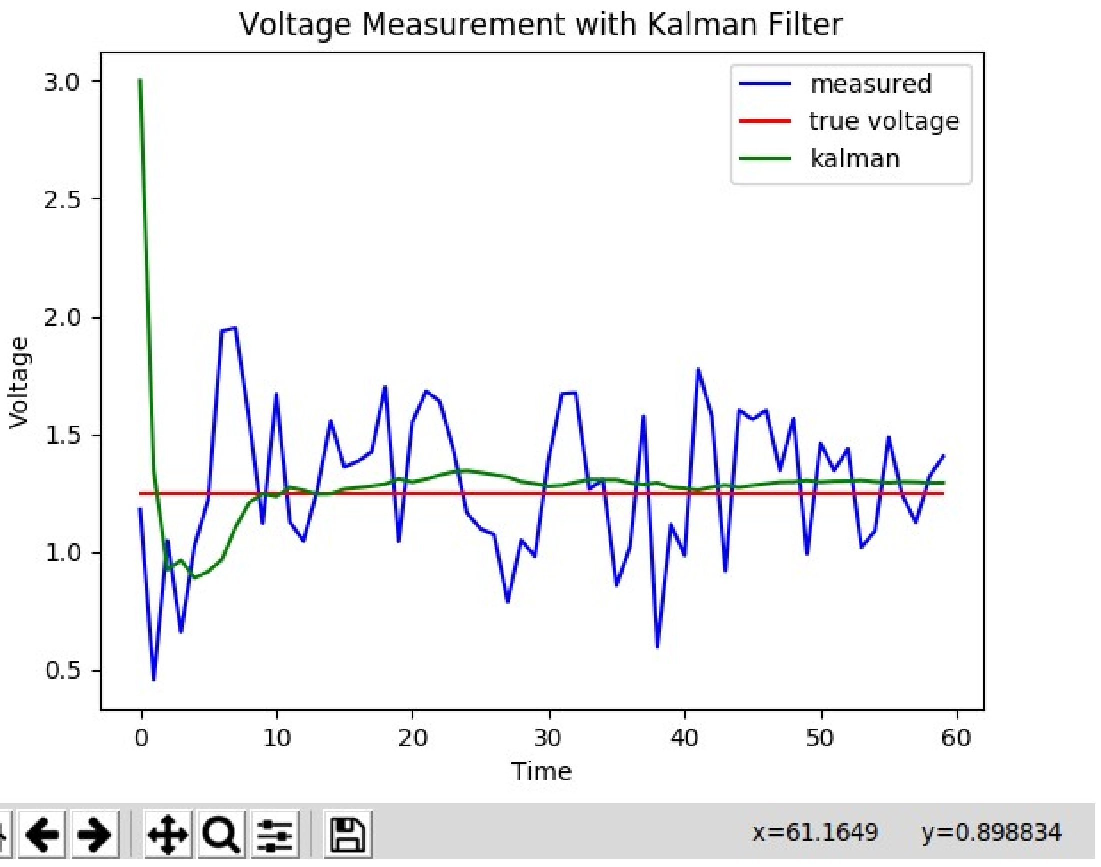
Instructor exercise overview:

Python code:

```
# kalman1.py
#
# Implements a single-variable linear Kalman filter.
#
# Note: This code is part of a larger tutorial "Kalman Filters for  Undergrads"
# located at http://greg.czerniak.info/node/5.
import random
import numpy
import pylab


# Implements a linear Kalman filter.
class KalmanFilterLinear:
  def   init  (self,_A, _B, _H, _x, _P, _Q, _R):
    self.A = _A                       # State transition matrix.
    self.B = _B                       # Control matrix.
    self.H = _H                       # Observation matrix.
    self.current_state_estimate = _x # Initial state estimate.
    self.current_prob_estimate = _P  # Initial covariance estimate.
    self.Q = _Q                       # Estimated error in process.
    self.R = _R                       # Estimated error in measurements.
  def GetCurrentState(self):

    return self.current_state_estimate

  def Step(self,control_vector,measurement_vector):

    #---------------------------Prediction step-----------------------------

    predicted_state_estimate = self.A * self.current_state_estimate + self.B  *
control_vector

    predicted_prob_estimate   =   (self.A   *   self.current_prob_estimate)   *
numpy.transpose(self.A) + self.Q

    #--------------------------Observation step-----------------------------

    innovation = measurement_vector - self.H*predicted_state_estimate

    innovation_covariance                                                     =
self.H*predicted_prob_estimate*numpy.transpose(self.H) + self.R

    #-----------------------------Update step------------------------------

    kalman_gain   =   predicted_prob_estimate   *   numpy.transpose(self.H)   *
numpy.linalg.inv(innovation_covariance)

    self.current_state_estimate  =  predicted_state_estimate  +  kalman_gain  *
innovation

    # We need the size of the matrix so we can make  an identity matrix.

    size = self.current_prob_estimate.shape[0]

    # eye(n) = nxn identity matrix.

    self.current_prob_estimate               =                (numpy.eye(size)-
kalman_gain*self.H)*predicted_prob_estimate


class Voltmeter:

  def   init  (self,_truevoltage,_noiselevel):

    self.truevoltage = _truevoltage

    self.noiselevel = _noiselevel

  def GetVoltage(self):
    return self.truevoltage
  def GetVoltageWithNoise(self):
    return random.gauss(self.GetVoltage(),self.noiselevel)

numsteps = 60

A = numpy.matrix([1])
H = numpy.matrix([1])
B = numpy.matrix([0])
Q = numpy.matrix([0.00001])
R = numpy.matrix([0.1])
xhat = numpy.matrix([3])
P    = numpy.matrix([1])

filter = KalmanFilterLinear(A,B,H,xhat,P,Q,R)
voltmeter = Voltmeter(1.25,0.25)

measuredvoltage = []
truevoltage = []
kalman = []

for i in range(numsteps):
    measured = voltmeter.GetVoltageWithNoise()
    measuredvoltage.append(measured)
    truevoltage.append(voltmeter.GetVoltage())
    kalman.append(filter.GetCurrentState()[0,0])
    filter.Step(numpy.matrix([0]),numpy.matrix([measured]))
```

```
pylab.plot(range(numsteps),measuredvoltage,'b',range(numsteps),truevoltage,'r',
range(numsteps),kalman,'g')

pylab.xlabel('Time')

pylab.ylabel('Voltage')

pylab.title('Voltage Measurement with Kalman Filter')

pylab.legend(('measured','true voltage','kalman'))

pylab.show()
```

Result:



End of Exercise.

Exercise 10. Installing ROS and other packages, basic programs

Estimated time:

90.00 minutes

What this exercise is about:

Installing ROS and other packages, basic programs.

What you should be able to do:

Install ROS and other packages, basic programs.

Instructor exercise overview:

Step1 : Installation of ROS and Dependencies:

1.1 Setting up source list :

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list'
```

1.2 Setting up keys :

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'  --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

1.3 Installation :

```
sudo apt update

sudo apt install ros-melodic-desktop
```

1.4 Initalize rosdep:

```
sudo rosdep init

rosdep update
```

1.5 Environment setup

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

1.6 Dependencies for building packages

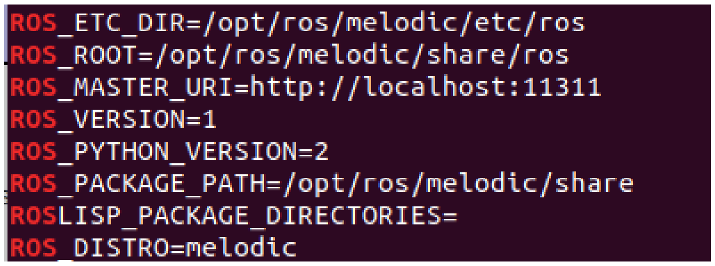
```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

Step 2: Managing the Environment

Check the environment settings with the command

```
printenv | grep ROS
```

the path settings should be shown like :



```
ROS_ETC_DIR=/opt/ros/melodic/etc/ros
ROS_ROOT=/opt/ros/melodic/share/ros
ROS_MASTER_URI=http://localhost:11311
ROS_VERSION=1
ROS_PYTHON_VERSION=2
ROS_PACKAGE_PATH=/opt/ros/melodic/share
ROSLISP_PACKAGE_DIRECTORIES=
ROS_DISTRO=melodic
```

Step 3: Creating a ROS workspace

```
mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

source devel/setup.bash

echo $ROS_PACKAGE_PATH , the you shoudl be able to see:

    /home/youruser/catkin_ws/src:/opt/ros/melodic/share
```

Step 4: Filesystem Tools

a) using rospack : allows to get information about packages

example : rospack find roscpp

which should return

YOUR_INSTALL_PATH/share/roscpp

b) using roscd: allows to change directory to a package or stack

```
example : roscd roscpp
then pwd should show
YOUR_INSTALL_PATH/share/roscpp
```

c)using rosls: allows to ls directly in a package rather than absolute path

example: rosls roscpp_tutorials

should return

```
cmake launch package.xml  srv
```

Step 5: ROS nodes :

A node really isn't much more than an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

a) Run the command roscore

we should get the summary

```
logging to ~/.ros/log/f52ddc1a-13fa-11ea-b314-9cb6d012a619/roslaunch-
machine_name-29249.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://machine_name:33919/
ros_comm version 1.14.3

SUMMARY
======

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES

auto-starting new master
process[master]: started with pid [13054]
ROS_MASTER_URI=http://machine_name:11311/

setting /run_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf
process[rosout-1]: started with pid [13067]
started core service [/rosout]
```

b) using rosnode:

Open up a **new terminal**, and let's use **rosnode** to see what running `roscore` did... Bear in mind to keep the previous terminal open either by opening a new tab or simply minimizing it.

Key in the commnad rosnode list

we should see

```
/rosout
```

This showed us that there is only one node running: rosout. This is always running as it collects and logs nodes' debugging output.

Lets try to get some info by the command   rosnode info /rosout we should see

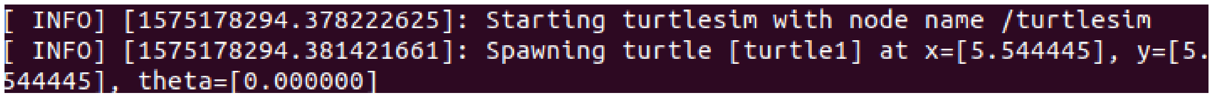
```
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level

contacting node http://machine_name:54614/ ...
Pid: 29270
```

c) using rosrun:

`rosrun` allows you to use the package name to directly run a node within a package (without having to know the package path).

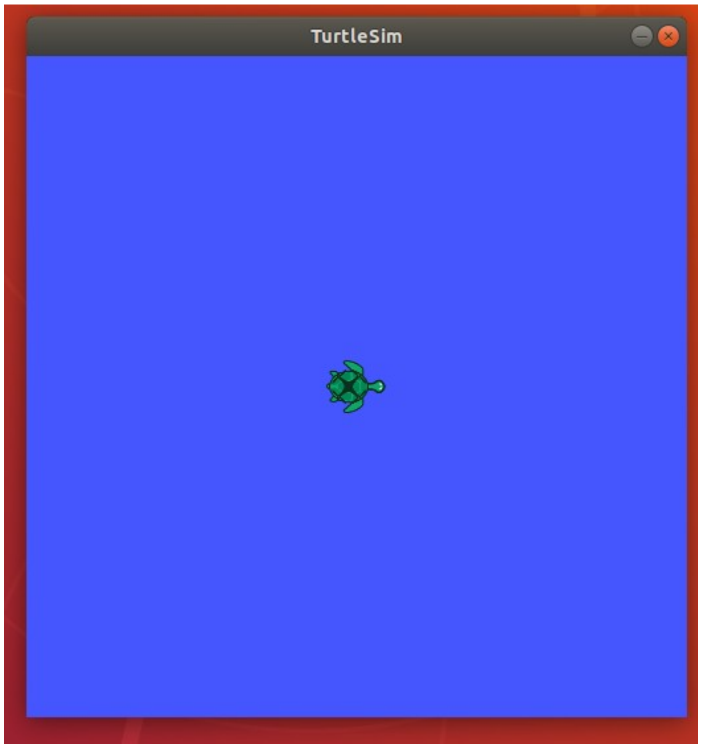
Run the command : rosrun turtlesim turtlesim_node

we should have output in two terminals as below:



```
[ INFO] [1575178294.378222625]: Starting turtlesim with node name /turtlesim
[ INFO] [1575178294.381421661]: Spawning turtle [turtle1] at x=[5.544445], y=[5.
544445], theta=[0.000000]
```

and



Now if we open another terminal and see the output for the command rosnode list

we will see:

```
/rosout
/turtlesim
```

now lets use a remapping argument to change the nodes name as:

```
rosrun  turtlesim  turtlesim_node   name:=my_turtle
```

we should see:

```
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://your home:42831/ time=0.482082ms
xmlrpc reply from http://your home:42831/ time=0.992060ms
xmlrpc reply from http://your home:42831/ time=0.972986ms
```

6. Understanding ROS Topics

a) Let's start by making sure that we have roscore running, in a new terminal:

roscore

If you left roscore running , you may get the error message:

- roscore cannot run as another roscore/master is already running.
  Please kill other roscore/master processes before relaunching

This is fine. Only one roscore needs to be running.

b) run **in a new terminal**:

rosrun turtlesim turtlesim_node

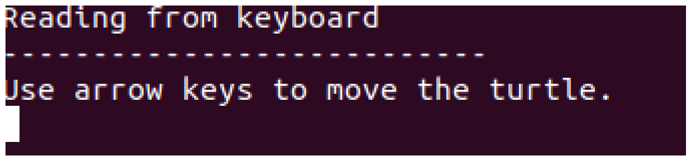
now lets try to control our turtle using

c) turtle keyboard teleoperation

We'll also need something to drive the turtle around with. Please run **in a new terminal**:

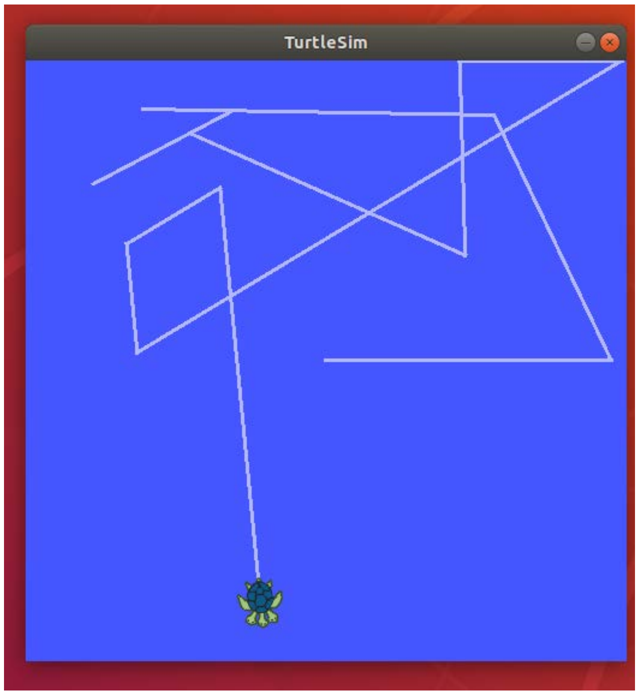
rosrun turtlesim turtle_teleop_key

we should get the following outputs

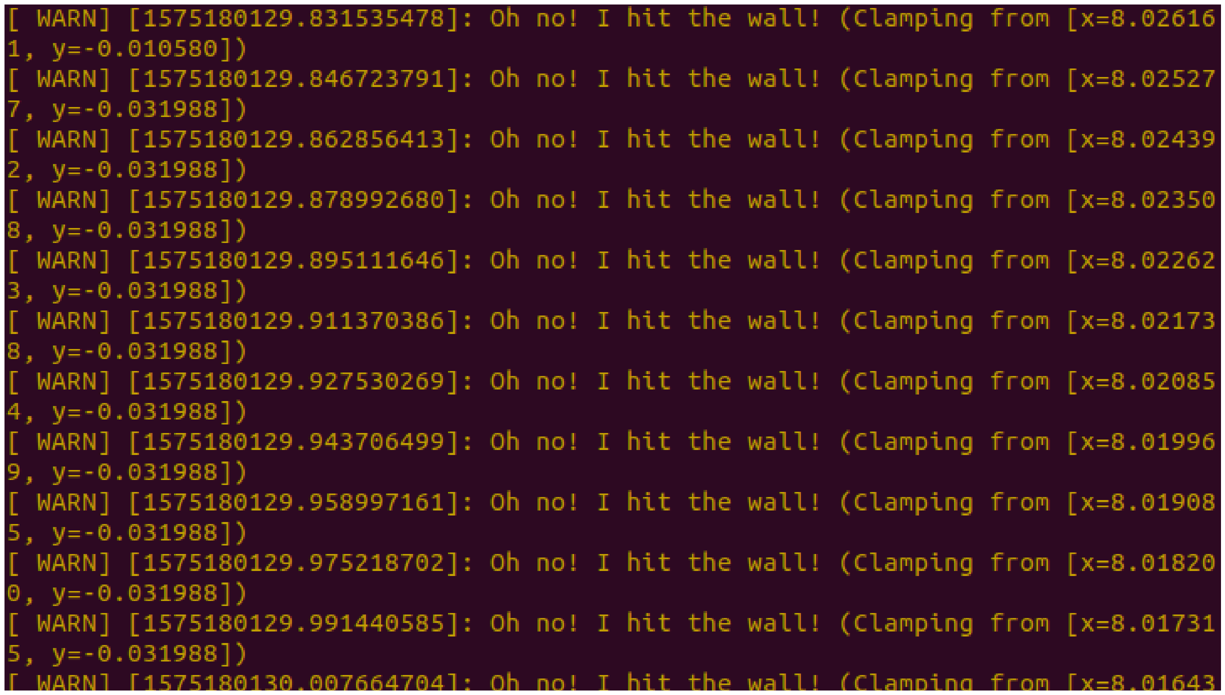


```
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

and



when the turtle hits the walls we also see this output

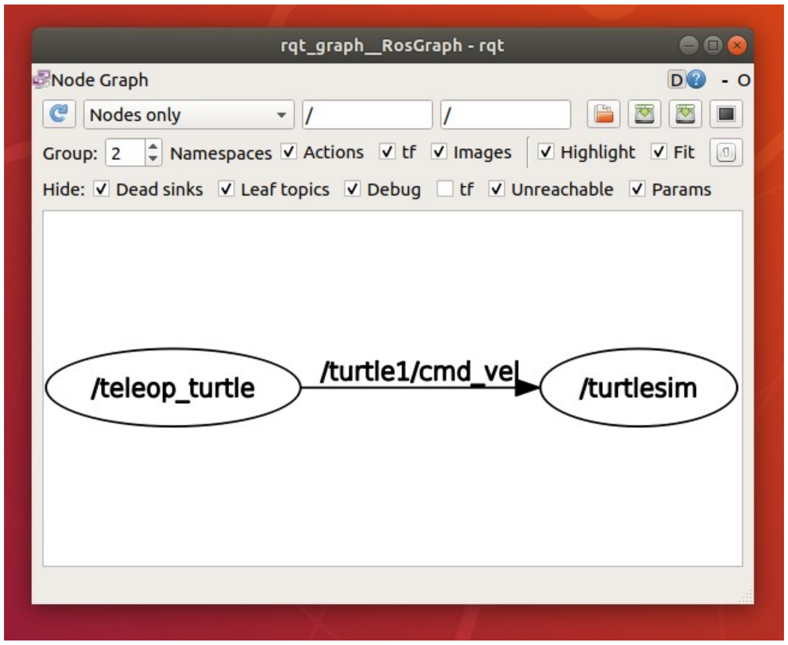


```
[ WARN] [1575180129.831535478]: Oh no! I hit the wall! (Clamping from [x=8.02616
1, y=-0.010580])
[ WARN] [1575180129.846723791]: Oh no! I hit the wall! (Clamping from [x=8.02527
7, y=-0.031988])
[ WARN] [1575180129.862856413]: Oh no! I hit the wall! (Clamping from [x=8.02439
2, y=-0.031988])
[ WARN] [1575180129.878992680]: Oh no! I hit the wall! (Clamping from [x=8.02350
8, y=-0.031988])
[ WARN] [1575180129.895111646]: Oh no! I hit the wall! (Clamping from [x=8.02262
3, y=-0.031988])
[ WARN] [1575180129.911370386]: Oh no! I hit the wall! (Clamping from [x=8.02173
8, y=-0.031988])
[ WARN] [1575180129.927530269]: Oh no! I hit the wall! (Clamping from [x=8.02085
4, y=-0.031988])
[ WARN] [1575180129.943706499]: Oh no! I hit the wall! (Clamping from [x=8.01996
9, y=-0.031988])
[ WARN] [1575180129.958997161]: Oh no! I hit the wall! (Clamping from [x=8.01908
5, y=-0.031988])
[ WARN] [1575180129.975218702]: Oh no! I hit the wall! (Clamping from [x=8.01820
0, y=-0.031988])
[ WARN] [1575180129.991440585]: Oh no! I hit the wall! (Clamping from [x=8.01731
5, y=-0.031988])
[ WARN] [1575180130.007664704]: Oh no! I hit the wall! (Clamping from [x=8.01643
```

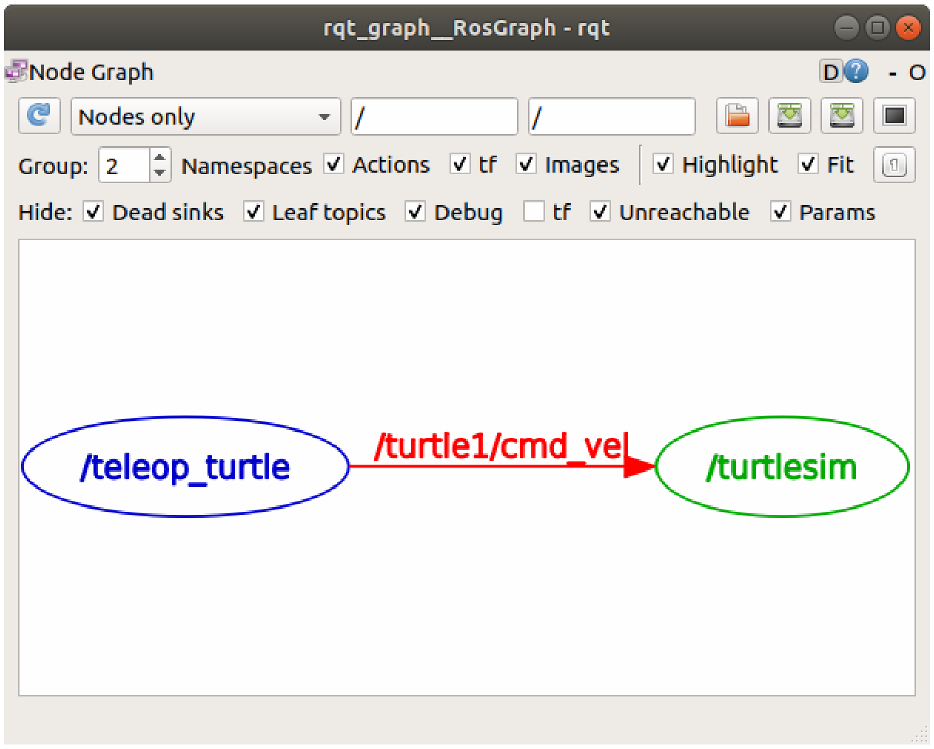
now lets try to see the communication bewtween the various nodes

d) ROS topics

In the new terminal run rosrun rqt_graph rqt_graph we should see this

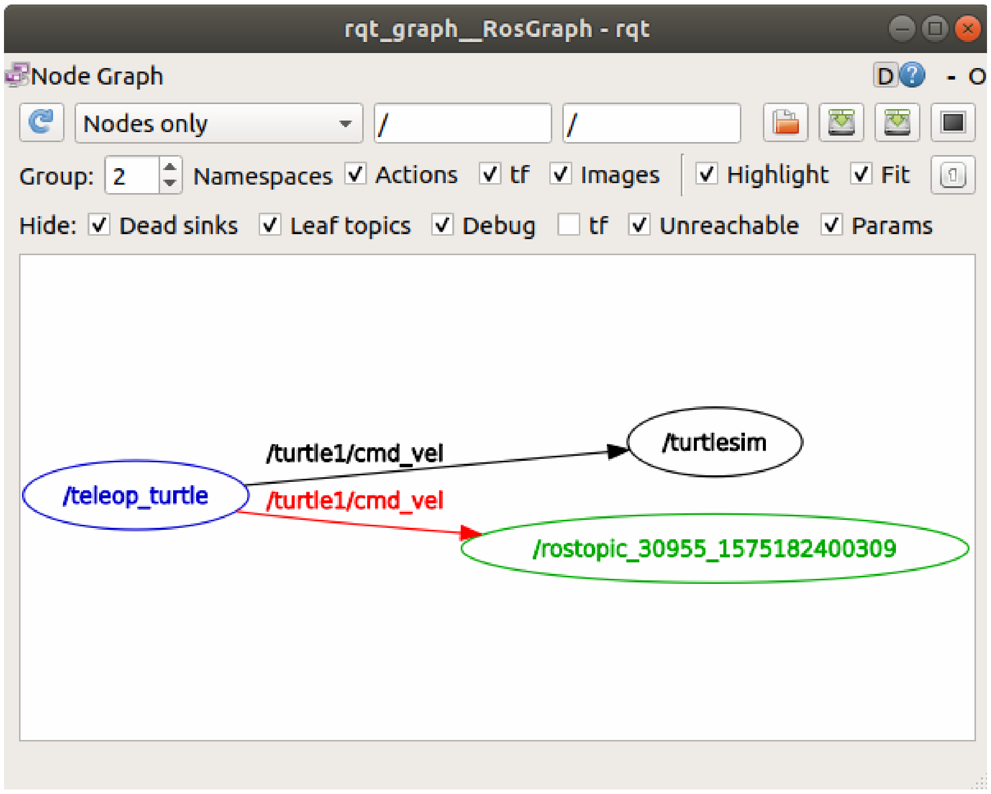

rqt_graph__RosGraph - rqt
Node Graph
Nodes only
Group: 2
Namespaces ✓ Actions ✓ tf ✓ Images ✓ Highlight ✓ Fit
Hide: ✓ Dead sinks ✓ Leaf topics ✓ Debug tf ✓ Unreachable ✓ Params
/teleop_turtle
/turtle1/cmd_vel
/turtlesim

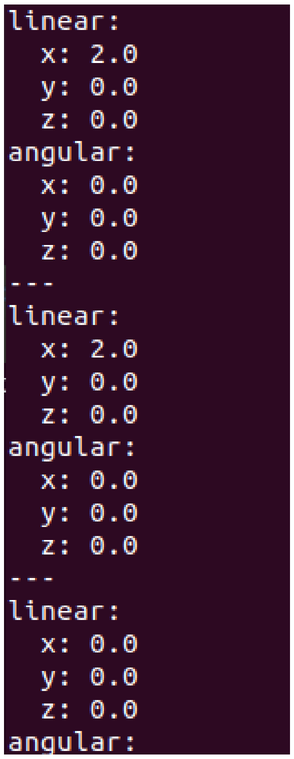
If we hover our mouse over /turtle1/cmd_vel it will highlight the ROS nodes (here blue and green) and topics (here red). As you can see, the turtlesim_node and the turtle_teleop_key nodes are communicating on the topic named /turtle1/cmd_vel.



We use the command rostopic echo /turtle1/cmd_vel

and we get the data being published as ros topic





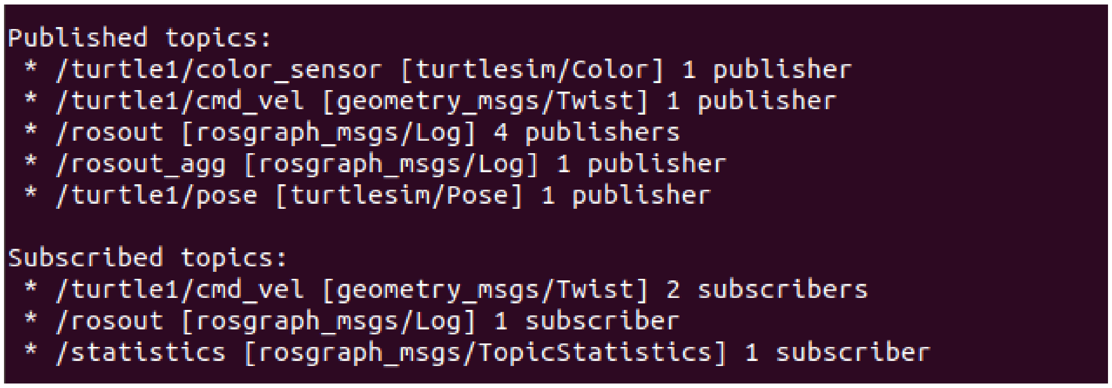
```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
```

Now let's look at rqt_graph again. Press the refresh button in the upper-left to show the new node. As you can see rostopic echo, shown here in red, is now also **subscribed** to the turtle1/cmd_vel topic.

e) rostopic list

This tell us the whole list of topics : the published and the subscribed

Use the command   rostopic list -v we should see the following :



```
Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
 * /rosout [rosgraph_msgs/Log] 4 publishers
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/cmd_vel [geometry_msgs/Twist] 2 subscribers
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /statistics [rosgraph_msgs/TopicStatistics] 1 subscriber
```

f) ROS Messages

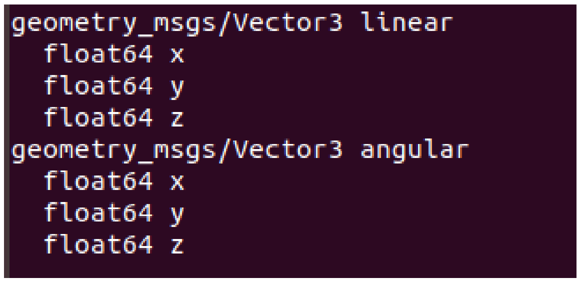
Communication on topics happens by sending ROS **messages** between nodes. For the publisher (turtle_teleop_key) and subscriber (turtlesim_node) to communicate, the publisher and subscriber must send and receive the same **type** of message. This means that a topic **type** is defined by the message **type** published on it. The **type** of the message sent on a topic can be determined using rostopic type.

Enter the command

rostopic type /turtle1/cmd_vel
we should get

geometry_msgs/Twist
We can look at the details of the message using rosmsg:

rosmsg show geometry_msgs/Twist



```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

also

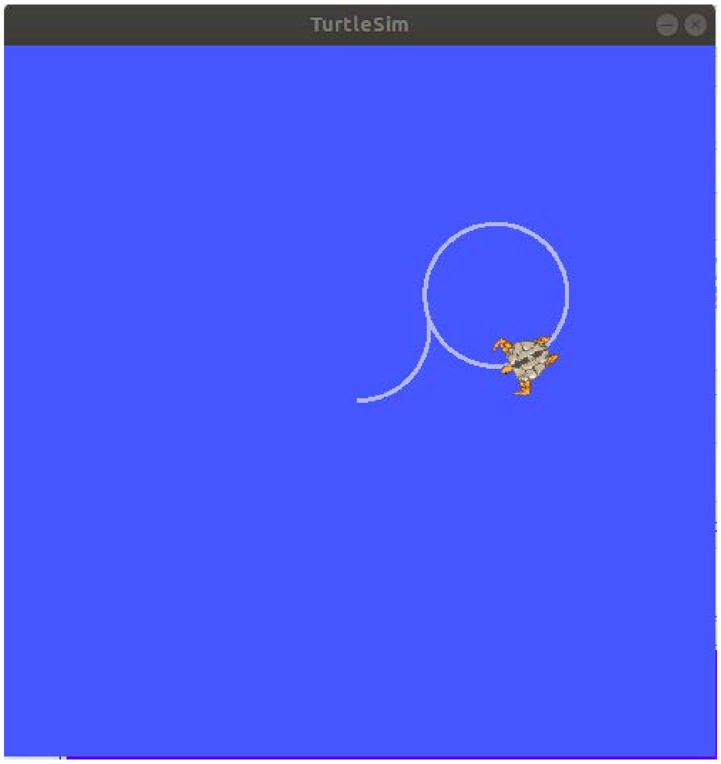
rostopic pub publishes data on to a topic currently advertised.

For example enter

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

This command will send a single message to turtlesim telling it to move with a linear velocity of 2.0, and an angular velocity of 1.8 .

You may have noticed that the turtle has stopped moving; this is because the turtle requires a steady stream of commands at 1 Hz to keep moving. We can publish a steady stream of commands using rostopic pub-r command: `rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]`



We can also look at what is happening in rqt_graph. Press the refresh button in the upper-left. The rostopic pub node (here in red) is communicating with the rostopic echo node (here in green):

now we can use the command to echo the details as

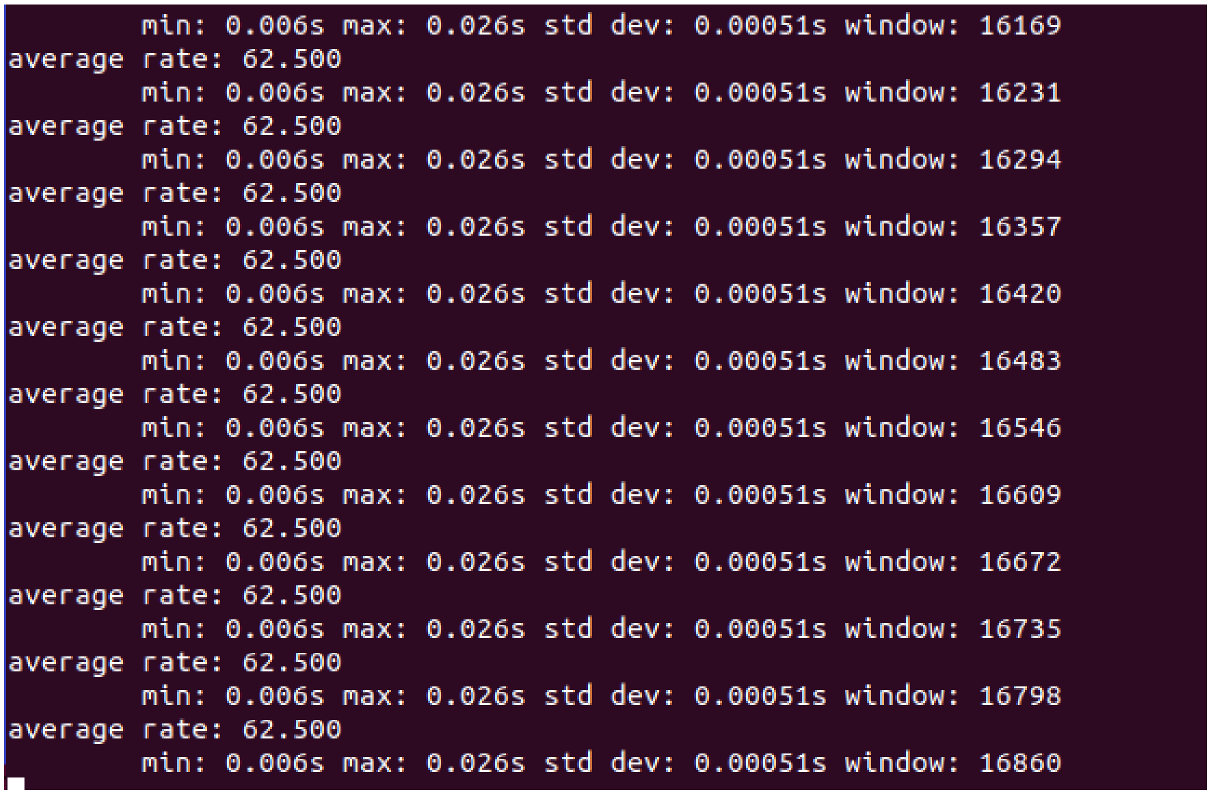
```
rostopic echo /turtle1/pose
```

we can also see how fast the turtlesim_node is publishing /turtle1/pose:

use the command

```
rostopic hz /turtle1/pose
```

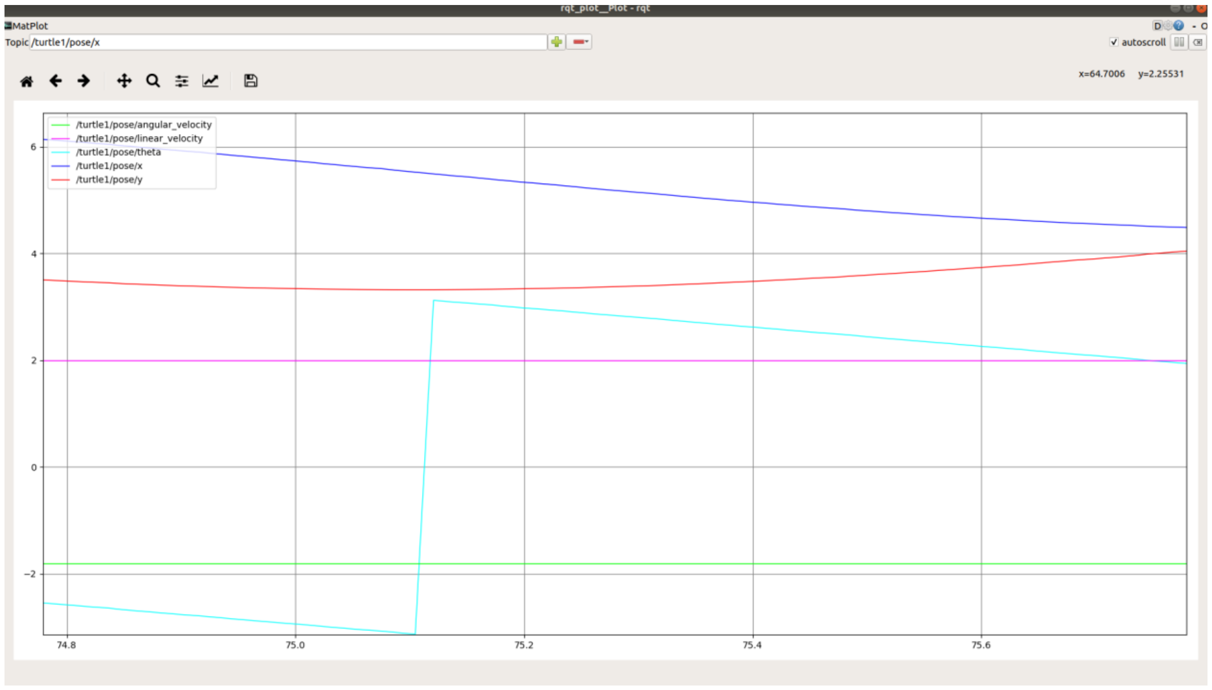
we should be able to see



```
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16169
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16231
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16294
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16357
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16420
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16483
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16546
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16609
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16672
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16735
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16798
average rate: 62.500
        min: 0.006s max: 0.026s std dev: 0.00051s window: 16860
```

we can use the rqt_plot to display a scrolling time plot of the data published on topics as

```
rosrun rqt_plot rqt_plot
```



End of Exercise.

Exercise 11. Testing the Simulator

Estimated time:

60.00 minutes

What this exercise is about:

Testing the simulator.

What you should be able to do:

Open world models and spawn robot models into the simulated environment.

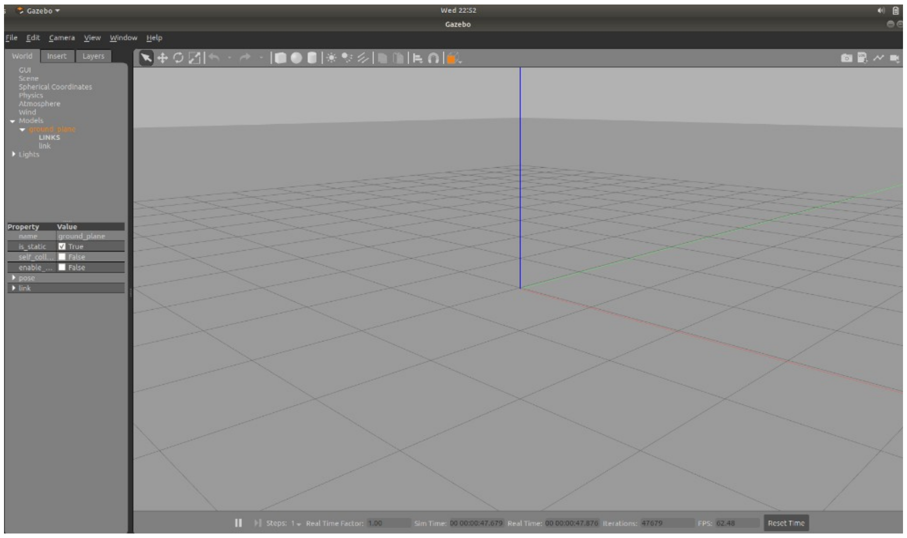
Introduction:

There are many ways to start Gazebo, open world models and spawn robot models into the simulated environment. In this experiment we cover the ROS-way of doing things:

Instructor exercise overview:

using rosrun and roslaunch.

```
a) Using roslaunch to Open World Models

   use the command

    roslaunch gazebo_ros empty_world.launch
```

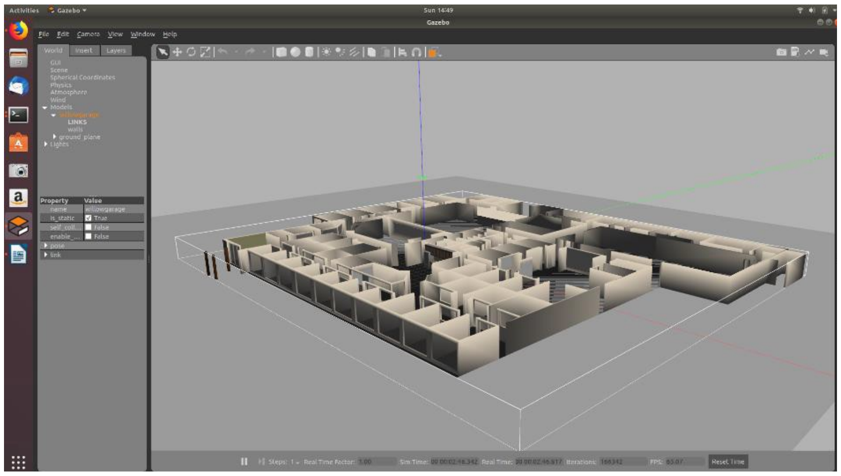


Property	Value
name	ground_plane
is_static	✓ True
self_coll...	False
enable_...	False
pose	
link	

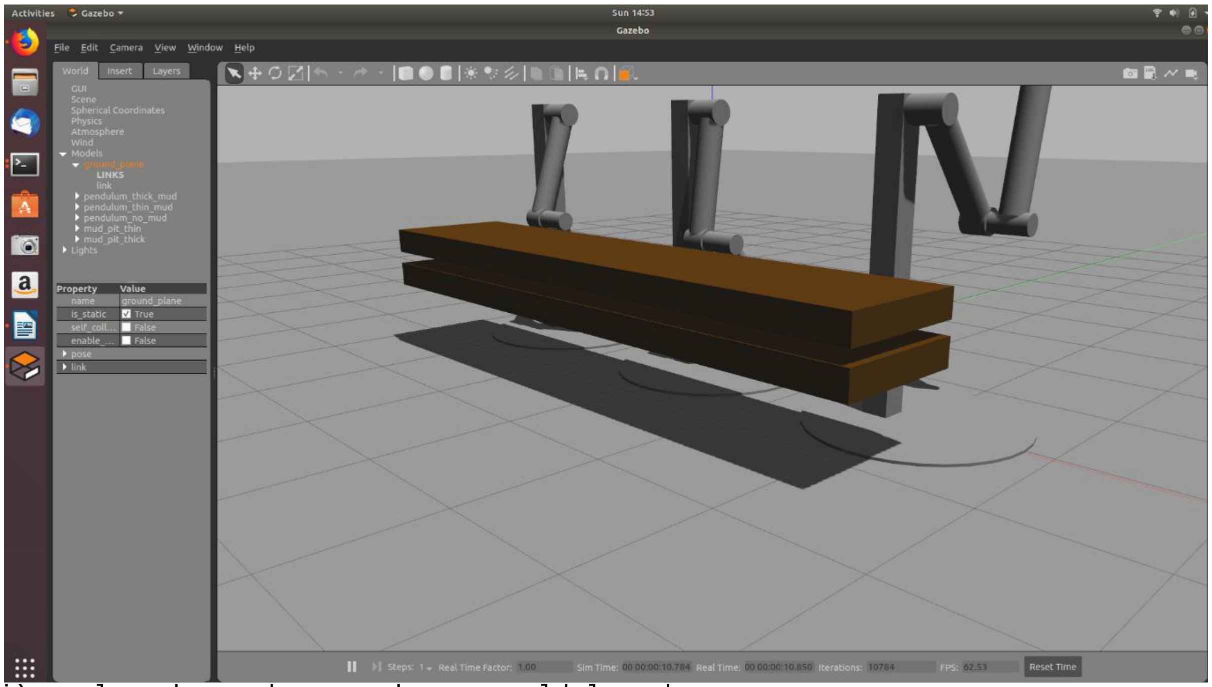
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:47.679 Real Time: 00 00:00:47.876 Iterations: 47679 FPS: 62.48

use the command to launch other demo worlds like

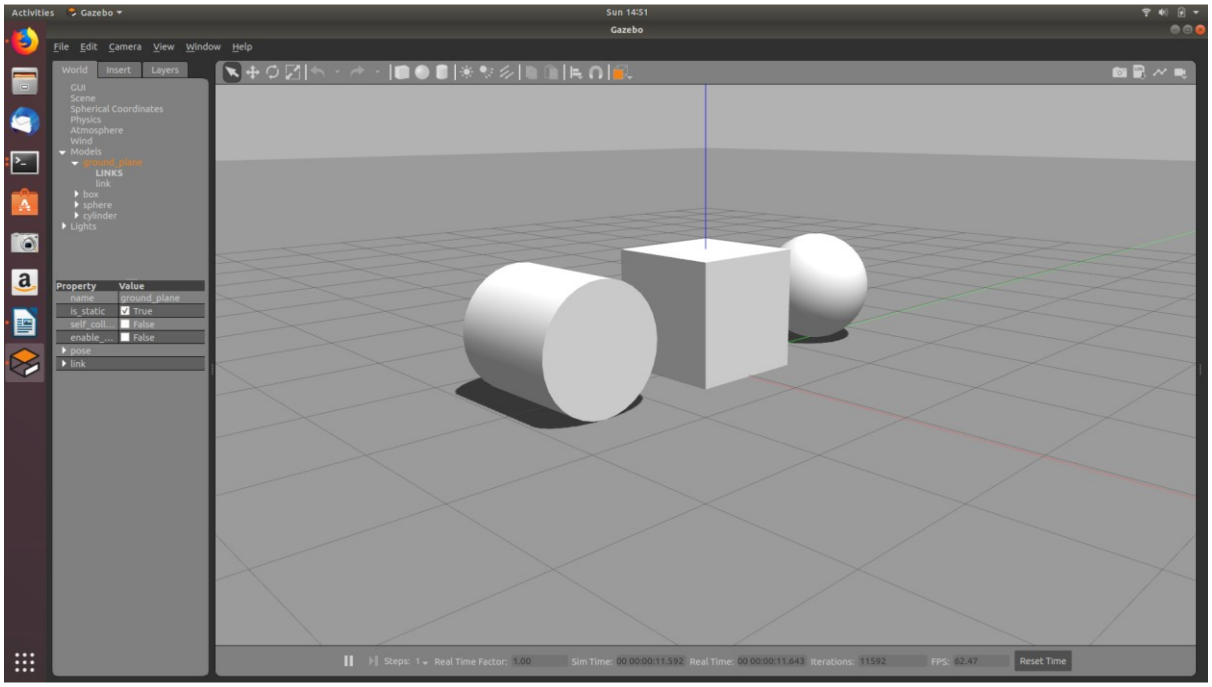
(i) roslaunch gazebo_ros willowgarage_world.launch



ii) roslaunch gazebo_ros mud_world.launch



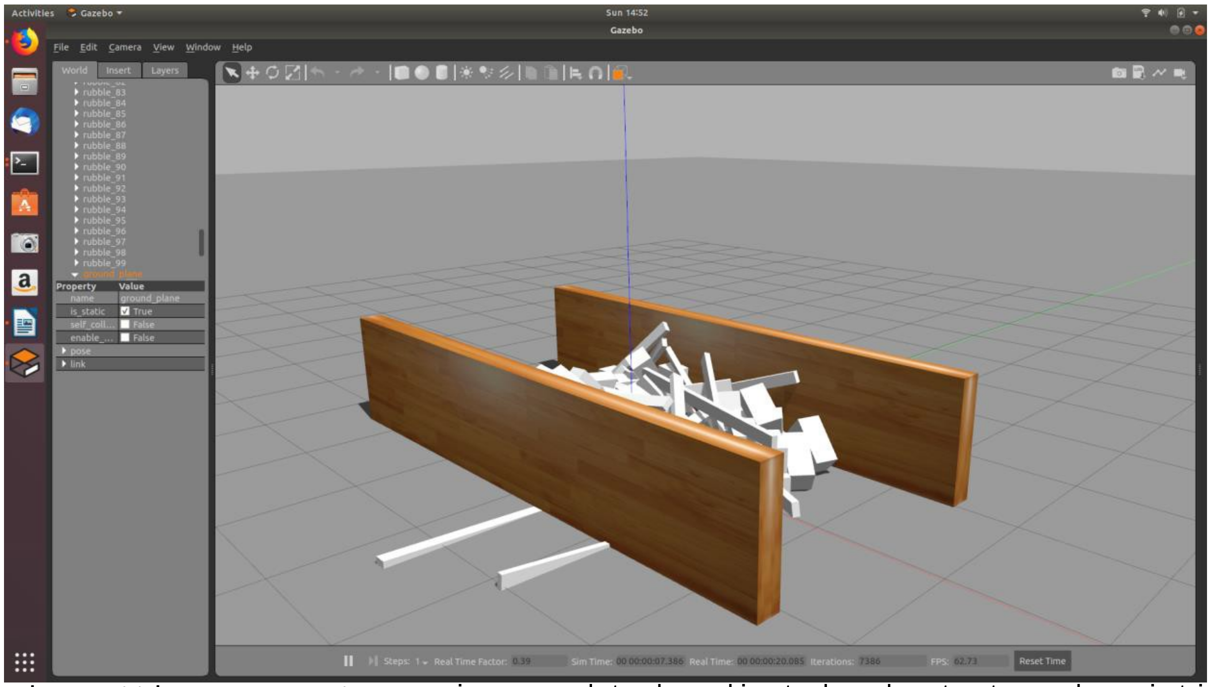
iii) roslaunch gazebo_ros shapes_world.launch



Property	Value
name	ground_plane
is_static	✓ True
self_coll...	False
enable_...	False

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:11.592 Real Time: 00 00:00:11.643 Iterations: 11592 FPS: 62.47

```
iv) roslaunch gazebo_ros rubble_world.launch
```



```
Variou settings, parameters, views need to be adjsuted and outputs and variations
needs to be recorded.
```

End of Exercise.

Exercise 12. Monitoring Robot motion using Simulator

Estimated time:

60.00 minutes

What this exercise is about:

Monitoring robot motion using simulator

What you should be able to do:

Monitor robot motion using simulator

Instructor exercise overview:

In this lab, we will launch a virtual robot called TurtleBot3 in the Rviz simulator.

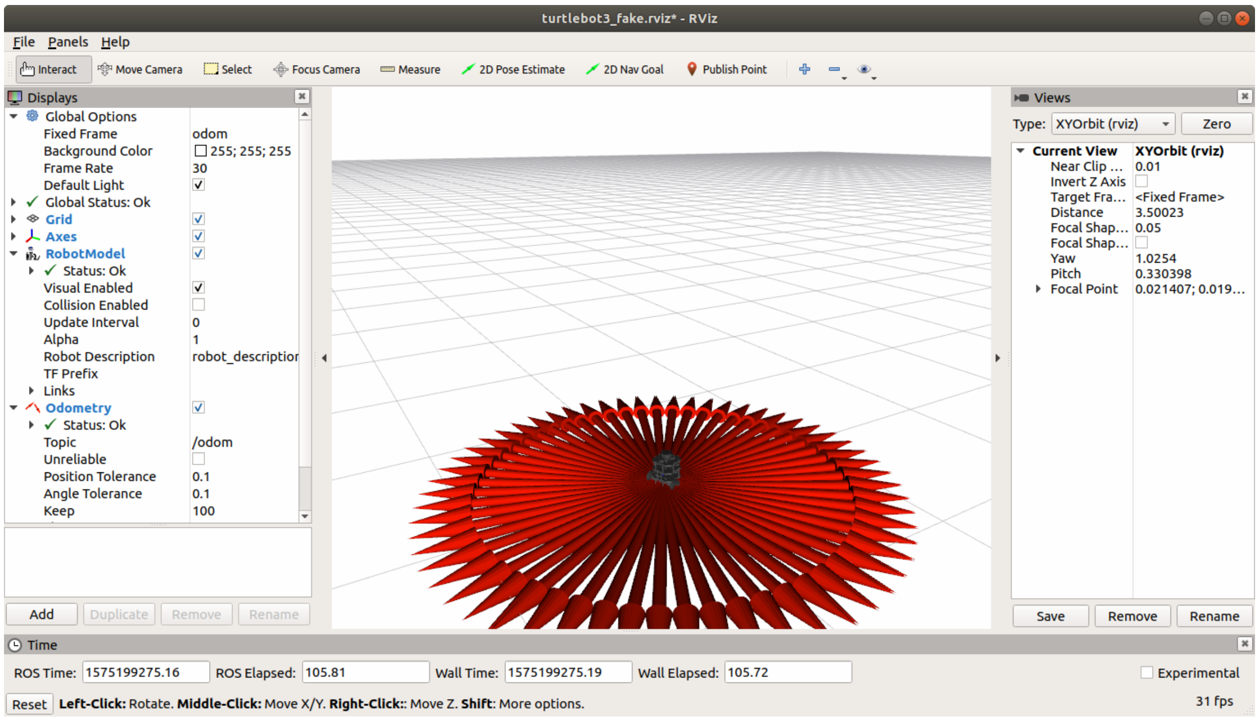
Use the command to launch a fake robot

```
roslaunch turtlebot3_fake turtlebot3_fake.launch
```

To move TurtleBot3 around the screen, open a new terminal

window, and type the following command:

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```



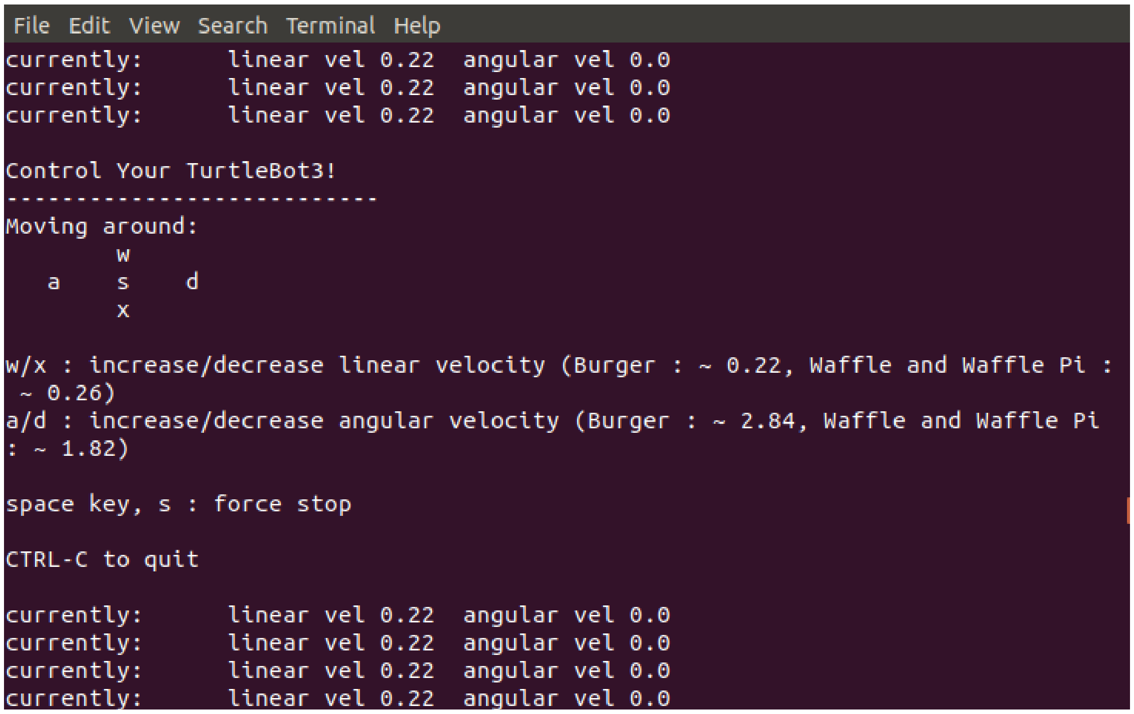
Global Options	
Fixed Frame	odom
Background Color	255; 255; 255
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
Axes	✓
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
Odometry	✓
Status: Ok	
Topic	/odom
Unreliable	
Position Tolerance	0.1
Angle Tolerance	0.1
Keep	100

Type: XYOrbit (rviz) Zero

Current View	XYOrbit (rviz)
Near Clip ...	0.01
Invert Z Axis	
Target Fra...	<Fixed Frame>
Distance	3.50023
Focal Shap...	0.05
Focal Shap...	
Yaw	1.0254
Pitch	0.330398
Focal Point	0.021407; 0.019...

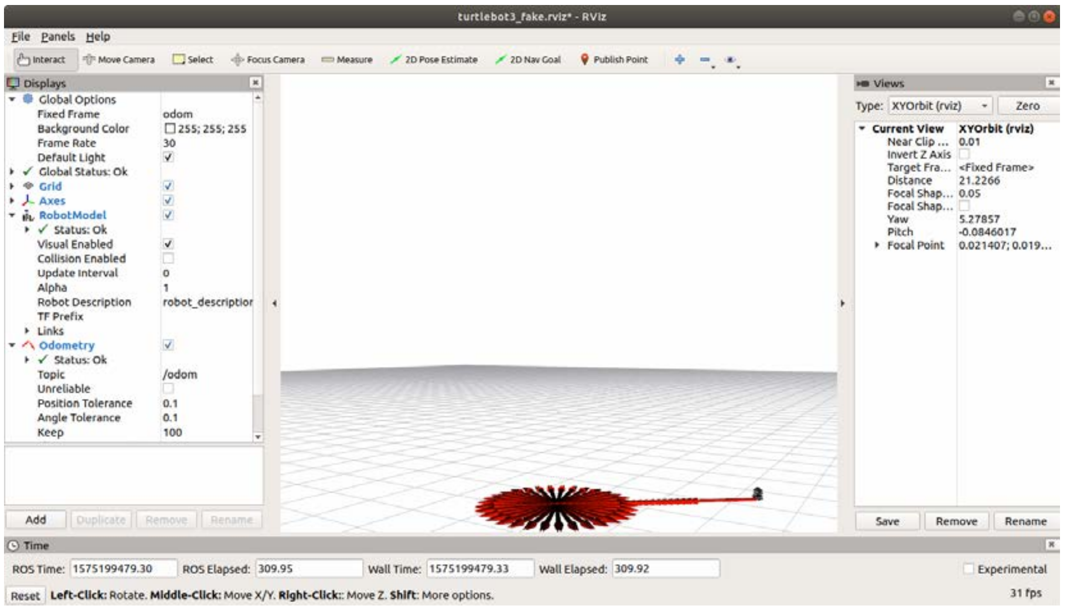
ROS Time: 1575199275.16 ROS Elapsed: 105.81 Wall Time: 1575199275.19 Wall Elapsed: 105.72 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options. 31 fps



```
File  Edit  View  Search  Terminal  Help
currently:      linear vel 0.22  angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi :
 ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi
: ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear vel 0.22  angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0
```

We then use the keyboard to move the robot around, this also is teleoperating the robot (Rviz).



Global Options	
Fixed Frame	odom
Background Color	255; 255; 255
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
Axes	✓
RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	1
Robot Description	robot_descriptior
TF Prefix	
Links	
Odometry	✓
Status: Ok	
Topic	/odom
Unreliable	☐
Position Tolerance	0.1
Angle Tolerance	0.1
Keep	100

Type: XYOrbit (rviz)

Current View	XYOrbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	21.2266
Focal Shap...	0.05
Focal Shap...	☐
Yaw	5.27857
Pitch	-0.0846017
Focal Point	0.021407; 0.019...

ROS Time: 1575199479.30 ROS Elapsed: 309.95 Wall Time: 1575199479.33 Wall Elapsed: 309.92 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options. 31 fps

End of Exercise.

Exercise 13. Teleoperating the simulated Robot

Estimated time:

60.00 minutes

What this exercise is about:

Gazebo implementing the TurtleBot3 simulation.

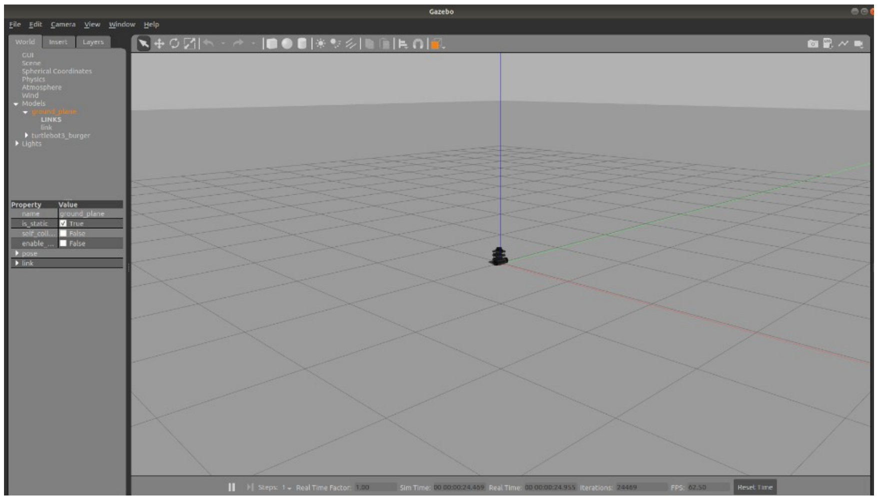
What you should be able to do:

Use Gazebo to implement the TurtleBot3 simulation.

Instructor exercise overview:

First, let's launch TurtleBot3 in an empty environment. Type this command:

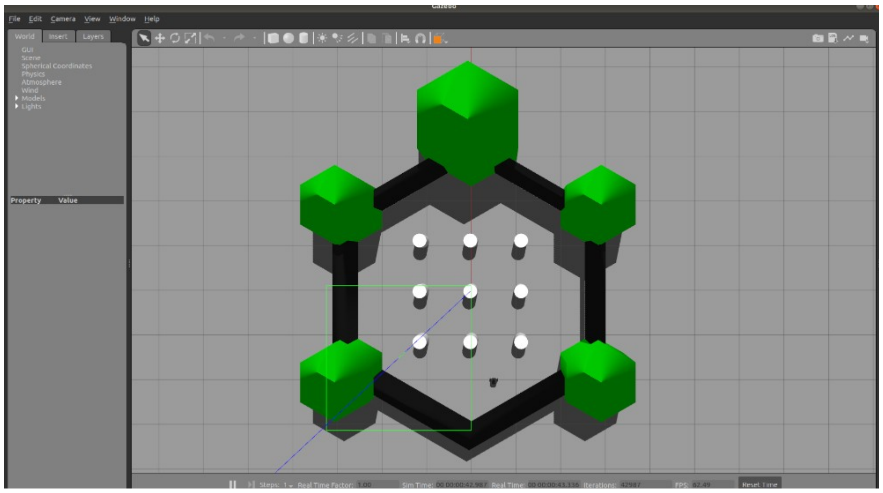
```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```



Let's look at our TurtleBot3 in a different environment. This environment is often used for testing SLAM and navigation algorithms. Simultaneous localization and mapping (SLAM) concerns the problem of a robot building or updating a map of an unknown environment while simultaneously keeping track its location in that environment.

In a new window type the command:

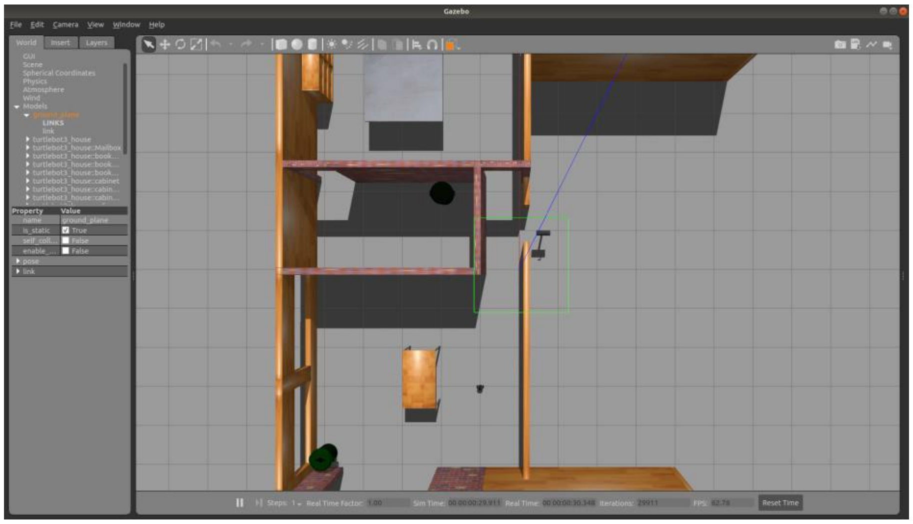
```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```



Press CTRL+C and close out all windows.

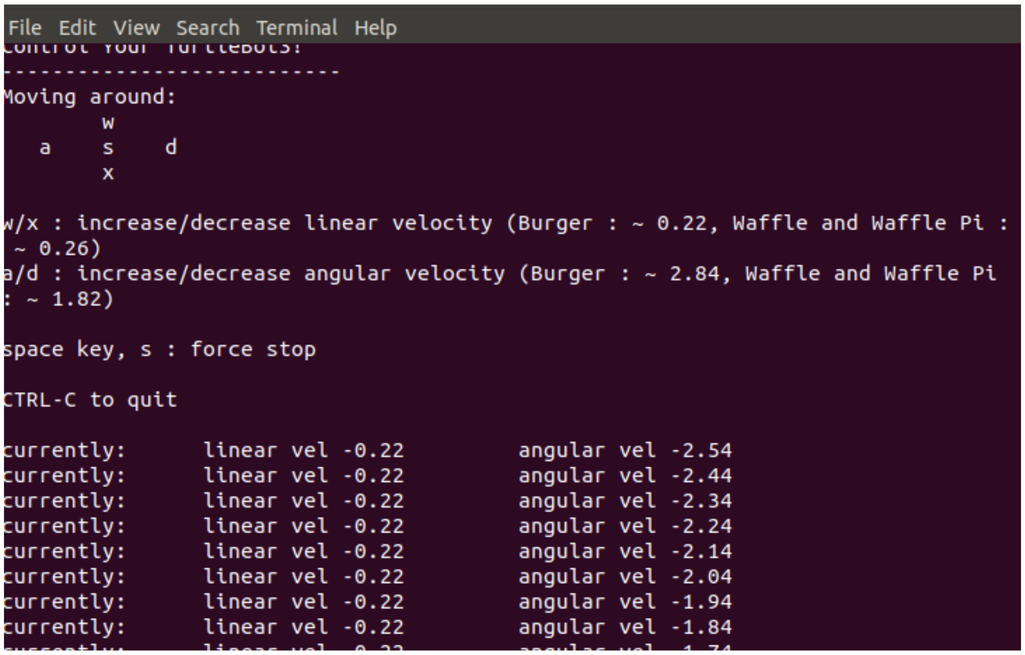
We can also simulate TurtleBot3 inside a house. Type this command and wait a few minutes for the environment to load.

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```



To move the TurtleBot with your keyboard, use this command in another terminal tab:

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```



```
Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi :
 ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi
: ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear vel -0.22         angular vel -2.54
currently:      linear vel -0.22         angular vel -2.44
currently:      linear vel -0.22         angular vel -2.34
currently:      linear vel -0.22         angular vel -2.24
currently:      linear vel -0.22         angular vel -2.14
currently:      linear vel -0.22         angular vel -2.04
currently:      linear vel -0.22         angular vel -1.94
currently:      linear vel -0.22         angular vel -1.84
```

End of Exercise.

Exercise 14. Avoiding Simulated obstacles

Estimated time:

60.00 minutes

What this exercise is about:

Implementing obstacle avoidance for the TurtleBot3 robot.

What you should be able to do:

Implement obstacle avoidance for the TurtleBot3 robot.

Instructor exercise overview:

Now let's implement obstacle avoidance for the TurtleBot3 robot. The goal is to have TurtleBot3 autonomously navigate around a room and avoid colliding into objects.

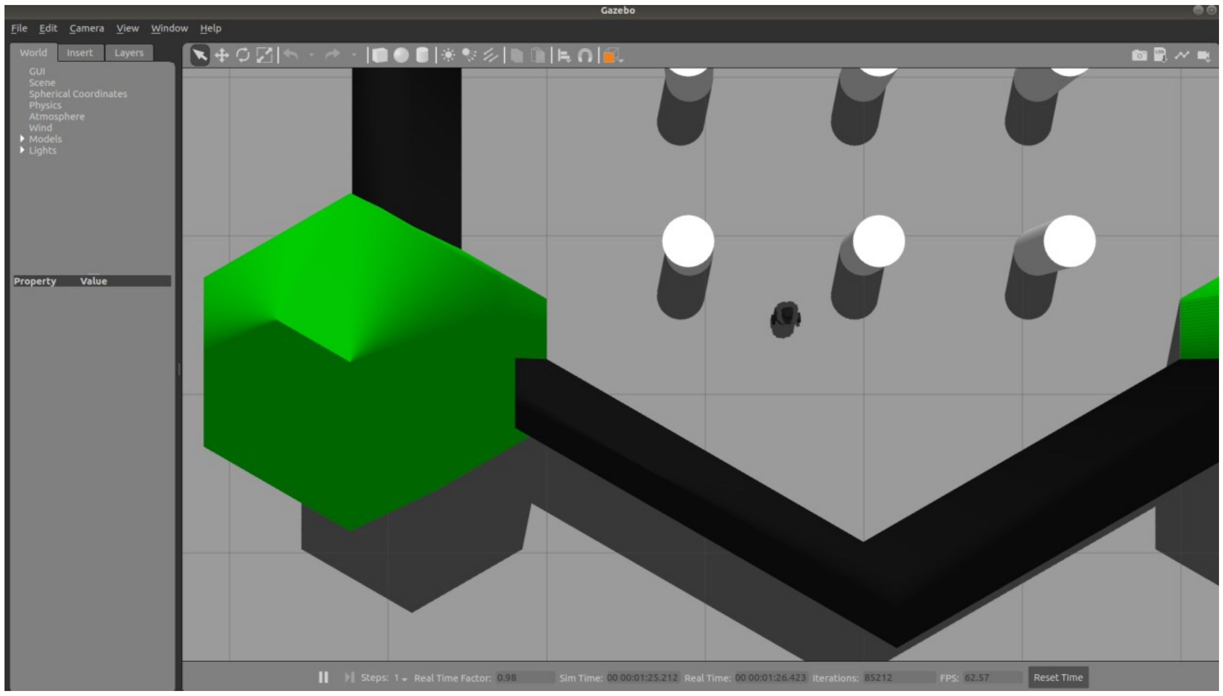
Open a new terminal and type:

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

In another terminal window type:

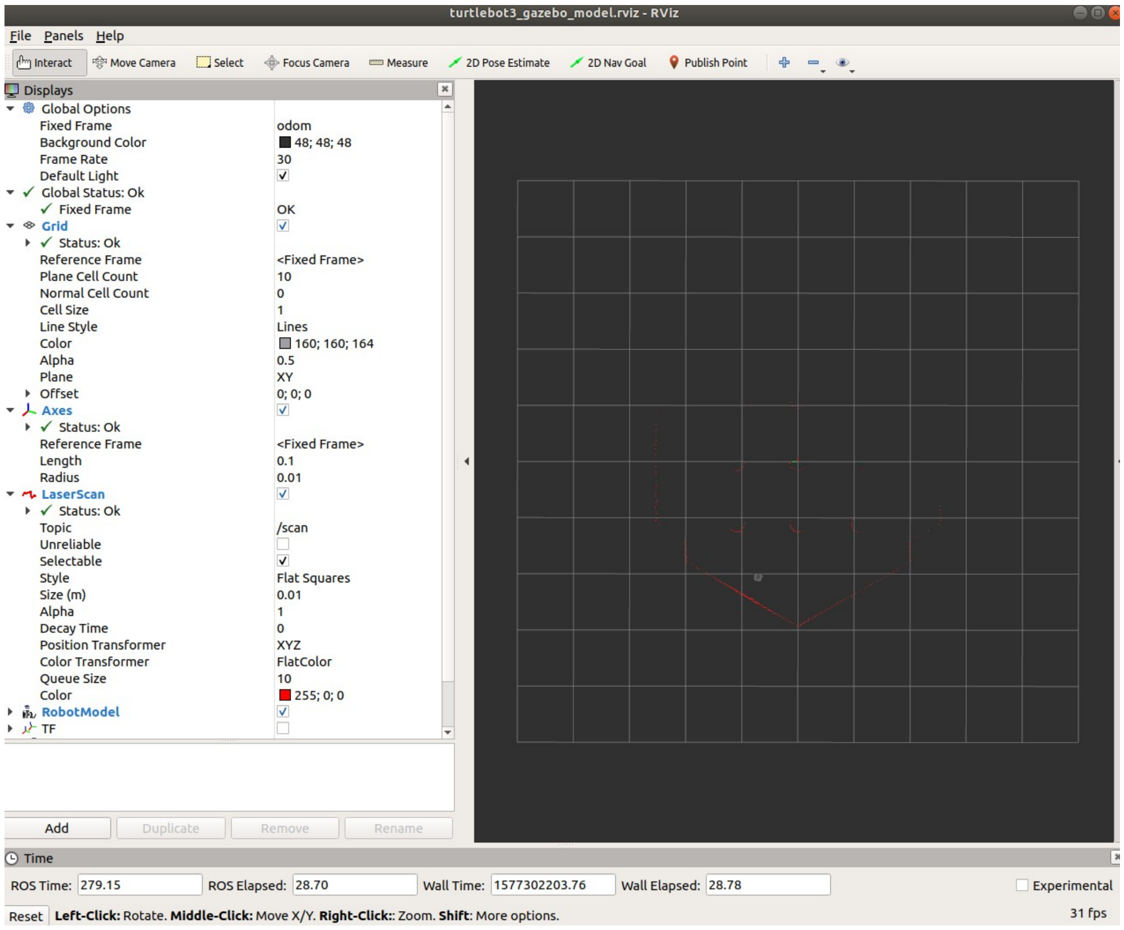
```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

We should see TurtleBot3 autonomously moving about the world and avoiding obstacles along the way.



We can open RViz to visualize the LaserScan topic while TurtleBot3 is moving about in the world. In a new terminal tab type:

```
roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```



Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
✓ Status: Ok	
Reference Frame	<Fixed Frame>
Plane Cell Count	10
Normal Cell Count	0
Cell Size	1
Line Style	Lines
Color	■ 160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
Axes	✓
✓ Status: Ok	
Reference Frame	<Fixed Frame>
Length	0.1
Radius	0.01
LaserScan	✓
✓ Status: Ok	
Topic	/scan
Unreliable	☐
Selectable	✓
Style	Flat Squares
Size (m)	0.01
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	FlatColor
Queue Size	10
Color	🟥 255; 0; 0
RobotModel	✓
TF	☐

ROS Time: 279.15 ROS Elapsed: 28.70 Wall Time: 1577302203.76 Wall Elapsed: 28.78

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click::** Zoom. **Shift**: More options. 31 fps

Simulating SLAM With TurtleBot3

Install SLAM module using:

```
sudo apt install ros-melodic-slam-gmapping
```

Start Gazebo in a new terminal window using

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

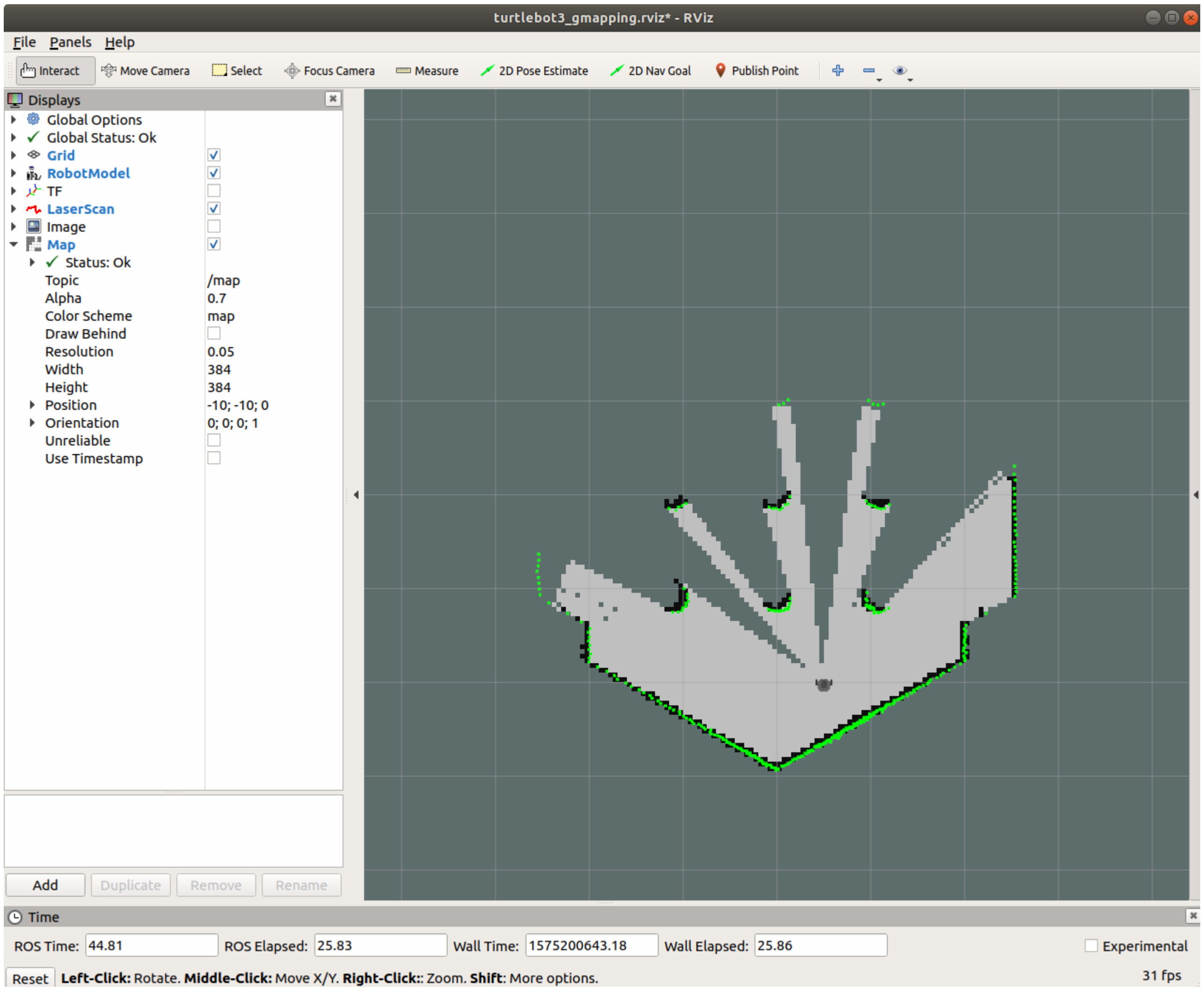
Start SLAM in a new terminal tab using

```
roslaunch turtlebot3_slam turtlebot3_slam.launch
slam_methods:=gmapping
```

Start autonomous navigation in a new terminal tab:

```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

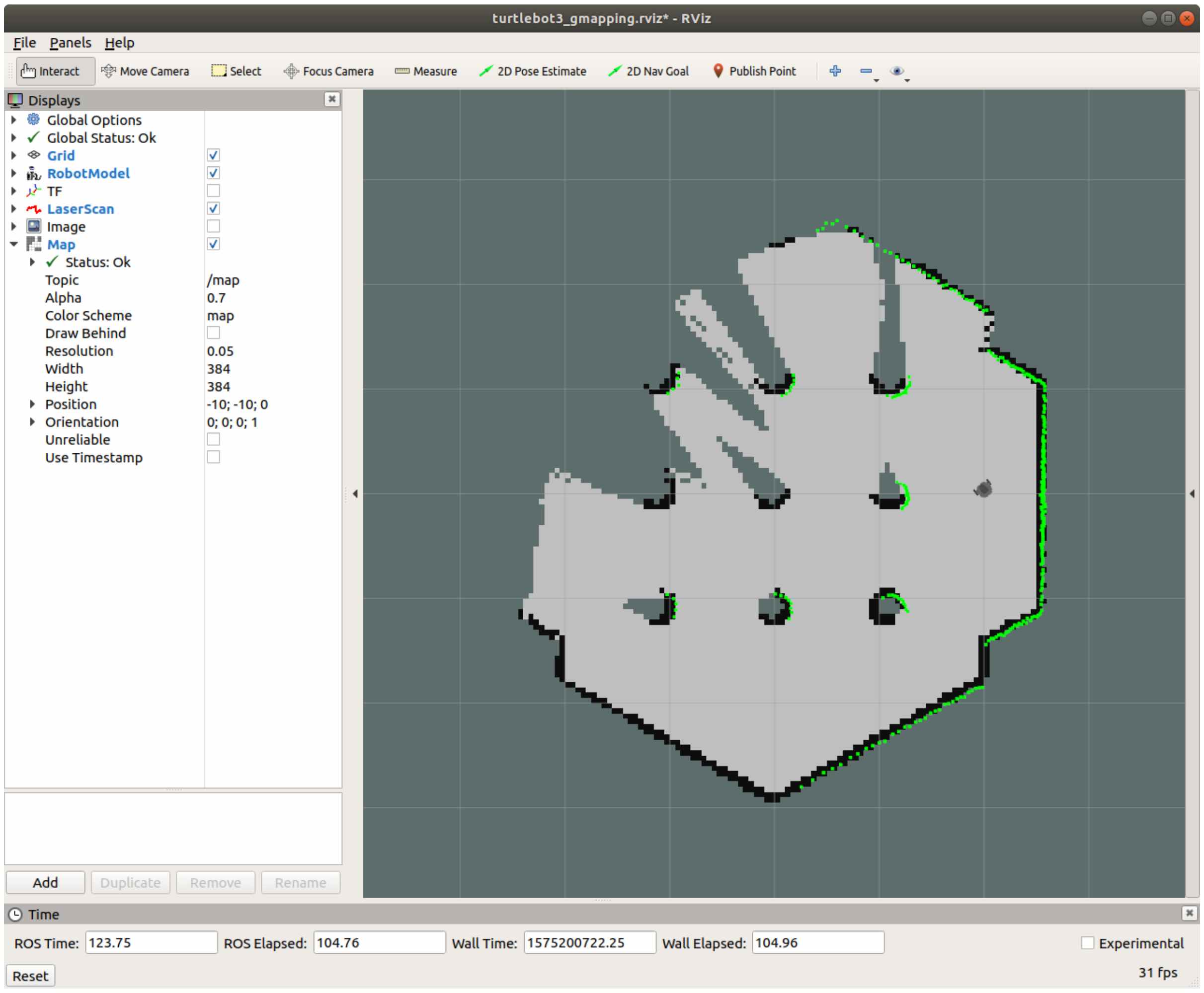
We can see the robot create a map of the environment as it autonomously moves from place to place!



```
Global Options
Global Status: Ok
Grid                ✓
RobotModel          ✓
TF
LaserScan           ✓
Image
Map                 ✓
  Status: Ok
  Topic             /map
  Alpha             0.7
  Color Scheme      map
  Draw Behind
  Resolution        0.05
  Width             384
  Height            384
  Position          -10; -10; 0
  Orientation       0; 0; 0; 1
  Unreliable
  Use Timestamp
```

ROS Time: 44.81 ROS Elapsed: 25.83 Wall Time: 1575200643.18 Wall Elapsed: 25.86

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options. 31 fps


turtlebot3_gmapping.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Add Duplicate Remove Rename
Time
Reset
Experimental
31 fps

Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	384
Height	384
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐

ROS Time: 123.75 ROS Elapsed: 104.76 Wall Time: 1575200722.25 Wall Elapsed: 104.96

End of Exercise.

Exercise 15. Multiple Turtle bot Simulation

Estimated time:

60.00 minutes

What this exercise is about:

Multiple turtle bot simulation

What you should be able to do:

Implement multiple turtle bot simulations

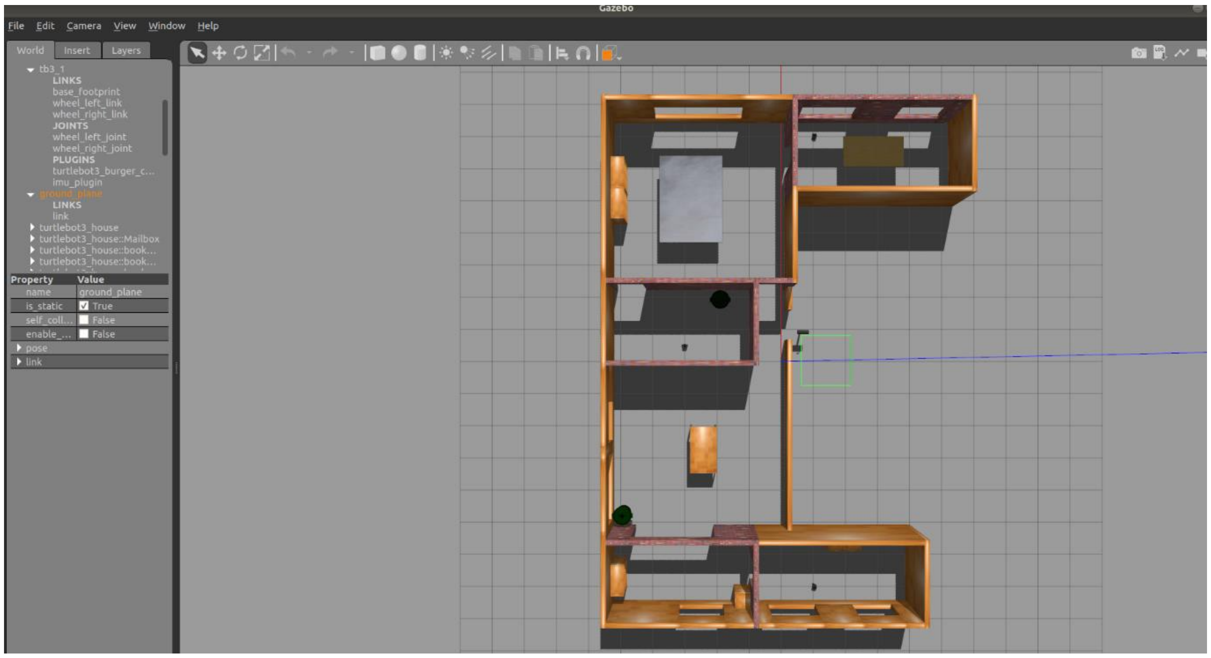
Instructor exercise overview:

Step1. Call Three TurtleBot3s in TurtleBot3 House using the command

```
roslaunch turtlebot3_gazebo multi_turtlebot3.launch
```

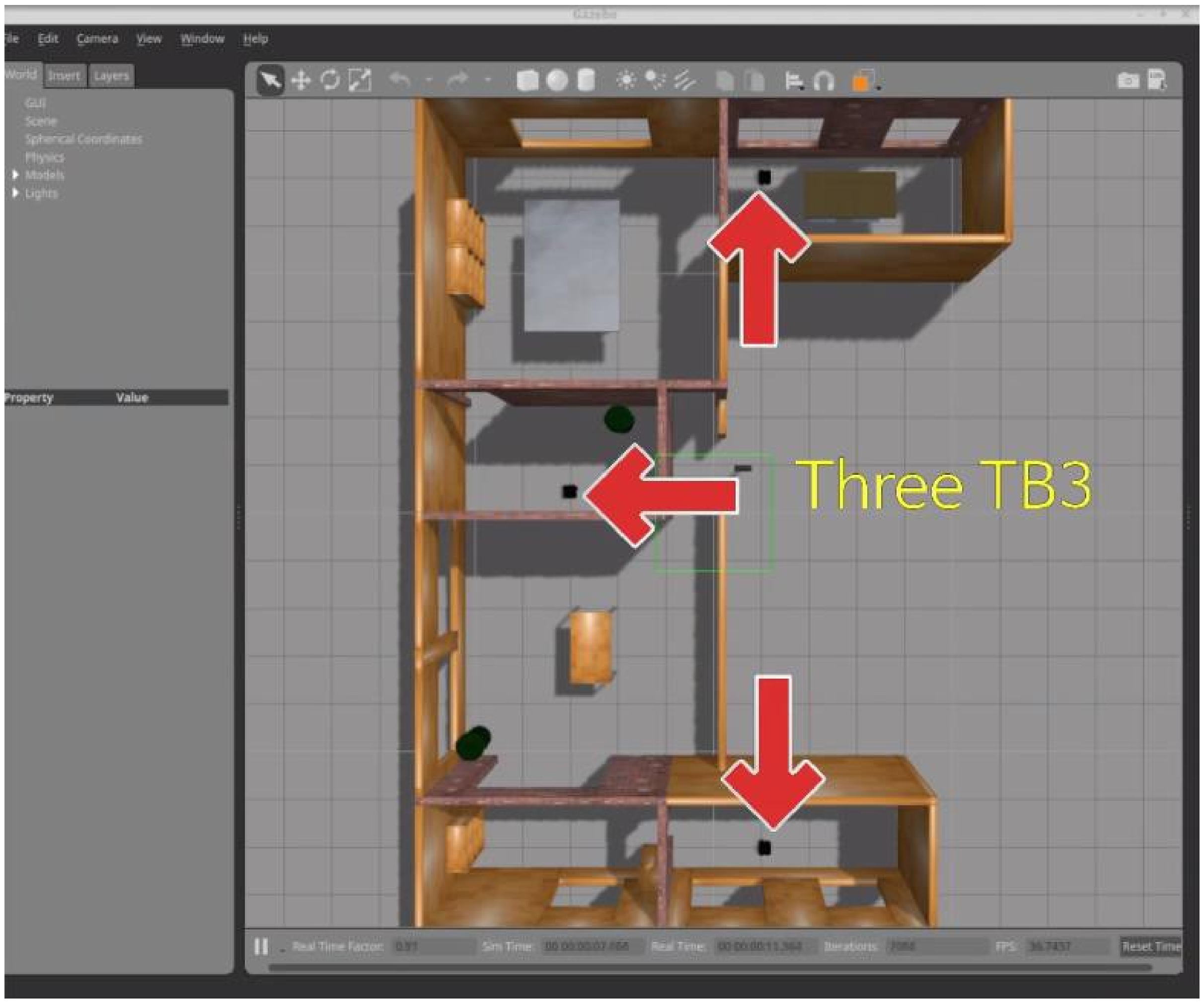
Step 2. Execute the SLAM in three different terminals using the commands

```
ROS_NAMESPACE=tb3_0    roslaunch    turtlebot3_slam    turtlebot3_gmapping.launch
set_base_frame:=tb3_0/base_footprint                   set_odom_frame:=tb3_0/odom
set_map_frame:=tb3_0/map

ROS_NAMESPACE=tb3_1    roslaunch    turtlebot3_slam    turtlebot3_gmapping.launch
set_base_frame:=tb3_1/base_footprint                   set_odom_frame:=tb3_1/odom
set_map_frame:=tb3_1/map


ROS_NAMESPACE=tb3_2    roslaunch    turtlebot3_slam    turtlebot3_gmapping.launch
set_base_frame:=tb3_2/base_footprint                   set_odom_frame:=tb3_2/odom
set_map_frame:=tb3_2/map
```



Step 3. Use the teleoperation to control the three turtle robots using the commands

```
ROS_NAMESPACE=tb3_0 rosrun turtlebot3_teleop turtlebot3_teleop_key
ROS_NAMESPACE=tb3_1 rosrun turtlebot3_teleop turtlebot3_teleop_key
ROS_NAMESPACE=tb3_2 rosrun turtlebot3_teleop turtlebot3_teleop_key
```



End of Exercise.

Exercise 16. Speech related experiment

Estimated time:

60.00 minutes

What this exercise is about:

Implementing text to speech which can be used for robots.

What you should be able to do:

Implement text to speech.

Instructor exercise overview:

1) install pyttsx3 2.71 using

```
pip install pyttsx3
```

2) usage :

```
import pyttsx3
engine = pyttsx3.init()
engine.say("I will speak this text")
engine.runAndWait()
```

3) Changing Voice , Rate and Volume :

```
import pyttsx3
engine = pyttsx3.init() # object creation

""" RATE"""
rate = engine.getProperty('rate')   # getting details of current speaking rate
print (rate)                        #printing current voice rate
engine.setProperty('rate', 125)     # setting up new voice rate


"""VOLUME"""
volume = engine.getProperty('volume')   #getting to know current volume level
(min=0 and max=1)
print (volume)                          #printing current volume level
engine.setProperty('volume',1.0)    # setting up volume level  between 0 and 1

"""VOICE"""
voices = engine.getProperty('voices')       #getting details of current voice
#engine.setProperty('voice', voices[0].id)  #changing index, changes voices. o
for male
engine.setProperty('voice', voices[1].id)   #changing index, changes voices. 1
for female

engine.say("Hello World!")
engine.say('My current speaking rate is ' + str(rate))
engine.runAndWait()
engine.stop()
```

End of Exercise.