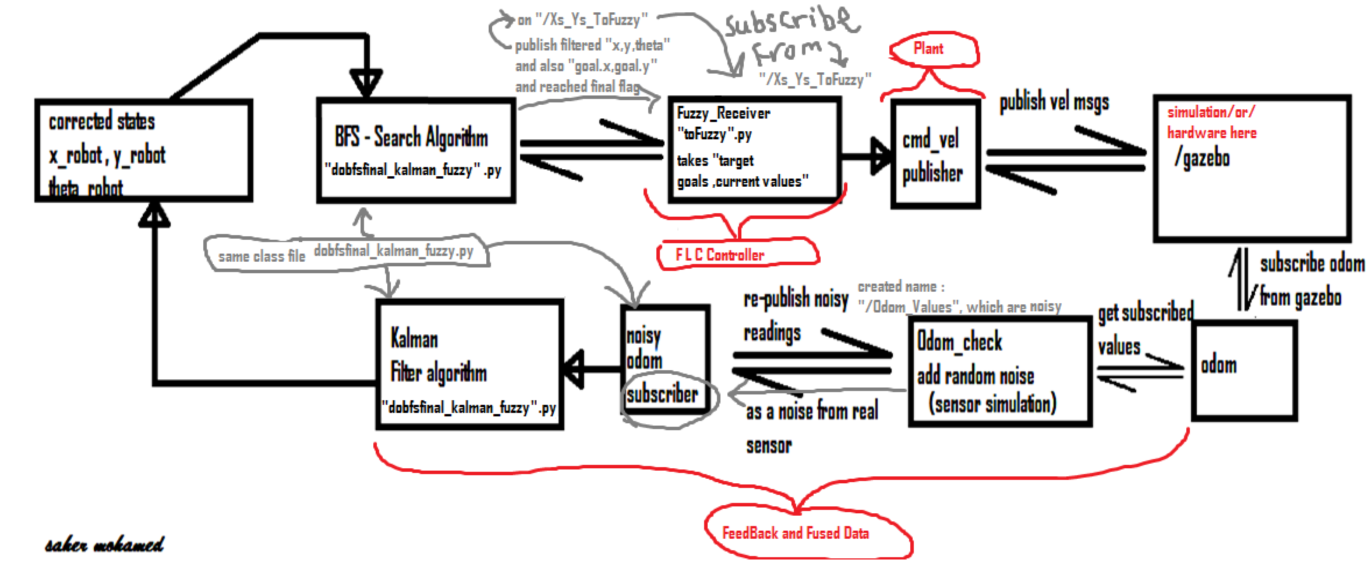
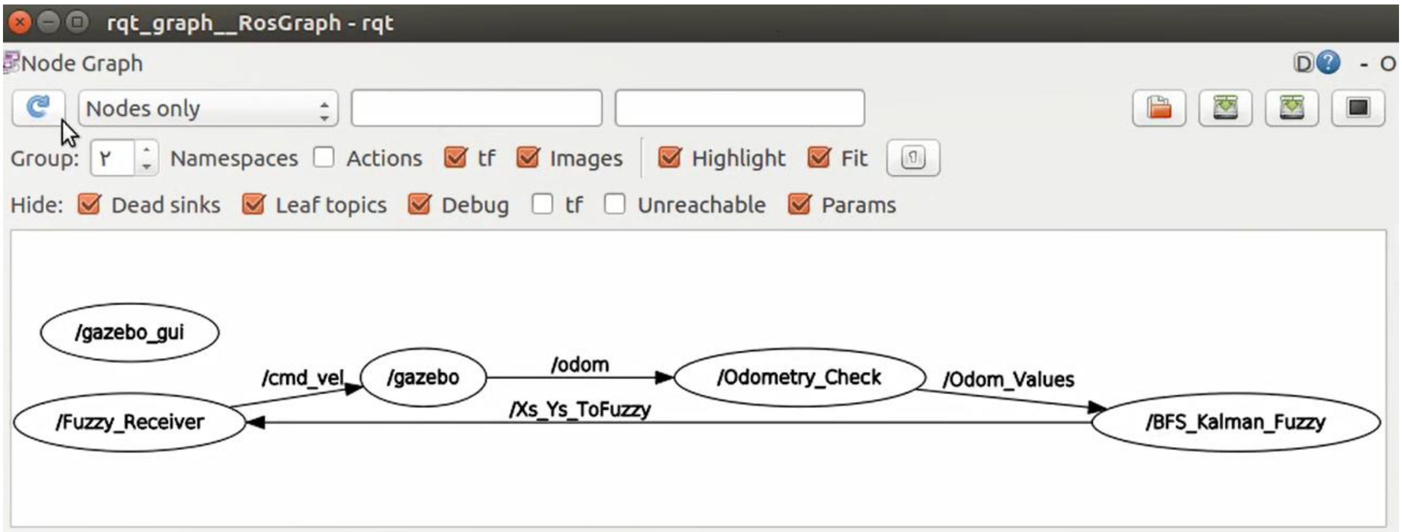
in this report :

the milestone was about making an implementation for FLC Controller , thus "Fuzzy_Rob_Point_2.py" was the fittest one that can be used in our closed loop system , as shown in digram i have drawn for that



and here is the feedbak from "/Odom_values" which published by file "Odom_Check.py" then filtration for sensor readings to get an estimated corrected reading from the sensor , and this can be done by (even simulating a sensor and it`s noise) , or by reading values from real position sensor , then implementing a filtration technique to avoid the noise from the sensor in reality , for example we have some techniques that i learned (Complementary filter : this takes a noisy readings about the linear accelerations (x,y) and by help of (/odom) it implements a filtration technique to get the corrected position ,& the other is kalman filter for only linear systems , and also extended kalman filter (EKF) , the tutorial codes given was only taking the sensor reading (x_p) about the x-axis , thus i implemented it 3 times and this succeeded to implement filtration techniques as shown the full complete rqt_graph assures that the above diagram is implemented successfully



as the code in "BFS_Kalman_Fuzzy" is ordered as well first to get the start and goal points from launch file , which first is to initialize ros node " BFS_Kalman_Fuzzy " and get ros parameters , then opens input image into 2D binarry array of integers , then start to implement the BFS algorithm , and then juming into creating subscriber for the noisy data from odometry_check which simulates the sensor noise ,then get into the kalman part which initialize it , and in the Main loop (while the robot not reach it`s goal) which is last (re-scaled) element in the output BFS path array , it will go inside it .

saher mohamed

inside the loop starts as kalman filter as well , but first it check if it reached first temp goal point "which inside the BFS path array" and if it not , it will then get the noisy x , y , theta and then get their filtered values , by calling the prediction and filteration methods , then sets (curr_x,y,theta) to "\Xs_Ys_ToFuzzy" which will publish (filtered current x , y & theta) then the launch file also lauch another fuzzy controller python file which is updated to ger "current x ,y & theta"filtered which are published on "\Xs_Ys_ToFuzzy" , also another important values are published

which they are goal.x , goal.y , as shown below

```
        #variable for controller x here is the curr x robot from filtered x
        #variable for controller y here is the curr y robot from filtered y
                x = Xfiltered.item(0)##current filtered x to controller
                y = Y_filtered.item(0)##same as x but in y value
                theta = Filtered_Theta.item(0)##same as x but in theta value

                XModel.append(Xpredicted.item(0))   #Array to hold value of x obtained by the sensor
                YModel.append(predicted_Y.item(0))  #Array to hold value of y obtained by the sensor
                ThetaModel.append(Predicted_Theta.item(0))  #Array to hold value of theta obtained by the sensor

                XNoisy.append(X_Previous)                   #Array to hold noisy value of x coordinates
                YNoisy.append(Y_Previous)                   #Array to hold noisy value of y coordinates
                ThetaNoisy.append(Theta_Previous)           #Array to hold noisy value of theta coordinates

                XFiltered.append(Xfiltered.item(0))         #Array to hold Filtered value of x coordinates
                YFiltered.append(Y_filtered.item(0))        #Array to hold Filtered value of y coordinates
                ThetaFiltered.append(Filtered_Theta.item(0)) #Array to hold Filtered value of theta coordinates

                TimeTotal.append(t)

                #print("iam          X filtered" + str(x))
                #print("iam          Y filtered" + str(y))
                #print("iam          Theta filtered" + str(theta))

                pose_msg.position.x = x
                pose_msg.position.y = y
                pose_msg.position.z = goal.x # Since robot will not move in Z axis ,i will exploit this to publish on it goal .x w
temp goal on BFs output path array
                pose_msg.orientation.x = theta
                pose_msg.orientation.y = 0

                pose_msg.orientation.w = goal.y # Since robot will not use it, i will exploit this to publish on it goal .y which
goal on BFs output path array
                pub.publish(pose_msg)
                r.sleep()
```

Filtered published Values

temp goals keep changed according to output BFS array

after it exits the main loop (reached the last temp goal in BFS array path) it sets another flag on the "\Xs_Ys_ToFuzzy" to value 1 to make the "Fuzzy controller on other side" to set , as shown here setting that flag to 1 then publish it before exiting loop

```
    else :
        if (path_counter+1 < turtlebot_len ) :
            path_counter = path_counter + 1
        elif (path_counter+1 >= turtlebot_len ) :
            pose_msg.orientation.z = 1 # publish this flag to fuzzy control also to exit it`s infinite loop there
            pub.publish(pose_msg)
            r.sleep()
            break
        goal.x = (20+float(turtleBotpath_points[path_counter][0])) / 100 #Temp Goal x in Output BFS array Re-scale
        goal.y =(425-float(turtleBotpath_points[path_counter][1])) / 100 #Temp Goal y in Output BFS array Re-scale
        print("i almost reached shifting goals to " + str(goal.x) + " " + str(goal.y))
        pass
'Finished , Destination Reached !! ")
```

then in "toFuzzy" we receive all values as shown in the receiver method used by subscriber in it on next page :

```python
flag_cont = 0   #Initialize flag by zero
pos_msg = Pose()         #Identify msg variable of data type Pose
position = np.zeros((1,6))
Velocity_msg = Twist()
velocity = np.zeros((1,6))
#########################################################################
#########################################################################
#Define the initial pose and velocity of the vehicle
x_p = 0.0
y_p = 0.0
vel_p_x =0.0#*cos(Rob_pos_0[2])
vel_p_y = 0.0#*sin(Rob_pos_0[2])
#########################################################################
##########################################################################################################
#########################################################################################################
x_actual_filtered = 0.0
y_actual_filtered = 0.0
theta_actual_filtered = 0.0
x_goal_rec=0.0
y_goal_rec=0.0
flag_cont_igot = 0
flag_i_exit_loop = 0
def Received_values(msg):
    global x_actual_filtered
    global y_actual_filtered
    global theta_actual_filtered
    global flag_cont_igot
    global x_goal_rec
    global y_goal_rec
    global flag_i_exit_loop
    x_actual_filtered = msg.position.x
    y_actual_filtered = msg.position.y
    theta_actual_filtered = msg.orientation.x
    #i now got filtered actual pose and below i get the goal to go
    x_goal_rec = msg.position.z
    y_goal_rec = msg.orientation.w
    flag_i_exit_loop = msg.orientation.z  # Master will make it 1 when it finsihes there
    flag_cont_igot = 1
```

File "toFuzzy.py"

received x,ytheta filtered

received temp goals x,y and exit loop flag which will be 1 when robot reaches final goal

Python ▾   Tab Widt

then the image below shows after receiving flag by value 1 it will set robot angular and linears velocities to zero then shutdown the ros publisher , and the other file starts plotting the arrays that was filled by "(model / predicted values) and (Noisy valuys) and (Filtered values) for each x,y,theta" and shows up the final figures , and by zooming into one of the output figures it shows up (the model / predicted state , noisy values , and filtered),as below image `s showing setting robot velocites to zero

toFuzzy.py (~/catkin_ws/src/Mile6_splitted/src) - gedit

Open ▾

```python
tau = 0.1 #Sampling Time
rob_mass = 1 #Robot Mass (Turtlebot 3 Waffle_pi)
last = 0
v = 0.0
w = 0.0
while not rospy.is_shutdown():
    if flag_i_exit_loop ==1:
        break
    if (sqrt((x_actual_filtered - x_goal_rec )**2+(y_actual_filtered - y_goal_rec)**2) < 0.07)  :
        vel_msg.linear.x = 0 #Linear Velocity
        vel_msg.linear.y = 0
        vel_msg.linear.z = 0
        vel_msg.angular.x = 0
        vel_msg.angular.y = 0
        vel_msg.angular.z = 0 #Angular Velocity
        #print('velocity is',v)
        pub1.publish(vel_msg)    #Publish msg
        rate.sleep()             #Sleep with rate
```

and below image shows after the loop sets the velocities to zero

```python
            flag_cont = 0
    else:
            #Set the values of the Twist msg to be publeshed
            vel_msg.linear.x = 0 #Linear Velocity
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = 0 #Angular Velocity
    #print('velocity is',v)
    pub1.publish(vel_msg)        #Publish msg
    rate.sleep()                 #Sleep with rate

print ("00000000000000iam here000000000000 End Fuzzy")
vel_msg.linear.x = 0 #Linear Velocity
vel_msg.linear.y = 0
vel_msg.linear.z = 0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
vel_msg.angular.z = 0 #Angular Velocity
pub1.publish(vel_msg)   #Publish msg
rospy.signal_shutdown("for plotting")
print ("00000000000000i finished End Fuzzy")
```

and the image below shows the launch file that responsible for operating the previous files , and it was explained in latter the scale trick for setting start and goal points , and "toFuzzy.py" file is launched without prameters as it will receive the temp goals and filtered x,y,theta and the loop exit flag by subscribing topic "/Xs_Ys_toFuzzy" , and the "odom_check" python file also takes ros parameters which is the standard deviation of the (x,y,theta) to add the noise according to them , as when the variance of the "Gaussian / 'Normal Distribution' curve increases the noise will increase ,which means the curve will be wider ", and the "narrower the curve , the less variance/noise", **finally** if you have finished all this reading "the trick behind name of "Mile6_Splitted" as i tried to bind all up in one file , but due to timing faults at "RunTime" inside the code the robot didn`t succeed to implement successfully , so i Split the FLC controller code and the temp goals and filtered positions are published then received by that "toFuzzy" , so that i renamed that test with name "splitted" for mention the "Split" action of the FLC controller code away from the Main "BFS_Kalman" file.

Mile6.launch (~/catkin_ws/src/Mile6_splitted/launch) - gedit

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="0.5"/>
  <arg name="y_pos" default="0.5"/>
  <arg name="z_pos" default="0.0"/>
<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="/home/saher/catkin_ws/src/Mile6_splitted/ID_Map/ID_World.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
<param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
 <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />

<node pkg="Mile6_splitted" name="Odometry_Check" type="Odom_Check.py">
    <param name="xstd" value="0.1" />
    <param name="ystd" value="0.1" />
    <param name="thetastd" value="0.1" />
   </node>
    <node pkg="Mile6_splitted" name="Fuzzy_Receiver" type="toFuzzy.py" output="screen" >
    </node>
  <node pkg="Mile6_splitted" name="BFS_Kalman_Fuzzy" type="dobfsfinal_kalman_fuzzy.py" output="screen" >
    <param name="x_Start" value="50.0" />
    <param name="y_Start" value="350.0" />
    <param name="x_Goal" value="600.0" />
    <param name="y_Goal" value="60.0" />
   </node>
  <!-- <node pkg="Mile6" name="Odometry_Plot" type="Plotter.py" output="screen" launch-prefix="xterm -e">
    <param name="xstd" value="0.1" />
    <param name="ystd" value="0.1" />
    <param name="thetastd" value="0.1" />
   </node>-->
 <node pkg="rqt_graph" name="rqt_graph" type="rqt_graph"/>

</launch>
```

Plain Text Tab Width: 8 Ln 7, Col 100 INS

saher mohamed