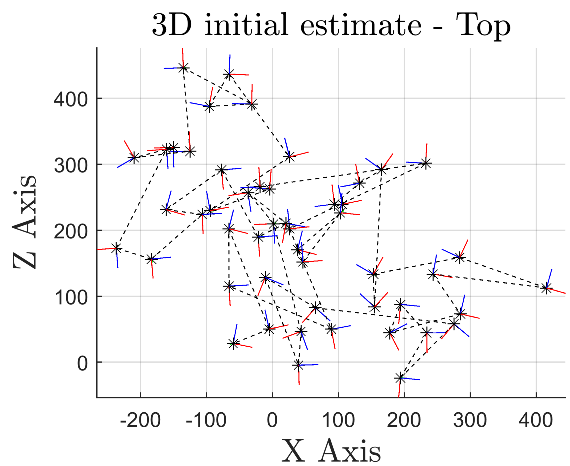
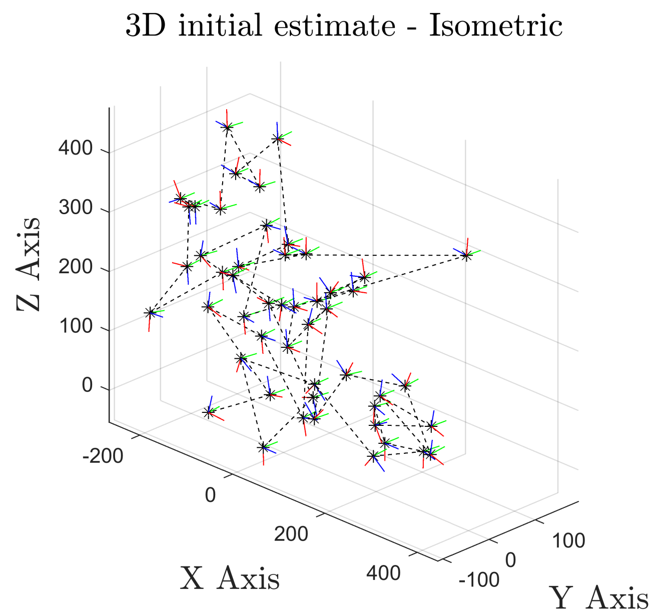
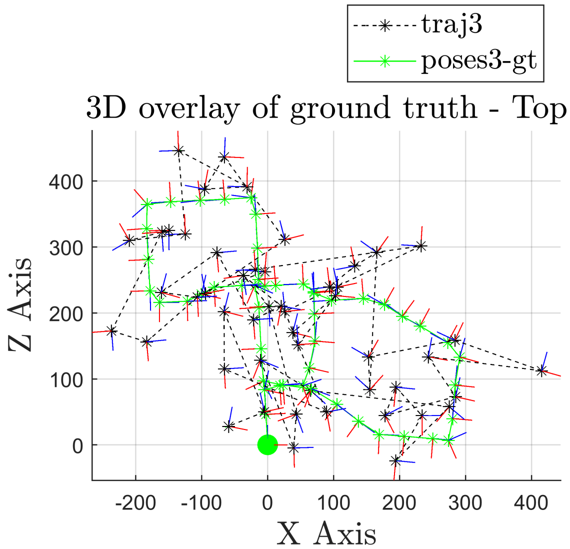
```
clear all; clc; close all; set(0,'defaultfigurecolor',[1 1 1]);
addpath('C:\Users\Daniel\Desktop\Vision Aided Navigation\Hw_i\Hw_4\gtsam_toolbox');

% ----------------------- Initialization of Project ----------------------- %
import gtsam.*
load('hw4_data.mat');                     % file contains the following subsets :
num_Poses = length(traj3);                % Number of states / camera poses
fig_size = [0 0 650 350];                 % figure [left bottom width height]
fig_subplot = [0 0 900 500];              % figure size for 2 adjecent plots

% --------------------------------- Run --------------------------------- %

% (!!) To be UNCOMMENTed in Clause 4 (!!) and re-run the Code
% R_4 =  [0.330571768  0.0494690228 -0.942483486;
%     0.0138000518 0.998265226   0.0572371968
%     0.943679959 -0.0319273223  0.329315626];
% t_4 = [-24.1616858 -0.0747429903 275.434963]';
% T_4 = [[R_4 t_4]; [0 0 0 1]];
% dpose{3} = T_4; dpose{42} = T_4;

% 1.a  Convert transformation(s) to gtsam.Pose3 objects
for i = 1:num_Poses
    gTc_traj.compose{i} = Pose3(traj3{i});
    gTc_true.compose{i} = Pose3(poses3_gt{i});
    if (i < num_Poses)
        gTc_dnoise.compose{i} = Pose3(dpose{i});
    end
end

% 1.b  Store it in an object, representing the initial estimate for robot trajectory
trajectory   = gtsam.Values;              % Instantiate as Values object
ground_Truth = gtsam.Values;              % Instantiate as Values object
Noisy_meas   = gtsam.Values;              % Instantiate as Values object
key = uint64(zeros(1,num_Poses));         % uint64 row vector initilization

for i = 1:num_Poses
    key(i) = gtsam.symbol('x', i-1);      % pointer for P(x_i| x_i-1, u)
    trajectory.insert(key(i), gTc_traj.compose{i});
    ground_Truth.insert(key(i), gTc_true.compose{i});
    if (i < num_Poses)
        Noisy_meas.insert(key(i), gTc_dnoise.compose{i});
    end
end

% 1.c     Display list of poses
figure;  set(gcf, 'Position', fig_subplot);
for i = 1:2
    subplot(1, 2, i)
    gtsam.plot3DTrajectory(trajectory, 'k--*', [], 30);
    axis equal tight; grid on;
    if (i==1)
        axis equal tight; view([0, -1, 0]);
```

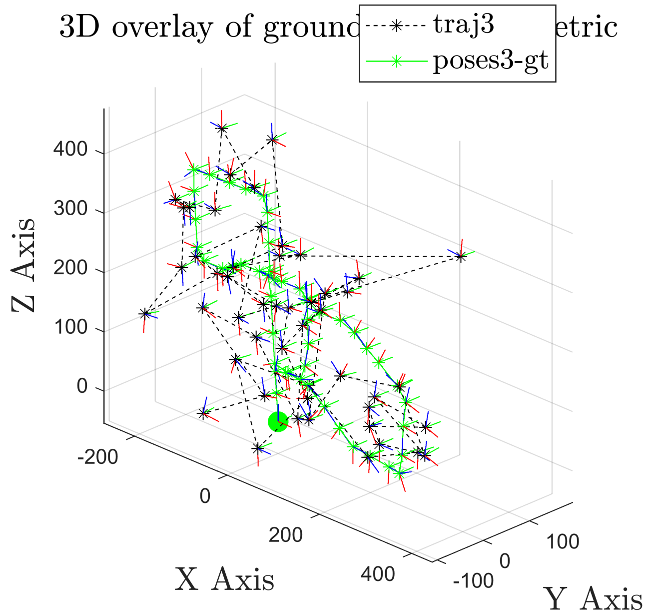
```
        set_Axes('3D initial estimate - Top', 'traj3', 'poses3-gt', 0);
    else
        axis equal tight; view(45, 25);
        set_Axes('3D initial estimate - Isometric', 'traj3', 'poses3-gt', 0);
    end
end
```





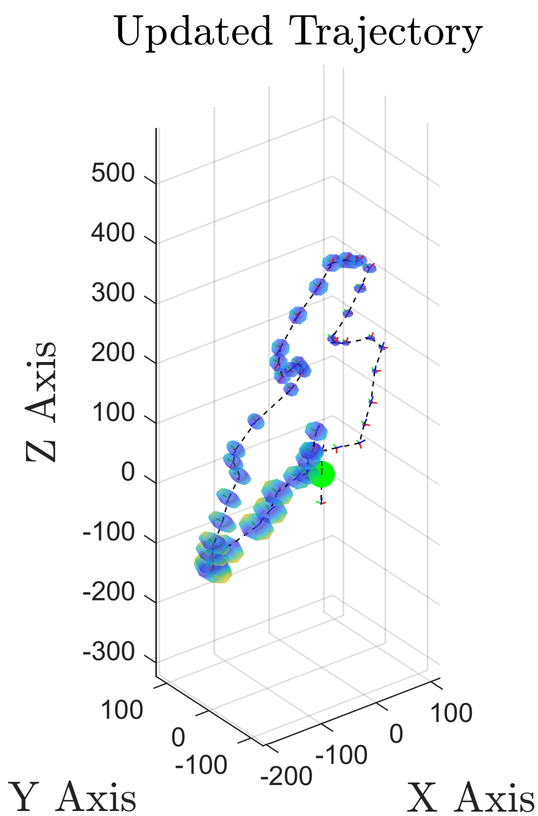

```
% 1.d     Overlay in your plot the ground truth trajectory given in poses3_gt
figure;  set(gcf,'Position', fig_subplot);
for i = 1:2
    subplot(1, 2, i)
    grid on; hold on;
    gtsam.plot3DTrajectory(trajectory,   'k--*', [], 30);
    gtsam.plot3DTrajectory(ground_Truth, 'g-*',  [], 30);
    axis equal tight; scatter3(0, 0, 0, 100, 'go', 'filled');
    if (i==1)
        set_Axes('3D overlay of ground truth - Top', 'traj3', 'poses3-gt', 1);
        axis equal tight; view([0, -1, 0]);
    else
        set_Axes('3D overlay of ground truth - Isometric', 'traj3', 'poses3-gt', 1);
        axis equal tight; view(45, 25);
    end
end
```



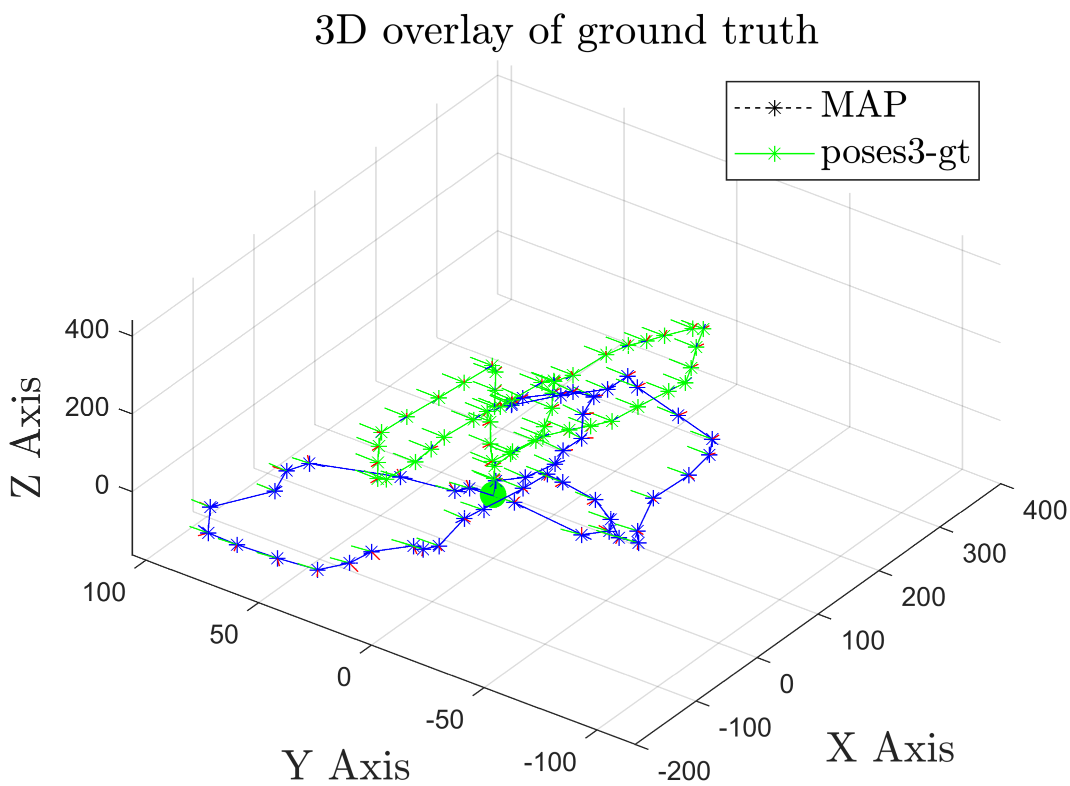



```
% Save objects :
saved_trajectory = trajectory.saveobj;
saved_ground_Tru = ground_Truth.saveobj;

%% 2       Construct factor graph using dpose variables (= noisy poses)
% 2.a     Create a general factor graph
graph = gtsam.NonlinearFactorGraph;

% 2.b     Add pose factors between 'measured' RELATIVE poses
S_r = 1e-3*[1 1 1];     S_t = 1e-1*[1 1 1];     % Standard deviations [rad, m]
Sig_v = noiseModel.Diagonal.Sigmas([S_r S_t]'); % Measurement covariance


% 'measured' relative pose
for i = 1:length(dpose)
    graph.add( BetweenFactorPose3(key(i), key(i+1), gTc_dnoise.compose{i}, Sig_v ));
end

% 2.c Assume robot is located & aligned at the origin
origin = gtsam.Pose3;                               % Relative pose
priorNoise = Sig_v;
graph.add(PriorFactorPose3(gtsam.symbol('x', 1), origin, priorNoise));

%% 3        Calculate and display the MAP trajectory estimate
% 3.a     Create and run an optimzer using the initial estimate
optimizer = gtsam.LevenbergMarquardtOptimizer(graph, trajectory);
result = optimizer.optimizeSafely();
```

```
% 3.b     Display updated trajectory estimate
marginals = Marginals(graph, result);

figure; grid on; hold on; % set(gcf,'Position', fig_size);
gtsam.plot3DTrajectory(result, 'k--', 1, 10, marginals);
scatter3(0, 0, 0, 100, 'go', 'filled');
set_Axes('Updated Trajectory', [], [], 0); view(3);
```




```
% 3.c     Overlay with the ground truth as in the 1st clause
figure; grid on; hold on; % set(gcf, 'Position', fig_subplot);
gtsam.plot3DTrajectory(result, 'b-*', [], 10);
gtsam.plot3DTrajectory(ground_Truth, 'g-*',  [], 10);
scatter3(0, 0, 0, 100, 'go', 'filled');
set_Axes('3D overlay of ground truth', 'MAP', 'poses3-gt', 1);
view(-54, 54);
```




```
%% 4    Loop closure. Suppose the robot observed the same scene twice :
% 4.a (!!!) image to be taken from previous clause and inserted here (!!!)

% 4.b Plot localization error (Euclidean distance)
V_len = length(traj3);
% X_c(:,1) = [0 0 0 1]; GT_c = X_c;            % Initialize at origin
Err_norm = zeros(1, V_len);
compare_trajectory = result;                 % Object to be compared with (!)
key_rs = gtsam.KeyVector(compare_trajectory.keys);
key_GT = gtsam.KeyVector(ground_Truth.keys);

% Extract 3D points from trajectories
for i = 0:V_len-1
    % Extract 3D points from estimated trajectory (result object)
    key_1 = key_rs.at(i); X_1 = compare_trajectory.at(key_1);
    V_1(:,i+1) = [X_1.x X_1.y X_1.z]';
    % Extract 3D points from Ground truth         (ground_Truth object)
    key_2 = key_GT.at(i); X_2 = ground_Truth.at(key_2);
    V_2(:,i+1) = [X_2.x X_2.y X_2.z];
    % Calculate location difference between every 3D points
    Err_norm(i+1) = norm(V_1(:,i+1) - V_2(:,i+1));% Euclidean Norm
end

figure;
plot(1:V_len, Err_norm, '-', 'linewidth', 2.5);
ind(1) = title( 'Localization error vs. time' );
```

```
ind(2) = xlabel('Time step');
ind(3) = ylabel('$\Delta$ err');
set(ind, 'Interpreter', 'latex', 'fontsize', 16); grid on;
```