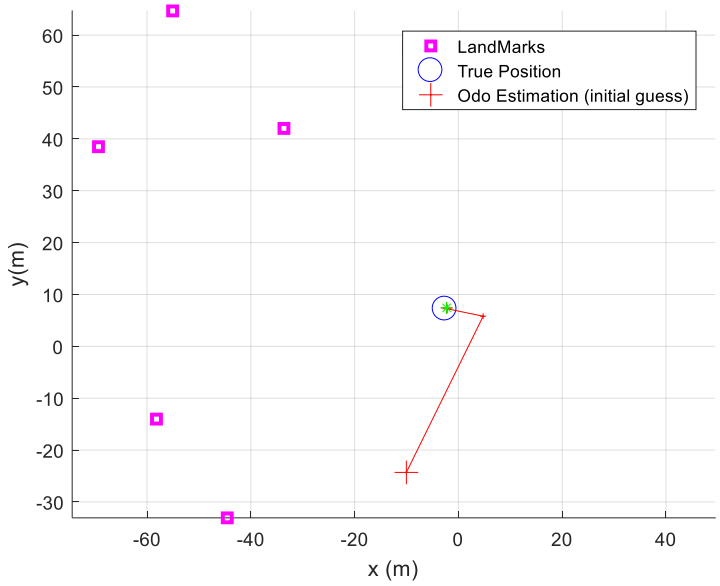
Robotics

Exercise 5.1 Least Squares Global Localization

The attached code partially implements the global localization of a robot equipped with a range sensor by means of Least Squares. This is done by the following steps:

1. The map is built. In this case, the map consists of a number of landmarks (`nLandmarks`).
2. The program asks the user to set the true position of the robot (`xTrue`) by clicking with the mouse in the map.
3. A new pose is generated from it, `xOdom`, which represents the pose that the robot thinks it is in. This simulates a motion command from an arbitrary pose that ends up with the robot in `xTrue`, but it thinks that it is in `xOdom`.
4. Then the robot takes a range measurement to each landmark in the map.
5. Finally, the robot employs a Least Squares definition of the problem and Gauss-Newton to iteratively optimize such a guess, obtaining a new (and hopefully better) estimation of its pose `xEst`.

The figure below shows an example of execution of this code (once completed).



Your tasks in this exercise are:

- Complete the code by filling the code-gaps.
- Which is the minimum number of landmarks needed for localizing the robot? Why?

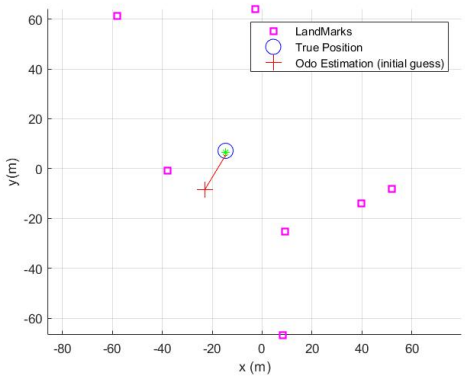


Image 1. LS with 7 landmarks

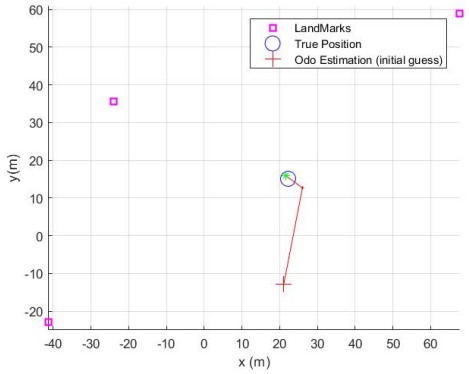


Image 2. LS with 3 landmarks

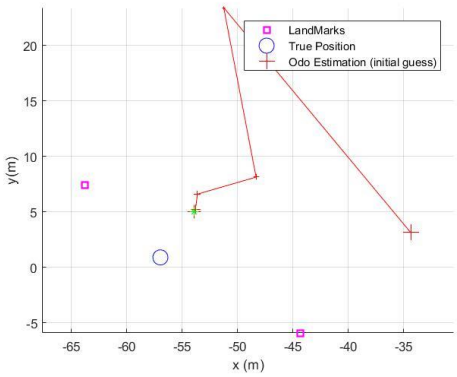


Image 3. LS with 2 landmarks

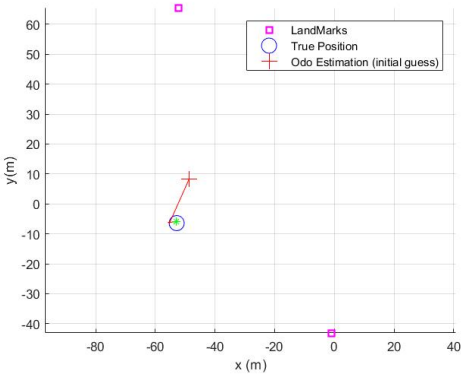


Image 4. LS with 2 landmarks

At least we need 3 landmarks for working well because it can triangulate the approximate real position of the robot. However, with 2 sometimes works well only if the estimated position is close enough to the real position. Furthermore, it works because we only doing a range-only positioning. Also, nearly always when the program runs need 4 iteration to localize where is the robot.

- Play with different “qualities” of the range sensor. Could you find a value for its variance, so the LS method fails?

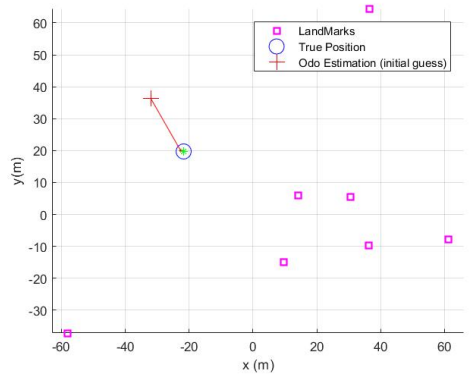


Image 5. LS with var_d=0.01

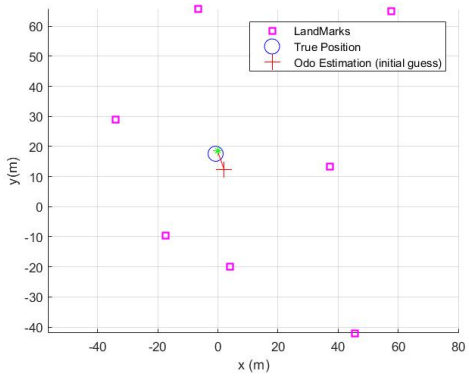


Image 6. LS with var_d=16

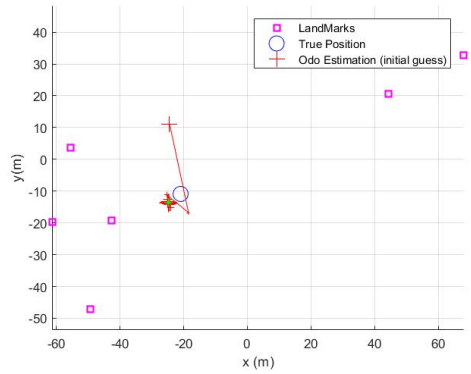


Image 7. LS with var_d=16

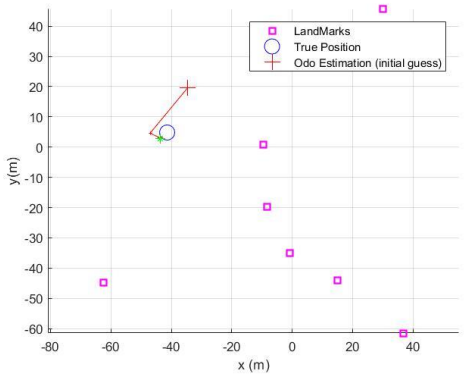


Image 8. LS with var_d=25

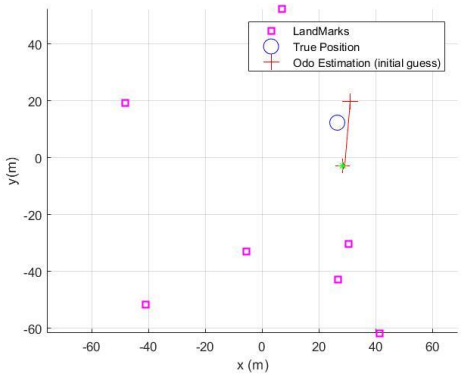


Image 9. LS with var_d = 100

The value that the variance of the robot measuring error for not working as intended is around 16 (4^2).

- Play also with different values for the odometry uncertainty.

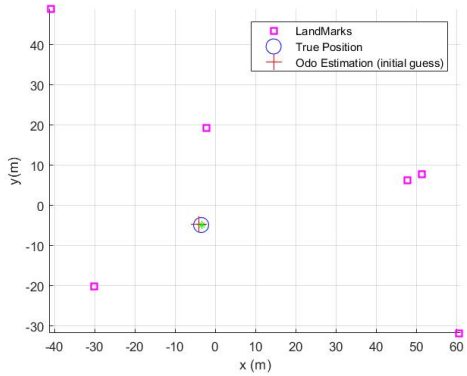


Image 10. LS with U = diag([1,1,pi/180]).^2

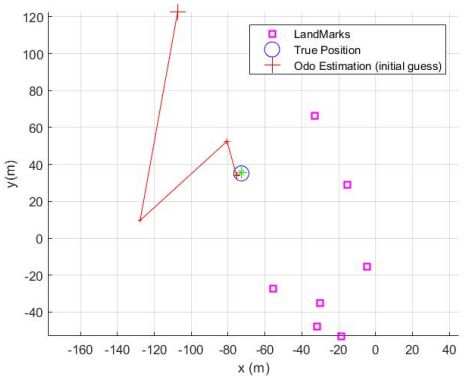


Image 11. LS with U = diag([100,100,pi/180]).^2

As we see in the images, when we have little values on the covariance, the program needs less iteration to find the real position of the robot. In the other hand, when we have big values on the covariance, it needs more iteration.

```
clear variables; close all; clc

% Initialization
%--------------------------------------------------------------------------

% Map/landmarks related
nLandmarks = 7;
mapSize = 140;  %Size of them environment in (m)
Map = mapSize*rand(2,nLandmarks)-mapSize/2; %Landmarks uniformly distributed in
the Map

% Sensor/odometry related
var_d = 0.5^2;      % variance (noise) of the range measurement
R = zeros(nLandmarks);  % Covariance of the observation of the landmarks
z = zeros(nLandmarks,1); % Initially. all the observations equals to zero
U = diag([9,20,1*pi/180]).^2; % Covariance of the odometry noise

% Robot pose related
xTrue = zeros(3,1); %True position, to be selected by the mouse
xEst = zeros(3,1);  %Position estimated by the LSE method
xOdom = zeros(3,1); %Position given by odometry (in this case xTrue affected by
noise)

% Initial graphics
figure(1); hold on; grid off; axis equal; grid on;
plot(Map(1,:),Map(2,:),'sm','LineWidth',2);hold on;
xlabel('x (m)');
ylabel('y(m)');
legend('LandMarks');

% Get the true position of the robot (ask the user)
fprintf('Please, click on the Figure where the robot is located: \n');
xTrue (1:2) = ginput(1)'; %
plot(xTrue(1),xTrue(2),'ob','MarkerSize',12)

% Set an initial guess: Where the robot believes it is (from odometry)
xOdom = xTrue + sqrtm(U)*randn(3,1);
plot(xOdom(1),xOdom(2),'+r','MarkerSize',12);
legend('LandMarks','True Position','Odo Estimation (initial guess)');
sqrt((xOdom(1)-xTrue(1))^2+(xOdom(2)-xTrue(2))^2)

% Take measurements
%--------------------------------------------------------------------------

% Get the observations to all the landmarks (data given by our sensor)
for kk = 1: nLandmarks
   % Take an observation to each landmark, i.e. compute distance to each
   % one (RANGE sensor) affected by gaussian noise
  lm = Map(:,kk)';
  pr = xTrue(1:2)';
  z(kk)=pdist2(pr,lm)+randn*sqrt(var_d);
end

% Pose estimation using Gauss-Newton for least squares optimization
%--------------------------------------------------------------------------

% Some parameters for the Gauss-Newton optimization loop
nIterations = 10; % sets the maximum number of iterations
tolerance = 0.001;% Minimum error needed for stopping the loop (convergence)
iteration = 0;

% Initialization of useful vbles
incr = ones(1,2); % Delta
```

```
jH = zeros(nLandmarks,2); % Jacobian of the observation function of all the
landmarks
xEst = xOdom;     %Initial estimation is the odometry position (usually noisy)

% Let's go!
while (norm(incr) > tolerance && iteration < nIterations)
    plot(xEst(1),xEst(2),'+r','MarkerSize',1 +
floor((iteration*15)/nIterations));

    % Compute the predicted observation (from xEst) and their respective
    % Jacobians

    % 1) Compute distance to each landmark from xEst
    % (estimated observations)
    ez = zeros(nLandmarks,1);
    for kk = 1: nLandmarks
        lm = Map(:,kk)';
        po = xEst(1:2)';
        ez(kk)= pdist2(po,lm);
    end
    %predicted observations

    % error = difference between real observations and prediced ones.
    e = z-ez;
    residual = sqrt(e'*e);  %residual error = srqt(xÂ²+yÂ²)

    % 2) Compute Jacobians with respect (x,y) (slide 13)
    % The jH is evaluated at our current guest (xEst) -> z_p
    dx=-1./ez; %% -1/dn
    z_p = xEst(1:2);
    jH=zeros(size(Map))';
    for i=1:size(Map,2)
        diff=Map(:,i)-z_p; %% (xn-x) (yn-y)
        jH(i,:)=[dx(i)*diff(1) dx(i)*diff(2)]; %%-1/dn * [(xn-x) (yn-y)]
    end


    % The observation variances R grow with the root of the distance
    R = diag(var_d*sqrt(z));

    % 3) Solve the equation --> compute incr
    Qinv=inv(R);
    Hf=jH'*Qinv*jH;
    gf=jH'*Qinv*e;
    incr=Hf\gf;


    % update position estimation
    plot([xEst(1), xEst(1)+incr(1)],[xEst(2) xEst(2)+incr(2)],'r');
    xEst(1:2) = xEst(1:2) + incr;
    fprintf('Iteration number %u residual: %1.4f [m] increment: %1.5f
[m]\n',iteration+1,residual,norm(incr));
    iteration = iteration + 1;

    pause(1);

end

plot(xEst(1),xEst(2),'*g') %The last estimation is plot in green
```