Robotics: Least Squares Global Localization

Salvador Carrillo Fuentes

November 2018

1 Introduction

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 tasks covered in this exercise are:

- Completion of the code by filling the code-gaps.
- Which is the minimum number of landmarks needed for localizing the robot? Why?
- Play with different "qualities" of the range sensor. Could you find a value for its variance so the LS method fails?
- Play also with different values for the odometry uncertainty.

2 First task

- Completion of the code by filling the code-gaps.

The complete code can be found in the file *practice4_1.m*. Next, the most relevant parts of the code are included along with their description.

As indicated in the introduction, in the fourth step the robot takes a range measurement to each landmark in the map. To perform that task, we do the same procedure for each landmark, so a *for* loop is required. The first thing we need to do is to isolate the coordinates of the *kk-th* landmark in the `Map`. So, `x_lm` indicates the *kk-th* landmark's *x-axis* coordinate, and `y_lm` the same but for the *y-axis*.

With this information, we are ready to compute de euclidean distance to the *kk-th* landmark using:

$$d_i = \sqrt{(x_i - x)^2 + (y_i - y)^2}$$

being (x_i, y_i) the coordinates of the landmark and (x, y) the coordinates of the robot.

At this point, the measurement has no error associated (`zz_ideal(kk)`). It's possible to simulate sensor imprecision adding gasussian noise. So, in `zz(kk)` we have the measurement affected by a gaussian error with a variance `var_d`.

```
% Get the observations to all the landmarks
% Data given by our sensor affected by gaussian noise
for kk = 1 : nLandmarks
   x_lm = Map(1,kk);
   y_lm = Map(2,kk);

   z_ideal(kk) = sqrt((x_lm - xTrue(1))^2 + (y_lm - xTrue
       (2))^2);

   z(kk) = z_ideal(kk) + normrnd(0,var_d);
end
```

In the fifth step, the robot employs a Least Squares definition of the problem and Gauss-Newton to iteratively optimize such a guess.

The *Gauss-Newton* algorithm is an iterative algorithm governed by the value of δ. The distance to each landmark from `xEst` (estimated observations) is computed using the same procedure used in step 4 for `xTrue`. Once we have `z_p`, we can calculate de difference between real observations and prediced ones. Next, we calculate jacobians with respect (x, y) for each landmark, ∇h_x (`hT`) using:

$$\frac{\partial}{\partial x}[(x_i - x)^2 + (y_i - y)^2]^{\frac{1}{2}} = -\frac{1}{d_i}(x_i - x)$$

$$\frac{\partial}{\partial y}[(x_i - x)^2 + (y_i - y)^2]^{\frac{1}{2}} = -\frac{1}{d_i}(y_i - y)$$

Finally, the iteration is finished by calculating δ (`incr`) using the jacobians just computed and the covariance of the observation of the landmarks (`R`):

$$\delta = (J_e^T Q^{-1} J_e)^{-1} J_e^T Q^{-1} e$$

```
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)

    z_p = zeros(nLandmarks,1); % predicted observations

    for kk = 1 : nLandmarks
       x_lm = Map(1,kk);
       y_lm = Map(2,kk);

       z_p(kk) = sqrt((x_lm - xEst(1))^2 + (y_lm - xEst
           (2))^2); % Euclidean distance to the kk-th
           landmark
    end

    % error = difference between real observations and
       prediced ones.
    e = z - z_p;
    residual = sqrt(e'*e);  %residual error

    % 2) Compute Jacobians with respect (x,y) (slide 13)
    % The jH is evaluated at our current guest (xEst) ->
       z_p
    for kk = 1 : nLandmarks
        x_lm = Map(1,kk);
        y_lm = Map(2,kk);

        jH(kk,1) = -(x_lm - xEst(1)) / z_p(kk);
        jH(kk,2) = -(y_lm - xEst(2)) / z_p(kk);
    end

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

```
34
35      % 3) Solve the equation --> compute incr
36      incr = inv(jH'*inv(R)*jH) * jH'*inv(R)*e;
37
38      % update position estimation
39      plot([xEst(1), xEst(1)+incr(1)],[xEst(2) xEst(2)+incr
            (2)],'r');
40      xEst(1:2) = xEst(1:2) + incr;
41      fprintf('Iteration number %u residual: %1.4f [m]
            increment: %1.5f [m]\n',iteration+1,residual,norm(
            incr));
42      iteration = iteration + 1;
43
44      pause(1);
45  end
```

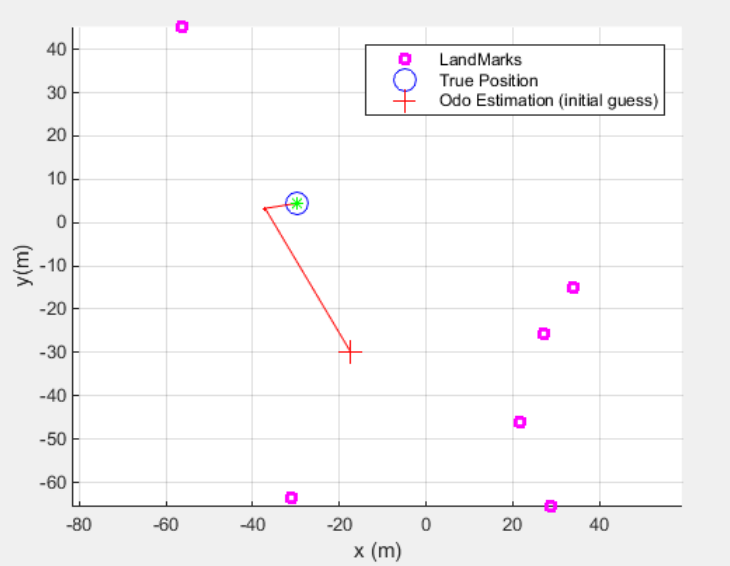

Figure 1: Example of execution of the code in *practice4_1.m*

3 Second task

- Which is the minimum number of landmarks needed for localizing the robot? Why?

We need at least 3 landmarks for localizing the robot univocally (one solution). In some cases, with 2 landmarks the solution it's not unique as indicated in figure 2. If just 1 landmark is observed, the problem gives infinitely many solutions (figure 3).

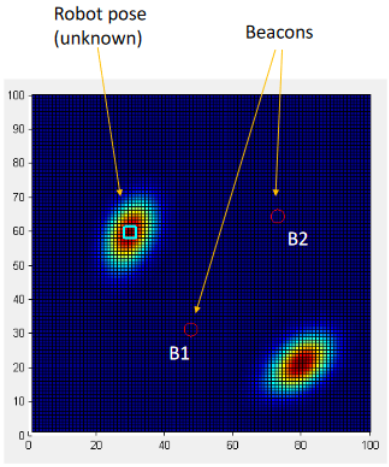

Figure 2: Estimated pose it's not unique

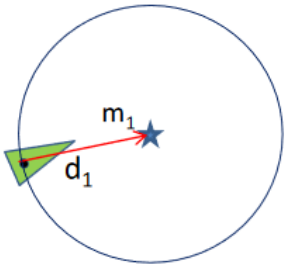

Figure 3: Any position of the circumference is a solution

4 Third task

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

Naturally, the greater the value of the variance, the more likely it is that the method will fail. In order to control the random nature of the call to `randn()`, it is included the command `rng('default')` for reproducibility:

```
rng('default');
r = randn();
z(kk) = z_ideal(kk) + r * var_d;
```

Doing this, `r` will always be same number in each iteration and in each execution of the complete program. It gives us chance to experiment with different values of `var_d` with more control and take better conclusions.

Also, if we examine the plot visually to determine if the method fails with a certain variance, we need to take into account the zoom we apply to the plot. So it's a bit subjective to assert that the method fails using a visual inspection without any quantitave control point.

For this reason, to decide if we have a good estimation or not, we compute the difference between `xTrue` and the last estimation `xEst` using the euclidean distance between them. We will take that if this distance $d > 0.5$, the estimation it's not good.

Assuming this, with `var_d` ≥ 1 and keeping the other variables the same as in the original template, it's not unsual to find $d > 0.5$.

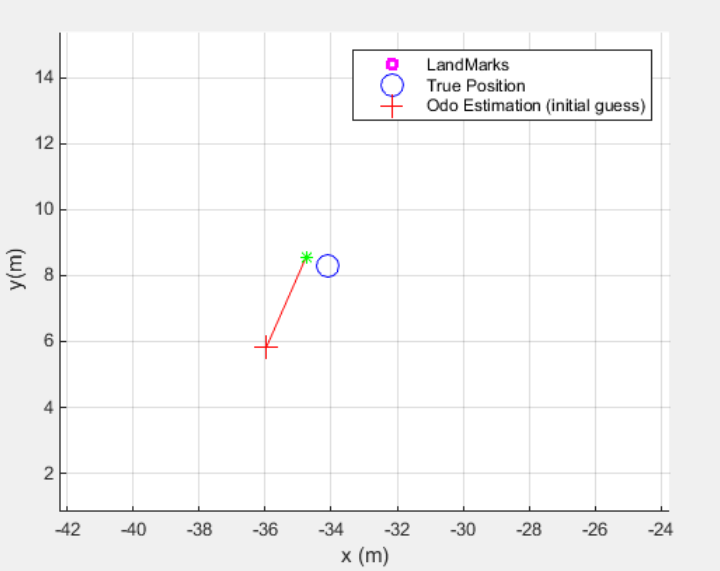

Figure 4: Plot resulting with a euclidean distance $d = 0.6543$ between `xTrue` and `xEst`

5 Fourth task

- Play also with different values for the odometry uncertainty.

Keeping the same conditions as in the previous exercise, except for `var_d` which it's setted to 0.5^2, it has been observed that the initial guess from odometry does not have much impact in the result as the quality of the range sensor. In fact, experimenting with a covariance of the odometry noise `U = diag([1000,170,15.5]).^2`, the euclidean distance between `xTrue` and the last estimation `xEst` is 0.1002, which it's not bad at all and it's similar to the result with `U = diag([9,20,1*pi/180]).^2`.

Figure 5 shows that the initial guess is very bad but the final localization is pretty good.

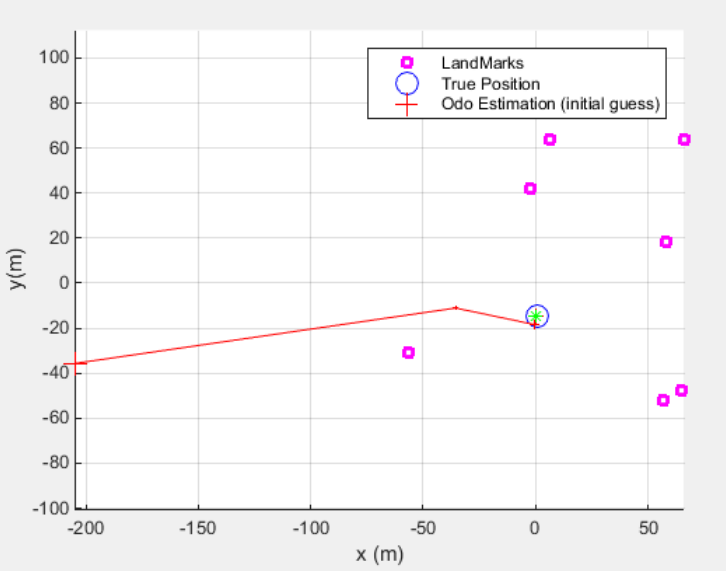

Figure 5: Bad initial guess from odometry but good true position estimation

6 Conclusions

Least Squares positioning is a method widely used in robot localization, and it behaves quite well. It is equivalent to Maximum Likelihood estimation if error is gaussian. *Gauss-Newton's* algorithm makes a good optimization in pose estimation.

We need at least 3 landmarks to obtain a unique solution. With 2 landmarks, sometimes, there may be more than one solution, and with 1 landmark there are infite solutions.

The quality of the range sensor is determinant in the estimation of the robot's pose, unlike the odometry uncertainty for initial guess.

Obviously, more landmarks and a good sensor will give us the best results but with only three landmarks and a sensor with a variance less than 1, we can achieve pretty acceptable estimations.

It can be interesting to elaborate a table which takes into account the effect of modify the different variables (number of landmarks, sensor error, odometry error ...) on the difference between the real position and estimated one.