CS-475 Assignment 2
Extended Kalman Filter

Michael Maravgakis
maravgakis@csd.uoc.gr

Release date: Monday 14/03/2022
Deadline: Thursday 24/03/2022, 23:59

1 Introduction

In this assignment you will implement the Extended Kalman Filter (EKF) for the turtlebot in order to estimate its state at each time step. As in the previous assignment the robot is always moving with a constant linear and angular velocity and thus moving in the circumference of a circle. This time the linear velocity is 0.5 m/s and the angular is 0.3 rad/s. In order to complete this assignment you'll need to fill the gaps marked as *** and then visualize and evaluate your results. If your implementation is correct, your output should look like this:

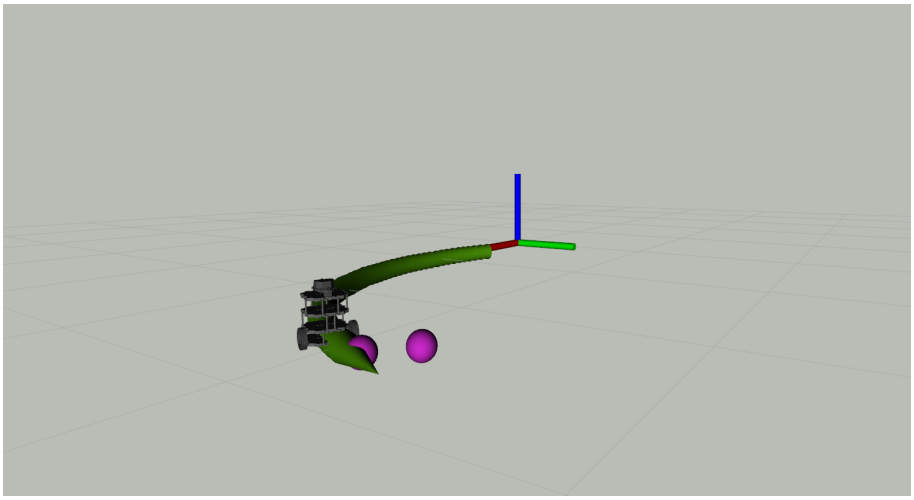


Figure 1: Expected result 1

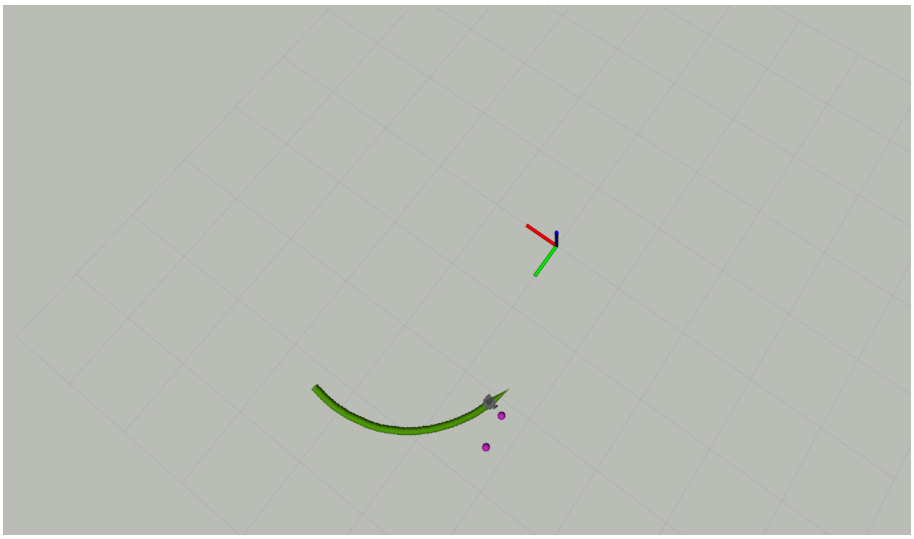
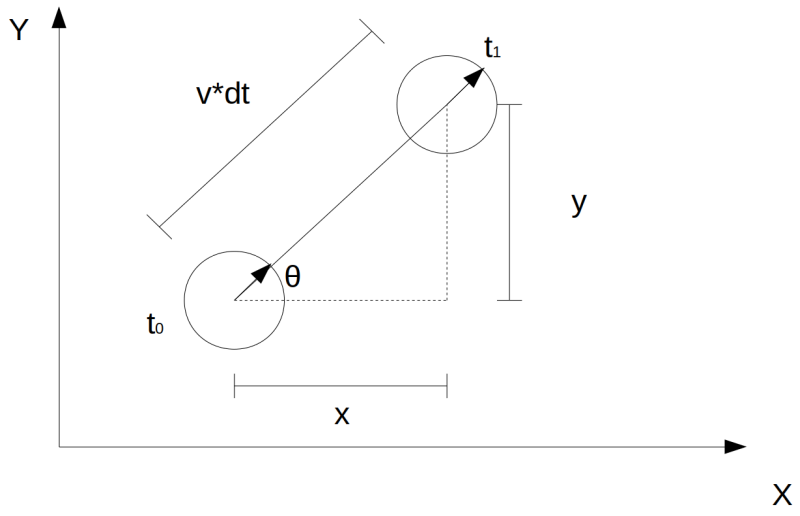


Figure 2: Expected result 2

2 Kalman Filter

Let's assume that we have a robot that is moving in a specific direction with constant velocity v. We know its position (x_0, y_0) at time t_0 and its direction and we want to predict it's position (x_1, y_1) at time t_1.



As it is obvious from the graph above, the new position will be:

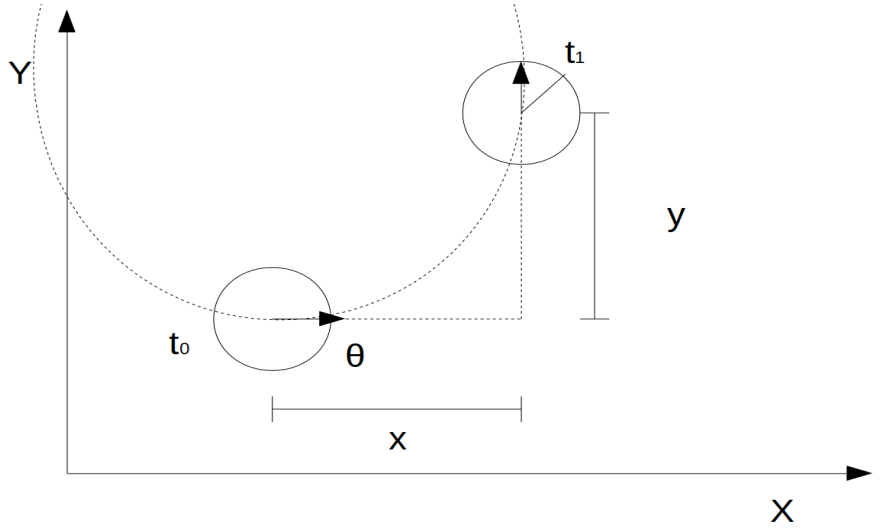
$$x_1 = x_0 + v \cdot dt \cdot cos\theta \tag{1}$$

$$y_1 = y_0 + v \cdot dt \cdot sin\theta \tag{2}$$

As you might noticed these are the equations we used in the previous assignment in order to predict the position of the turtlebot. This was an approximation with the purpose of making the trajectory of the robot linear. Kalman Filter also known as linear Kalman Filter works only in linear state transition situations and the trajectory of the previous assignment clearly wasn't one of them since it was moving in circles. Although, with low velocities and high sampling rate, one can approximate heuristically the trajectory of the robot as small linear consecutive movements.

3 Extended Kalman Filter (EKF)

In the real world, most systems are non-linear and that's where the EKF shines. It practically linearizes the non-linearity by using the Taylor expansion. In this assignment we are not going to do any approximations regarding the trajectory of the robot which is depicted in the following image:



The EKF algorithm with a similar notation to the one used inside the given code template is the following:

1. $X_t^{pred} = g(X^{(t-1)}, u)$
2. $\Sigma_t^{(pred)} = G_t \Sigma_{t-1} G_t^T + R_t$
3. $K_t = \Sigma_t^{(pred)} H_t^T (H_t \Sigma_t^{pred} H_t^T + Q_t)^{-1}$
4. $X_{est} = X_t^{pred} - K_t(z_t - h(X_t^{pred}))$
5. $\Sigma_{est} = (I - K_t H_t)\Sigma_t^{(pred)}$

g is a non linear function that represents the dynamic and non-linear evolution of the system. Σ is the covariance matrix that is named `Pest` in the template code. G_t is the Jacobian of g. R_t is called measurement noise matrix and contains the error of our inaccurate sensors. Q_t is called process noise and represents the error in the prediction due to various factors such as the robot slips or wind etc. EKF algorithm is almost the same as KF and thus the problem statement has been changed a bit to avoid repeatability and ensure that you understand key concepts behind these localization algorithms.
The state of our robot now is: `X = [x,y,dir,v,w].T`
where `dir` is the direction of the robot (yaw), `v` is the linear velocity and `w` is the angular velocity. This time, linear and angular velocities have noise and can be considered as both measurements and control inputs. Maybe this will get more clear with the following example, let's say that you monitor the pressure on the gas pedal of a car. The fact the driver might has pushed it all the way down doesn't mean that the car is actually accelerating with full thrust since it depends on the gear, the slope of the ground and other parameters. So if you also have a GPS or other measurements that indicate that the car is not moving you can estimate its actual velocity more precisely.

4 Implementation

By far the hardest part of this assignment is to figure out the state prediction function g, so this part of the assignment is a `BONUS` 10% in future mistakes in your exercises. It's pretty hard to figure out and requires a solid understanding in physics and math behind circular motion. Since the bonus part is at the beginning of the assignment and you can't carry on without the prediction step just send me an email at `maravgakis@csd.uoc.gr` with subject `"[CS-475] Give bonus"` and I will reply with the math equations that describe the motion of the robot so you can continue your implementation. Make sure not to waste a lot of time on it. In case you computed the matrix g you will need to test its correctness by removing the noise from all measurements and publishing the predictions. The result should be a perfect tracking of the robot's position.

To conclude, you have the turtlebot whose state is represented as its x and y coordinates, direction, linear and angular velocity. The noisy inputs that you are getting are: `Gps_x,Gps_y,Linear_vel,Angular_vel`. The implementation is almost the same as in the previous assignment, but now you have a different state and you'll have to compute the jacobians by hand.

5 Setup

In the .zip file that you've downloaded there is an `assign2/` folder, copy this to your workspace and compile using `catkin_make`. To start the simulator you'll have to open a terminal and run `$ roslaunch assign2 burger.launch`. Fi-

nally when your implementation is completed use `$ rosrun assign2 ekf.py` to run your node. (You might also need to make it executable by `$ chmod +x ekf.py`).

6 Tips

1. Finding g is hard, don't waste much time, this is not the purpose of the assignment
2. Comment blocks of code and add additional lines (e.g. return) in order to make sure that the steps you are making are correct.
3. The jacobians are very lengthy, I suggest computing each element separately and then combine them into one matrix
4. Feel free to change the given template code as you wish.
5. You should revisit your assignment 1 code since some parts are almost the same.

7 Submission

Sent your node (`ekf.py`) attached via email at: **maravgakis@csd.uoc.gr** with subject `"[CS-475] Assignment 2 submission"` Don't forget to mention your name and registration number. The deadline is at **Thursday 24/03/2022 23:59**