CS475 - Assignment 2

Extended Kalman Filter

Kalykakis Emmanouil
csdp1210@csd.uoc.gr

Release date: Sunday 12/03/2023
Deadline: Thursday 22/03/2023, 23:59

Introduction

In the previous assignment, we used the Kalman filter for an obviously non-linear problem. Kalman Filter also known as linear Kalman Filter, works only in linear state transition situations. 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.

In this assignment we want to implement the EKF for a (correctly modelled) non-linear problem. Knowing its closed form/analytical solution, we will be able to verify the EKF's superior performance. While the EKF in general does not guarantee optimality anymore (in comparison, the standard Kalman filter guarantees to obtain the minimum possible MSE for linear problems), the mere fact that it more accurately models non-linear dynamics results in much better performance.

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 on the circumference of a circle. This time the linear velocity is $v = 0.5\frac{m}{s}$ and the angular is $\omega = 0.3\frac{rad}{sec}$

EKF Algorithm

Prediction

$$\hat{\mathbf{x}}_t^- = g(\hat{\mathbf{x}}_{t-1}, u_t) \quad \text{Predicted state estimate (mean)}$$

$$\hat{P}_t^- = G_t P_{t-1} G_t^T + Q_t \quad \text{Predicted covariance estimate}$$

Correction

$$K_t = \hat{P}_t^- H_t^T \left(H_t \hat{P}_t^- H_t^T + R_t\right)^{-1} \quad \text{Kalman gain}$$

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + K_t\left(z_t - h(\hat{\mathbf{x}}_t^-)\right) \quad \text{Updated state estimate (mean)}$$

$$\hat{P}_t = (I - K_t H_t) P_t^- \quad \text{Updated covariance estimate}$$

x_t^- is the prior estimate, the rough estimate before the measurement update correction. g is a non-linear function that represents the dynamic and non-linear evolution of the system. P is the covariance matrix. R is called measurement noise matrix and contains the error of our inaccurate sensors. Q is called process noise and represents the error in the prediction due to various factors such as the robot slips or wind etc.. The state of our robot is $\mathbf{x} = [x, y, \theta, v, \omega]^T$. Where θ is the direction of the robot (yaw) with respect to the x-axis of the global reference frame $\{W\}$, v is the linear velocity and ω is the angular velocity.

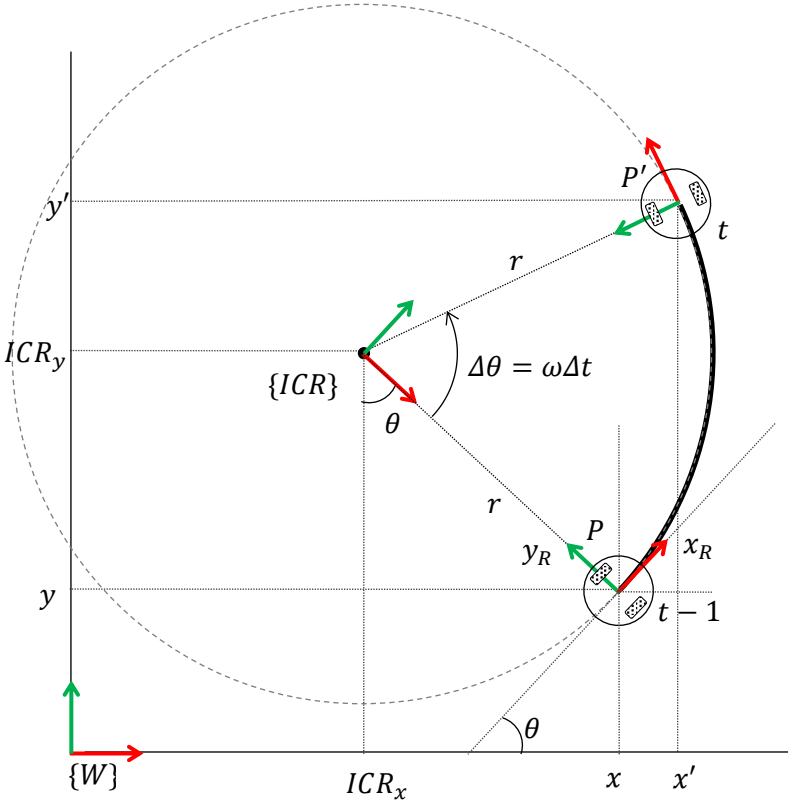
Matrices G and H which contain the (partial) Taylor expansions of g and h with respect to each of their parameters. Most literature only uses first order Taylor expansion (i.e. it aborts the Taylor expansion after the first term) which effectively turns G and H into **Jacobians** (a matrix

of partial derivatives of a vector-valued function with respect to all its parameters). So, G is the Jacobian of g with respect to $\mathbf{x}$.

This time, linear and angular velocities have noise and can be considered as both measurements and control inputs. Maybe this will get clearer with the following example. Let's say that you monitor the pressure on the gas pedal of a car. The fact that the driver might has pushed it all the way down doesn't mean that the car is 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.

Differential-steered vehicle kinematics

Turtlebot has two driven wheels and a front and back castor to provide stability. The robot steers by independently controlling the speed of the wheels on each side of the vehicle – if the speeds are not equal the vehicle will turn. The vehicle's velocity v is in the vehicle's x-direction, and zero in the y-direction since the wheels cannot slip sideways. The pose of the vehicle is represented by the body coordinate frame (local) shown in drawing, with its x-axis in the vehicle's forward direction and its origin at the centroid of the two driven wheels. The vehicle follows a curved path centered on the Instantaneous Center of Rotation (ICR) and the distance from the ICR to the origin of the robot's coordinate frame is $r = \frac{v}{\omega}$



The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by θ. We can describe the pose of the robot as vector with three elements $P = [x, y, \theta]^T$. The equations that specify the position P' of the robot on the plane with respect to the global reference frame, while it is moving on the circumference of a circle in time step Δt, are:

$$x' = x + \frac{v}{\omega}(sin(\theta + \omega\Delta t) - sin\theta)$$

$$y' = y - \frac{v}{\omega}(cos(\theta + \omega\Delta t) - cos\theta)$$

With the time interval Δt to be small enough.

The yaw angle using Taylor approximation within the same time step is:

$$\theta' = \theta + \dot{\theta}\Delta t = \theta + \omega\Delta t$$

While it is considered that $v' = v$ and $\omega' = \omega$.

The full motion model is given by the following equation:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \\ u_t \\ \omega_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \frac{v}{\omega}(\sin(\theta_{t-1} + \omega\Delta t) - sin\theta_{t-1}) \\ y_{t-1} - \frac{v}{\omega}(\cos(\theta_{t-1} + \omega\Delta t) - cos\theta_{t-1}) \\ \theta_{t-1} + \omega\Delta t \\ u_{t-1} \\ \omega_{t-1} \end{bmatrix}$$

The state prediction matrix g is constructed using the previous equation.

Implementation

In order to complete this assignment, you'll need to fill the gaps marked as ??? and then visualize and evaluate your results. The noisy inputs that you are getting are: **gps_x**, **gps_y**, **linear_velocity, angular_velocity**. 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. If you remove the noise from all measurements and publishing the predictions the result should be a perfect tracking of the robot's position. If your implementation is correct, your output should look like this:

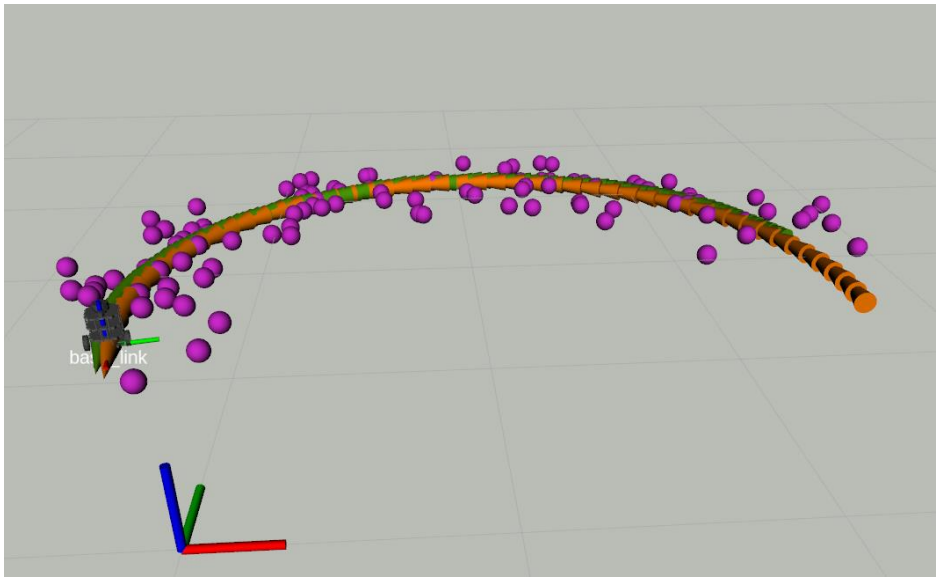


Figure 1 Screenshot captured from RViz. The green arrows represent the EKF estimates, the purple dots the noisy GPS measurements. The orange arrows represent the mostly accurate position and orientation of the robot.

Setup

In the zip file that you've downloaded, there is a **cs475_asgmt2** folder, copy this to the workspace (**/home/<user_name>/asgmt1_ws/src**) that we have created in the previous assignment and compile using **catkin_make** command. Open the file **run.sh** and change the package name cs475_asgmt1 into **cs475_asgmt2** in the **roslaunch** command if you want to load everything executing this shell script. Also you can start the simulator running the command **roslaunch cs475_asgmt2 burger.launch** since you firstly have exporting the variable TURTLEBOT3_MODEL (**export TURTLEBOT3_MODEL=burger**). When your implementation is completed, use **rosrun cs475_asgmt2 ekf.py** to run your node. You might also need to make it executable by **chmod u+x ekf.py**

Tips

1. Comment the blocks of the source code and add additional lines (e.g. return) in order to make sure that the steps you are making are correct.
2. The Jacobians are very lengthy, I suggest to compute each element separately and then to combine them into one matrix.
3. Feel free to change the given template code as you wish.
4. You should revisit your assignment 1 source code since some parts are almost the same.

Submission

Send your node (**ekf.py**) attached via email at: csdp1210@csd.uoc.gr Don't forget to add at the top of the file your name and your registration number as a comment. The subject of the email should be the following: **[CS475]: Assignment 2 submission**. The deadline is due to **Wednesday 22/03/2023 23:59**.