Robot Operating System (ROS)

Lab 7: Kalman Filters in ROS

Haitham El-Hussieny, PhD

December 20, 2022

Department of Mechatronics and Robotics Engineering
Egypt-Japan University of Science and Technology (E-JUST)
Alexandria, Egypt.

Outline

Recap of Kalman Filters

Linear Kalman Filter

A more general model:

Suppose we have a LTI system

$$\mathbf{x}_k = A\ \mathbf{x}_{k1} + B\ \mathbf{u}_k + \mathbf{w}_k$$
$$\mathbf{z}_k = C\ \mathbf{x}_k + \mathbf{v}_k$$

x_0 and P_0 are initialized first.

- A, B, C, and D are the system matrices.
- $\mathbf{w}_k$ and $\mathbf{v}_k$ are process and measurement noise with covariance $Q \in \mathbb{R}^{n_s \times n_s}$ and $R \in \mathbb{R}^{n_z \times n_z}$.
- n_s and n_z are number of states and measurements respectively.

1. Prediction of system state:

$$\hat{\mathbf{x}}_k = A\ \hat{\mathbf{x}}_{k-1} + B\ \mathbf{u}_k$$
$$P_k = A\ P_{k-1}\ A^T + Q$$

2. Update of system state:

$$G_k = P_k\ C^T(C\ P_k\ C^T + R)^{-1}$$
$$\hat{\mathbf{x}}_k \leftarrow \hat{\mathbf{x}}_k + G_k * (\mathbf{z}_k - C\ \hat{\mathbf{x}}_k)$$
$$P_k \leftarrow (I - G_k\ C)P_k$$

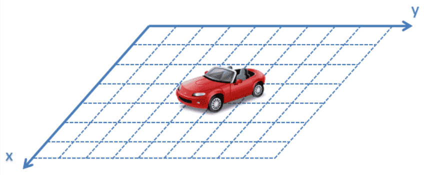
2D Object Tracking with Kalman Filter

2D Object Tracking with Kalman Filter

Problem Definition

2D Tracking

In this example, we would like to estimate the vehicle's location on the XY plane. The vehicle has an onboard location sensor that reports X and Y coordinates of the system. We assume constant acceleration dynamics.



2D Object Tracking with Kalman Filter

Problem Definition

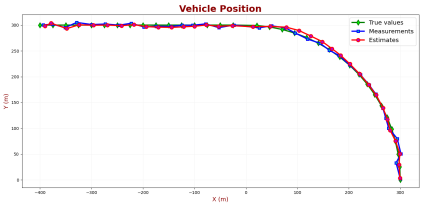
$$\begin{bmatrix} \hat{x}_{n+1} \\ \hat{\dot{x}}_{n+1} \\ \hat{\ddot{x}}_{n+1} \\ \hat{y}_{n+1} \\ \hat{\dot{y}}_{n+1} \\ \hat{\ddot{y}}_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_{n} \\ \hat{\dot{x}}_{n} \\ \hat{\ddot{x}}_{n} \\ \hat{y}_{n} \\ \hat{\dot{y}}_{n} \\ \hat{\ddot{y}}_{n} \end{bmatrix} \quad Q = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t & 0 & 0 & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ 0 & 0 & 0 & \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} \sigma_a^2$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad R_n = \begin{bmatrix} \sigma_{x_m}^2 & 0 \\ 0 & \sigma_{y_m}^2 \end{bmatrix}$$

2D Object Tracking with Kalman Filter

Results

```python
def kalman_estimate(x0, A, Q, P0, R, measurements, C):
    I = np.eye(x0.shape[0])
    x_hat_record = np.reshape(x0.T, (1,-1))
    for z in measurements.T:
        #ESTIMATE:-
        x_hat = A.dot(x0) # Motion model
        P = np.dot(np.dot(A, P0),np.transpose(A)) + Q # Prediciton uncertainty
        #UPDATE:-
        K = P.dot(C.T).dot(np.linalg.inv(C.dot(P).dot(C.T)+R)) # Kalman Gain
        x_hat = x_hat + K.dot((z - C.dot(x_hat))) # correction of estimate
        P = (I - K.dot(C)).dot(P) # corrrection of uncertainty
        x0 = x_hat
        P0 = P
        print(P)
        x_hat_record = np.append(x_hat_record, (np.reshape(x_hat.T, (1,-1))), axis=0)
    return x_hat_record
```



Download the Package

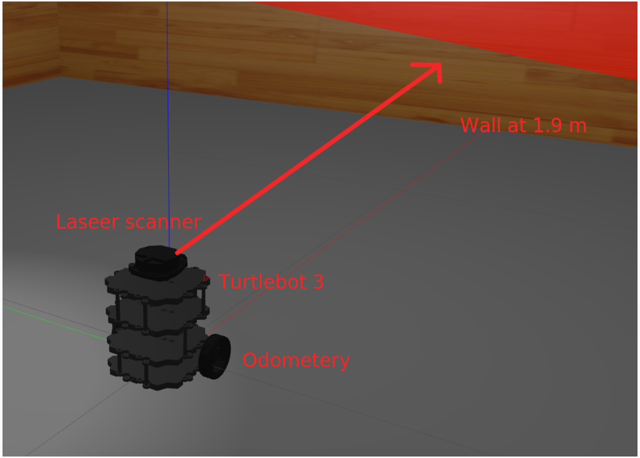
1-D Localization with Laser Scanner

1-D Localization with Laser Scanner

Problem Definition

1-D Robot Localization

The task aims to localize the x-direction of the Turtlebot 3 robot based on the encoder readings received on the topic /odom and using the laser scanner readings received on the topic /scan.



1-D Localization with Laser Scanner

Installation of Gazebo Simulation

- Install TurtleBot3 via Debian Packages.

```
$ sudo apt install ros-noetic-dynamixel-sdk
$ sudo apt install ros-noetic-turtlebot3-msgs
$ sudo apt install ros-noetic-turtlebot3
```

- Install Turtlebot3 simulation

```
$ cd ~/catkin_ws/src/
$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
$ cd ~/catkin_ws

$ catkin_make
```

1-D Localization with Laser Scanner

Installation of Gazebo Simulation

- Run the gazebo simulator

```
$ roslaunch turtlebot3_gazebo turtlebot3_stage_1.launch
```

- Run your 1-D Kalman node

```
$ rosrun kalman_tracking turtle3_localize.py
```

- Publish a command velocity on the `cmd_vel` topic.
- If you need to reset the robot's position

```
$ rosservice call /gazebo/reset_simulation
```

1-D Localization with Laser Scanner

Laser Scanner Data Callback

```
def laser_ray_recieved(msg):
    obstacle_front_x_axis = 1.925000
    global robot_x_ray
    # keep the minimum distance reading from 10 rays pointing to the front
    # second min is required to filter out 'inf' values, in that case 12 is used
    front_laser_ray = min(min(msg.ranges[0:10]), 12)
    front_laser_ray = np.random.normal(front_laser_ray, .3,1)[0]
    #rospy.loginfo("Distance to object in front (front_laser_ray): %s", front_laser_ray)
    # calculate robot position in the world considering the known position of an obstacle in front
    # example: position of obstacle: 10, laser_ray_reading = 8  => robot_x_ray = 2
    # This assumes/requires a robot moving straigt and parallel to x-axis, with orientation = [0,0,0,1]
    robot_x_ray = obstacle_front_x_axis - front_laser_ray
    #rospy.loginfo("X position in map frame (robot_x_ray): %s", robot_x_ray)
    #print(robot_x_ray)
```

1-D Localization with Laser Scanner

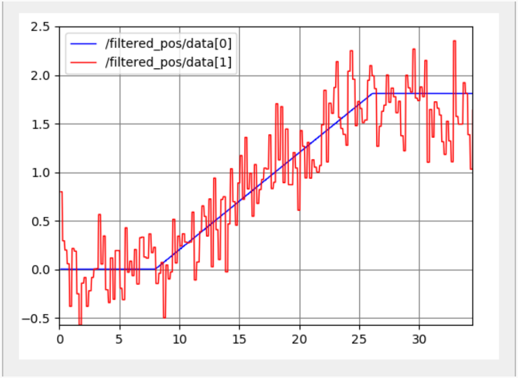
Linear Kalman Filter: Reception of Odometery

```
def odometry_recieved(msg):
    #ESTIMATE:-
    global A, P_0, Q, R, C, I, robot_x_ray

    x_robot = np.array([msg.pose.pose.position.x, msg.twist.twist.linear.x, 0])
    print("Robot position{}".format(x_robot))
    P = (A.dot(P_0).dot((A.T)))#+ Q # Prediciton uncertainty
    #UPDATE:-
    K = P.dot((C.T)).dot(np.reciprocal((C.dot(P).dot((C.T)))+R)) # Kalman Gain
    x_robot = x_robot + K.dot((robot_x_ray - C.dot(x_robot))) # correction of estimate
    P = (I - K.dot(C)).dot(P)#.dot((I - K.dot(C)).T) + K.dot(R).dot((K.T)) # corrrection of uncertainty
    print(P)
    #print(x_robot)
    P_0 = P
    robot_filtered_x.x = x_robot[0]
    robot_filtered_x.y = robot_x_ray
    filtered_pos_pub.publish(robot_filtered_x)
```

1-D Localization with Laser Scanner

Result: Moving in a Straight Line



End of Lecture

haitham.elhussieny@ejust.edu.eg