

Record For Implementing KF on Single Movement

Mingze CHEN

June 2, 2021

1 Implementing One Dimensional Kalman Filter

After connecting to the robot through *ssh*, then execute the following command:

```
1 husarion@husarion: $ roscore
```

start 2nd. command line window and **execute following command**:

```
1 $ roslaunch rosbot_ekf all.launch rosbot_pro:=true
```

start 3rd. command line window and launch the *robot_localization* through executing:

```
1 ~/pathTo/catkin_ws$ source ./devel/setup.bash
2 ~/pathTo/catkin_ws$ roslaunch playground start_filter.launch
```

Now our purpose is implementing *KF* algorithm into existing *move.py* script.

To launch the *UWB tag* through executing:

```
1 ~/pathTo/catkin_ws$ source ./devel/setup.bash
2 ~/pathTo/catkin_ws$ roslaunch localizer-dwm1001 dwm1001.launch
```

In Kalman filter theory, the most important part is **Status update equation**:

The Kalman expression or **status update equation** is:

$$\begin{aligned} \text{Current state estimated value} &= \text{Predicted value of current state} + \\ &\text{Kalman Gain} * (\text{measured value} - \text{predicted value of the state}) \end{aligned}$$

which is:

$$\hat{X}(t) = X_p(t) + K \times [X_m(t) - X_p(t)] \quad (1)$$

where $K = \frac{\sigma_p^2}{\sigma_p^2 + \sigma_m^2}$

One sample implementation ¹ can be like:

```
1 from collections import namedtuple
2 gaussian = namedtuple('Gaussian', ['mean', 'var'])
3 gaussian.__repr__ = lambda s: '({:.3f}, {:.3f})'.format(
4     s[0], s[1])
5
6 def update(prior, measurement):
7     x, P = prior          # mean and variance of prior
8     z, R = measurement    # mean and variance of measurement
9
10    y = z - x              # residual
11    K = P / (P + R)        # Kalman gain
12
13    x = x + K*y            # posterior
14    P = (1 - K) * P        # posterior variance
15    return gaussian(x, P)
16
17 def predict(posterior, movement):
18     x, P = posterior      # mean and variance of posterior
19     dx, Q = movement     # mean and variance of movement
20     x = x + dx
21     P = P + Q
22     return gaussian(x, P)
```

Inside the *playground* package there's a **Python** script called *kalman_filter.py*.

Its code snippet is here:

```
1 def predict_step(mean1, var1, mean2, var2):
2     global new_mean, new_var
3     new_mean = mean1 + mean2
```

¹<https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/04-One-Dimensional-Kalman-Filters.ipynb>

```

4     new_var = var1 + var2
5     return new_mean, new_var
6
7 # correct step function
8 def correct_step(mean1, var1, mean2, var2):
9     """
10    This function takes in two means and two squared
        variance terms, and return updated gaussian
        parameters.
11    """
12    # calculate the new gaussian parameters
13    new_mean = (var1 * mean2 + var2 * mean1) / (var1 + var2
14    )
15    # also equals to var1 * var2 / (var1 + var2)
16    new_var = 1 / (1 / var1 + 1 / var2)
17    return new_mean, new_var

```

Although no variable called `kalman_gain` was calculated explicitly, through mathematical derivation we can know, they're the same.

Now execute:

```

1 $ husarion@husarion:~/pathTo/catkin_ws$ rosrun playground
    kalman_filter.py -s kf3105.csv

```

The running process was: we let the robot move forward 1m, R and Q are derived from extensive experiments:

```

1 process_var = 0.028 ** 2
2 sensor_var = 0.077 ** 2

```

Afterwards, position data based on calculation and *KF* are collected in a file called *kf3105.csv* and after plotting, we get Fig. 1 and Fig. 2:

Implemented KF tracks the position of tag so closely after convergence that the measurement error from UWB tag affect the KF result which is not expected.

Our calculation process is:

1. Initial position was calculated based on *UWB*
2. Afterwards, every small step was calculated based on an *internal EKF* with only *IMU* and *Odometry* as input and an external *KF* with fused output from the *EKF* and *UWB tag*

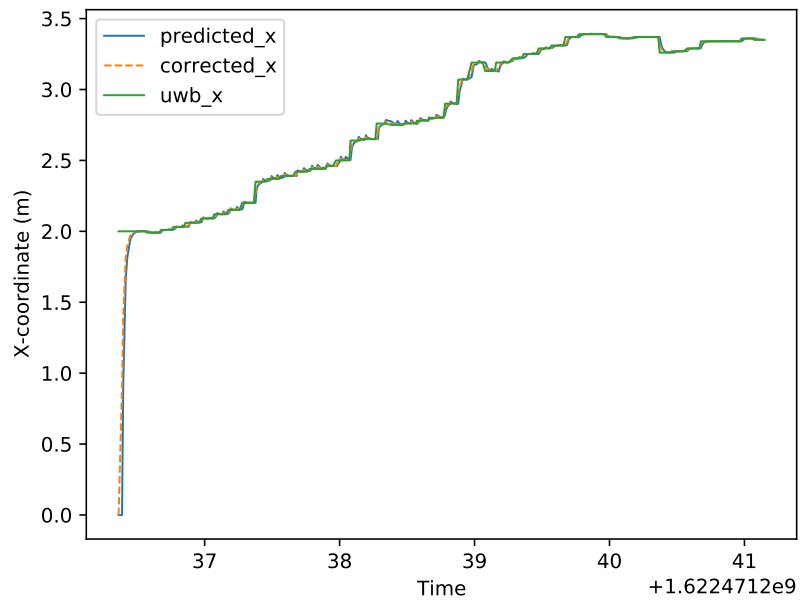


Figure 1: Plot from script *kfMergePlot.py*

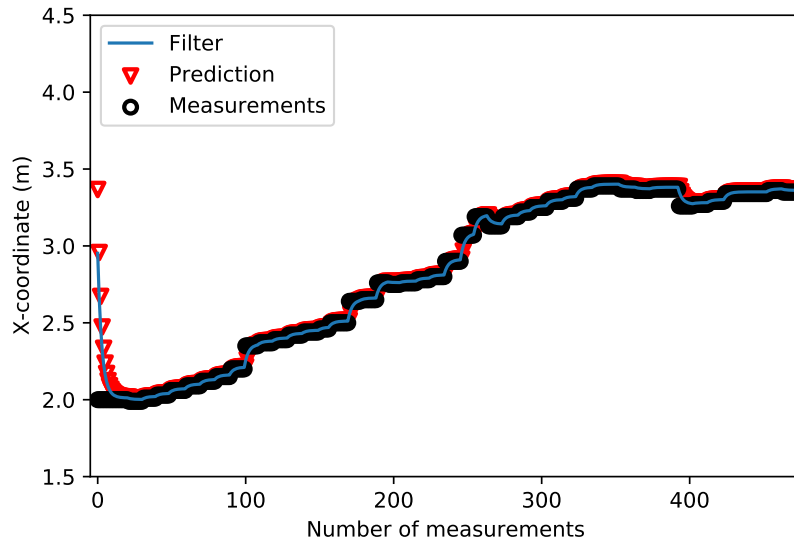


Figure 2: Plot from *Kalman Filter.ipynb*

The computation of KF affects the efficiency of robot movement logic more or less, so the better solution is:

1. moving the robot
2. gathering all necessary data in a csv file
3. conducting sensor fusion algorithm

2 Idea for next step

1. Extend one dimensional KF to multi dimension/variables KF, UKF and EKF
2. Consider designing a random walk model for the final demonstration