# 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:

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

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

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

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

```
~/pathTo/catkin_ws$ source ./devel/setup.bash
~/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:

```
~/pathTo/catkin_ws$ source ./devel/setup.bash
~/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:

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

which is:

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

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

One sample implementation [1] can be like:

```python
from collections import namedtuple
gaussian = namedtuple('Gaussian', ['mean', 'var'])
gaussian.__repr__ = lambda s: '(={:.3f}, 2={:.3f})'.format(
    s[0], s[1])

def update(prior, measurement):
    x, P = prior         # mean and variance of prior
    z, R = measurement   # mean and variance of measurement

    y = z - x        # residual
    K = P / (P + R)  # Kalman gain

    x = x + K*y      # posterior
    P = (1 - K) * P  # posterior variance
    return gaussian(x, P)

def predict(posterior, movement):
    x, P = posterior # mean and variance of posterior
    dx, Q = movement # mean and variance of movement
    x = x + dx
    P = P + Q
    return gaussian(x, P)
```

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

Its code snippet is here:

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

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

```python
    new_var = var1 + var2
    return new_mean, new_var

# correct step function
def correct_step(mean1, var1, mean2, var2):
    """
    This function takes in two means and two squared variance terms, and return updated gaussian parameters.
    """
    # calculate the new gaussian parameters
    new_mean = (var1 * mean2 + var2 * mean1) / (var1 + var2)
    # also equals to var1 * var2 /(var1  +  var2)
    new_var = 1 / (1 / var1 + 1 / var2)
    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:

```
$ 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:

```
process_var = 0.028 ** 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. A bigger sensor measurement variance can make the localisation trajectory smoother, which is closer to the real movement process, as shown in Fig. 3 :

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

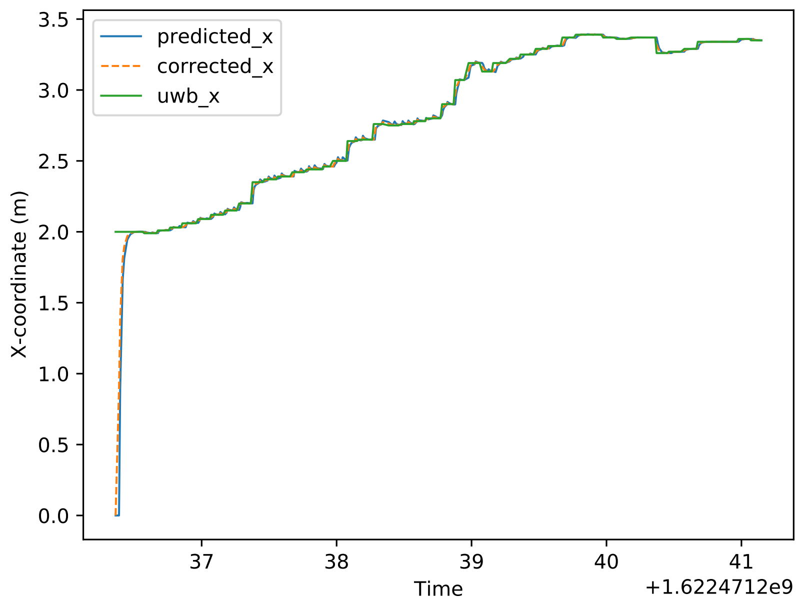


Figure 1: Plot from script *kfMergePlot.py*

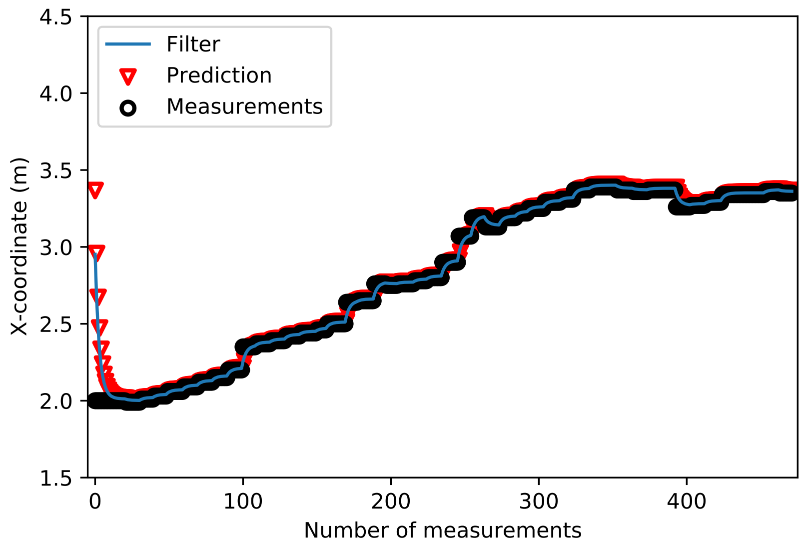


Figure 2: Plot when sensor_var = 0.077 ** 2

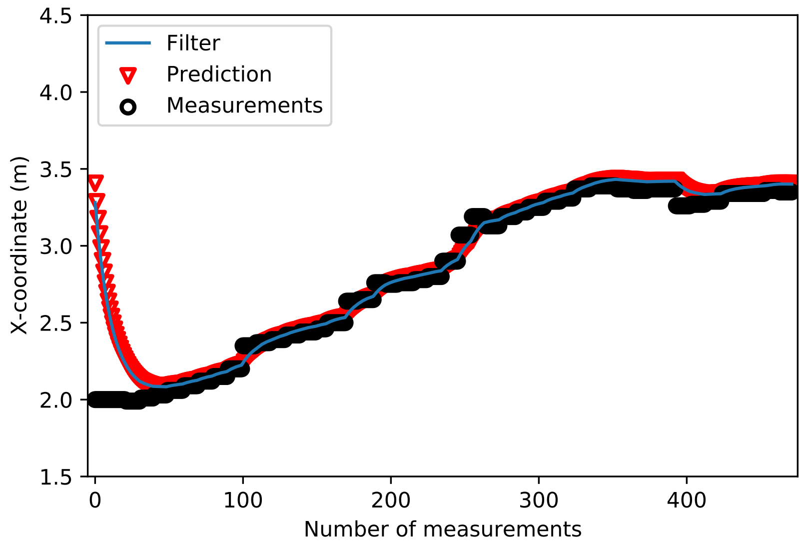


Figure 3: Plot when sensor_var = 0.3 ** 2

output from the *EKF* and *UWB tag*

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