Paper review: A Generalized Extended Kalman Filter Implementation for the Robot Operating System[1]

Hsiu-Wen Yen A59010599 hsyen@ucsd.edu

I. Introduction

Localization is always an important part of robotics. All decision made by robots are based on its location. For example, in path-tracking robot needs to know how far it is from the desired path, and in path-finding robot needs to know its location to apply any path searching algorithms like A*.

Basically, sensor is the only way to measure location, however, it is not always accurate or sufficient to get location. For example, gyro has drift problem, so gyro will tell you the robot is still moving even though it is stationary, and wheel encoder can only get linear velocity in robot frame which isn't enough to know when it turns. Therefore, we use multiple sensors to compensate for their disadvantages.

robot_localization pkg introduced in the paper [1] is widely used in ROS to do localization with sensor fusion. It can deal with a bunch of different type of sensors and handles the asynchronous problem between each sensor. The main purpose of this report is to discuss how does ***robot_localization pkg*** work.

II. Bayes Filter, Kalman Filter, EKF

The cornerstone of ***robot_localization pkg*** is EKF (extended Kalman filter). EKF is derived from Kalman Filter, and Kalman Filter is derived from Bayes Filter, so we start with Bayes filter.

A. Bayes Filter

First, we need to know the structure of robotics problems. $\boldsymbol{t}$ is time, $\boldsymbol{x_t}$ is robot state at time $\boldsymbol{t}$, $\boldsymbol{u_t}$ is control input at time $\boldsymbol{t}$, $\boldsymbol{z_t}$ is observation at time $\boldsymbol{t}$. For finite time horizon (up to time T) we only know $\boldsymbol{u_{0:T}}$ and $\boldsymbol{z_{0:T}}$ (from sensors), we don't know $\boldsymbol{x_{1:T}}$ (suppose $\boldsymbol{x_0}$ is given). Therefore, our problem is to know $\boldsymbol{x_{1:T}}$, and we introduce **Markov assumption** here. **Markov assumption** states $\boldsymbol{x_{t+1}}$ only depends on ($\boldsymbol{x_t}$, $\boldsymbol{u_t}$) and $\boldsymbol{z_t}$ only depends on $\boldsymbol{x_t}$. However, we know that data from sensors is always noisy, we can quantify the noise to get

$$x_{t+1} = f(x_t, u_t, w_t) \sim P_f(x_{t+1}|x_t, u_t) \quad (1)$$

$$z_t = h(x_t, v_t) \sim P_h(z_t|x_t) \quad (2)$$

where $\boldsymbol{w_t}$ is control noise, f is state transition function (motion model), P_f is a probability distribution due to uncertainty of $\boldsymbol{w_t}$, $\boldsymbol{v_t}$ is observation noise, h is observation model, P_h is a probability distribution due to uncertainty of $\boldsymbol{v_t}$. Recall that we are interested in $\boldsymbol{x_{1:T}}$, which is finding

$$P(x_t|z_{0:t}, u_{0:t}) \ \forall t \quad (3)$$

Denote (3) as $P_{t|t}(x_t)$ for simplicity.

Then, derivation for $P_{t+1|t+1}(x_{t+1})$ is

$$P_{t+1|t+1}(x_{t+1}) = P(x_{t+1}|z_{0:t+1}, u_{0:t+1})$$
$$\overset{Markov}{\Longrightarrow} P(x_{t+1}|z_{0:t+1}, u_{0:t}) = P(x_{t+1}|z_{t+1}, z_{0:t}, u_{0:t})$$
$$\overset{Bayes}{\Longrightarrow} \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}, z_{0:t}, u_{0:t}) P(x_{t+1}|z_{0:t}, u_{0:t})$$
$$\overset{Markov}{\Longrightarrow} \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) P(x_{t+1}|z_{0:t}, u_{0:t})$$
$$\overset{Total\ prob.}{\Longrightarrow} \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) \int P(x_{t+1}, x_t|z_{0:t}, u_{0:t}) dx_t$$
$$\overset{Conditional\ prob.}{\Longrightarrow} \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) * \int P(x_{t+1}|z_{0:t}, u_{0:t}, x_t) P(x_t|z_{0:t}, u_{0:t}) dx_t$$
$$\overset{Markov}{\Longrightarrow} \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) \int P(x_{t+1}|x_t, u_t) P(x_t|z_{0:t}, u_{0:t}) dx_t$$

$$\Rightarrow \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) \int P(x_{t+1}|x_t, u_t) P_{t|t}(x_t) dx_t$$

where $\eta_{t+1} = P(z_{t+1}|z_{0:t}, u_{0:t})$, and we define predicted pdf as $P_{t+1|t}(x_{t+1})$ with

$$P_{t+1|t}(x_{t+1}) = \int P(x_{t+1}|x_t, u_t) P_{t|t}(x_t) dx_t \quad (4)$$

and updated pdf as $P_{t+1|t+1}(x_{t+1})$ with

$$P_{t+1|t+1}(x_{t+1}) = \frac{1}{\eta_{t+1}} P(z_{t+1}|x_{t+1}) P_{t+1|t}(x_{t+1}) \quad (5)$$

Observe that computing $P(x_T|z_{0:T}, u_{0:T})$ is to alternatively compute $P_{t+1|t}(x_{t+1})$ (called prediction step) and $P_{t+1|t+1}(x_{t+1})$ (called update step) from $t = 0$ to $t = T$. *in the paper [1] correction step is referred to update step.

B. Kalman Filter

Kalman Filter is Bayes filter with some assumptions. So, we can simplify prediction step and update step. It assumes that (1) prior distribution is gaussian (2) motion and observation model is linear in x_t with gaussian noise (3) noise and state are independent. With assumptions we get

Asm.1 $$P(x_0) \sim N(\mu_0, \Sigma_0)$$

Asm.2 $$w_t \sim N(0, W)$$
Asm.2 $$f(x_t, u_t, w_t) = Fx_t + Gu_t + w_t$$
Asm.3 $$P_f(x_{t+1}|x_t, u_t) \sim N(Fx_t + Gu_t, w)$$

Asm.2 $$v_t \sim N(0, v)$$
Asm.2 $$h(x_t, v_t) = Hx_t + v_t$$
Asm.3 $$P_h(z_t|x_t) \sim N(Hx_t, V)$$

where F, G, and H are linear operator which can be time dependent.

Recall prediction step

$$P_{t+1|t}(x_{t+1}) = \int P(x_{t+1}|x_t, u_t) P_{t|t}(x_t) dx_t$$

Now, both of $P(x_{t+1}|x_t, u_t)$ and $P_{t|t}(x_t)$ are gaussian distribution, and convolution of two gaussian is also gaussian which is proved by [3]. Therefore, we can directly find its mean and covariance to know the distribution.

$$\mu_{t+1|t} = E[x_{t+1}] = E[Fx_t + Gu_t + w_t]$$
$$= E[Fx_t] + E[Gu_t] + E[w_t]$$
$$= F * E[x_t] + Gu_t + 0$$
$$= F\mu_{t|t} + Gu_t$$

$$\Sigma_{t+1|t} = Var[x_{t+1}] = Var[Fx_t + Gu_t + w_t]$$
$$= Var[Fx_t] + Var[Gu_t] + Var[w_t]$$
$$= E\left[F\left(x_t - \mu_{t|t}\right)\left(x_t - \mu_{t|t}\right)^T F^T\right] + 0 + W$$
$$= F\Sigma_{t|t}F^T + W$$

As a result, we get

$$P_{t+1|t}(x_{t+1}) \sim N(F\mu_{t|t} + Gu_t, F\Sigma_{t|t}F^T + W)$$
$$= N(\mu_{t+1|t}, \Sigma_{t+1|t}) \quad (6)$$

For update step, we know that $\boldsymbol{z_{t+1}}$ and $\boldsymbol{x_{t+1}}$ are individually gaussian and they are jointly gaussian (didn't prove here, but true). Therefore, we can find $P(x_{t+1}|z_{t+1})$ by conditional distribution of jointly gaussian.

For example,

$$X = \begin{pmatrix} X_A \\ X_B \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_A \\ \mu_B \end{pmatrix}, \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}\right)$$
$$P(X_A|X_B) \sim N(\mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B - \mu_B), \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA})$$

Plug $\boldsymbol{x_{t+1}}$ and $\boldsymbol{z_{t+1}}$ into equations above, we get

$$P(x_{t+1}, z_{t+1}) = \begin{pmatrix} x_{t+1} \\ z_{t+1} \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_{t+1|t} \\ m \end{pmatrix}, \begin{bmatrix} \Sigma_{t+1|t} & C \\ C^T & S \end{bmatrix}\right)$$

$$m = E[z_{t+1}] = E[Hx_{t+1} + v_{t+1}] = H * E[x_{t+1}] + E[v_{t+1}]$$
$$= H\mu_{t+1|t} + 0 = H\mu_{t+1|t}$$

$$S = Var[z_{t+1}] = Var[Hx_{t+1} + v_{t+1}]$$
$$= Var[Hx_{t+1}] + Var[v_{t+1}] = H\Sigma_{t+1|t}H^T + V$$

$$C = Cov(x_{t+1}, z_{t+1})$$
$$= E[\left(x_{t+1} - \mu_{t+1|t}\right)\left(z_{t+1} - H\mu_{t+1|t}\right)^T]$$
$$= E[\left(x_{t+1} - \mu_{t+1|t}\right)\left(Hx_{t+1} + v_{t+1} - H\mu_{t+1|t}\right)^T]$$
$$= E[\left(x_{t+1} - \mu_{t+1|t}\right)\left(x_{t+1} - \mu_{t+1|t}\right)^T H^T]$$
$$= E\left[\left(x_{t+1} - \mu_{t+1|t}\right)\left(x_{t+1} - \mu_{t+1|t}\right)^T\right] H^T$$
$$= \Sigma_{t+1|t}H^T$$

As a result,

$$P(x_{t+1}|z_{t+1}) \sim N(\mu_{t+1|t+1}, \Sigma_{t+1|t+1})$$

$$\mu_{t+1|t+1} = \mu_{t+1|t} + K\left(z_{t+1} - H\mu_{t+1|t}\right)$$
$$\Sigma_{t+1|t+1} = (I - KH)\Sigma_{t+1|t} \quad (7)$$
$$K = \Sigma_{t+1|t}H^T\left(H\Sigma_{t+1|t}H^T + V\right)^{-1}$$

*K is usually called Kalman gain.

C. Extended Kalman Filter (EKF)

EKF is used for non-linear motion model and observation model. The idea is using first-order Tyler series to approximate non-linear models to apply Kalman Filter. Recall Tyler series approximation

$$f(x) = f(a) + f'(a)(x - a) \quad (8)$$

For non-linear motion model, we can approximate it as

$$f(x_t, u_t, w_t) \approx f\left(\mu_{t|t}, u_t, 0\right) + \left[\frac{\partial f}{\partial x_t}\left(\mu_{t|t}, u_t, 0\right)\right](x_t - \mu_{t|t})$$
$$+ \left[\frac{\partial f}{\partial w_t}\left(\mu_{t|t}, u_t, 0\right)\right](w_t - 0) \quad (9)$$

$$h(x_{t+1}, v_{t+1}) \approx h\left(\mu_{t|t}, 0\right) + \left[\frac{\partial h}{\partial x_{t+1}}\left(\mu_{t+1|t}, 0\right)\right](x_{t+1} -$$
$$\mu_{t+1|t}) + \left[\frac{\partial h}{\partial v}\left(\mu_{t+1|t}, 0\right)\right](v_{t+1} - 0) \quad (10)$$

Let $F_t = \frac{\partial f}{\partial x_t}(\mu_{t|t}, u_t, 0)$ and $Q_t = \frac{\partial f}{\partial w_t}(\mu_{t|t}, u_t, 0)$, then

$$f(x_t, u_t, w_t) \approx f(\mu_{t|t}, u_t, 0) + F_t(x_t - \mu_{t|t}) + Q_t w_t$$

with this to recompute (6), get

$$\begin{aligned}
&\mu_{t+1|t} = \int\int [f(\mu_{t|t}, u_t, 0) + F_t(x_t - \mu_{t|t}) + Q_t w_t] \\
&\phi(x_t; \mu_{t|t}, \Sigma_{t|t})\phi(w_t; 0, W) dx_t dw_t \\
&= f(\mu_{t|t}, u_t, 0) + F_t\left(\int x_t \phi(x_t; \mu_{t|t}, \Sigma_{t|t}) dx_t - \mu_{t|t}\right) \\
&+ Q_t \int w_t \, \phi(w_t; 0, W) dw_t \\
&= f(\mu_{t|t}, u_t, 0) + F_t(E[x_t] - \mu_{t|t}) + Q_t(E[w_t]) \\
&= f(\mu_{t|t}, u_t, 0)
\end{aligned}$$

$$\begin{aligned}
&\Sigma_{t+1|t} = \int\int [f(\mu_{t|t}, u_t, 0) + F_t(x_t - \mu_{t|t}) + Q_t w_t] \\
&[f(\mu_{t|t}, u_t, 0) + F_t(x_t - \mu_{t|t}) + Q_t w_t]^T \phi(x_t; \mu_{t|t}, \Sigma_{t|t}) \\
&\phi(w_t; 0, W) dx_t dw_t - \mu_{t+1|t}\mu_{t+1|t}^T \\
&= f(\mu_{t|t}, u_t, 0)\left(\int (x_t - \mu_{t|t})^T \phi(x_t; \mu_{t|t}, \Sigma_{t|t}) dx_t\right) F_t^T + \\
&F_t\left(\int (x_t - \mu_{t|t})\, \phi(x_t; \mu_{t|t}, \Sigma_{t|t}) dx_t\right) f(\mu_{t|t}, u_t, 0)^T + \\
&F_t\left(\int (x_t - \mu_{t|t})(x_t - \mu_{t|t})^T \phi(x_t; \mu_{t|t}, \Sigma_{t|t}) dx_t\right) F_t^T + \\
&Q_t\left(\int w_t w_t^T \, \phi(w_t; 0, W)\right) Q_t^T \\
&= F_t \Sigma_{t|t} F_t^T + Q_t W Q_t^T
\end{aligned}$$

Let $H_{t+1} = \frac{\partial h}{\partial x_{t+1}}(\mu_{t+1|t}, 0)$ and $R_{t+1} = \frac{\partial h}{\partial v_{t+1}}(\mu_{t+1|t}, 0)$, then

$$\begin{aligned}
&h(x_{t+1}, v_{t+1}) \approx h(\mu_{t+1|t}, 0) + H_{t+1}(x_{t+1} - \mu_{t+1|t}) \\
&+ R_{t+1} v_{t+1}
\end{aligned}$$

with this to recompute (7), get

$$\begin{aligned}
&m = \\
&\int\int h(x_{t+1}, v_{t+1})\, \phi(x_{t+1}; \mu_{t+1|t}, \Sigma_{t+1|t})\phi(v_{t+1}; 0, V) dx_{t+1} dv_{t+1} \\
&\approx h(\mu_{t+1|t}, 0)
\end{aligned}$$

$$\begin{aligned}
&S = \int\int (h(x_{t+1}, v_{t+1}) - m)\,(h(x_{t+1}, v_{t+1}) - m)^T \\
&\phi(x_{t+1}; \mu_{t+1|t}, \Sigma_{t+1|t})\phi(v_{t+1}; 0, V) dx_{t+1} dv_{t+1} \\
&\approx H_{t+1}\Sigma_{t+1|t}H_{t+1}^T + R_{t+1} V R_{t+1}^T
\end{aligned}$$

$$\begin{aligned}
&C = \int\int (x_{t+1} - \mu_{t+1|t})(h(x_{t+1}, v_{t+1}) - m)^T \\
&\phi(x_{t+1}; \mu_{t+1|t}, \Sigma_{t+1|t})\phi(v_{t+1}; 0, V) dx_{t+1} dv_{t+1} \\
&\approx \Sigma_{t+1|t} H_{t+1}^T
\end{aligned}$$

The conditional gaussian distribution of $x_{t+1}|z_{t+1}$ is

$$\begin{aligned}
&\mu_{t+1|t+1} = \mu_{t+1|t} + K(z_{t+1} - h(\mu_{t+1|t}, 0)) \\
&\Sigma_{t+1|t+1} = (I - KH_{t+1})\Sigma_{t+1|t} \\
&K = \Sigma_{t+1|t} H_{t+1}^T (H_{t+1}\Sigma_{t+1|t}H_{t+1}^T + R_{t+1} V R_{t+1}^T)^{-1}
\end{aligned} \tag{11}$$

III. Thectival Approches

A. State

robot_localization pkg keeps track of all states of robot .

$$\boldsymbol{x} = [x, y, z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, a_x, a_y, a_z, \alpha_x, \alpha_y, \alpha_z]^T \tag{12}$$

where x are robot position in world frame, v_x is velocity along x axis in robot frame, ω_x is angular velocity around x axis in robot frame, a_x is acceleration along x axis in robot frame, α_x is angular acceleration around x axis in robot frame. Same for y and z.

B. Motion model

robot_localization pkg uses generic omnidirectional motion model [2], which assumes that velocity and acceleration are constant over small-time intervals t. We get

$$x_{t+1} = x_t + \overline{v_{x_t}} * t + 0.5 * \overline{a_{x_t}} * t^2$$

$$v_{x_{t+1}} = v_{x_t} + a_{x_t} * t$$

$$\omega_{x_{t+1}} = \omega_{x_t} + \alpha_{x_t} * t$$

$$a_{x_{t+1}} = a_{x_t}$$

$$\alpha_{x_{t+1}} = \alpha_{x_t}$$

*Similar for y and z.

where $\overline{v_{x_t}}$ and $\overline{a_{x_t}}$ are v_{xyz} and a_{xyz} converted into world frame of x axis direction.

Therefore, F_t is Jacobian matrix of equations above.

$$F_t = \begin{bmatrix} \frac{\partial x_{t+1}}{\partial x_t} & \cdots & \frac{\partial x_{t+1}}{\partial \alpha_t} \\ \vdots & \ddots & \vdots \\ \frac{\partial \alpha_{t+1}}{\partial x_t} & \cdots & \frac{\partial \alpha_{t+1}}{\partial \alpha_t} \end{bmatrix} \tag{13}$$

It also assumes that each state variable has independent noise, like

$$\begin{aligned}
x_{t+1} &= \cdots + w_0 \\
y_{t+1} &= \cdots + w_1 \\
&\vdots \\
\alpha_{z_{t+1}} &= \cdots + w_{14}
\end{aligned}$$

Therefore, we can get Q_t with

$$Q_t = \begin{bmatrix} \frac{\partial x_{t+1}}{\partial w_0} & \cdots & \frac{\partial x_{t+1}}{\partial w_{14}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \alpha_{t+1}}{\partial w_0} & \cdots & \frac{\partial \alpha_{t+1}}{\partial w_{14}} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} = I \tag{14}$$

C. Observation model

In ***robot_localization pkg***, it assumes that only independent additive noises exist as

$$\begin{aligned} z_x &= x_{t+1} + v_0 \\ z_y &= y_{t+1} + v_1 \\ &\vdots \\ z_{\alpha_z} &= \alpha_{z_{t+1}} + v_{14} \end{aligned}$$

where $z_x, z_y, \dots, z_{\alpha_z}$ are corresponding observation for the state, $x_{t+1}, y_{t+1}, \dots, \alpha_{z_{t+1}}$ are predictions from prediction step.

As a result, H_{t+1} is

$$H_{t+1} = \begin{bmatrix} \frac{\partial z_x}{\partial x_{t+1}} & \cdots & \frac{\partial z_x}{\partial \alpha_{z_{t+1}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_\alpha}{\partial x_{t+1}} & \cdots & \frac{\partial z_\alpha}{\partial \alpha_{z_{t+1}}} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} = I \qquad (15)$$

and R_{t+1} is

$$R_{t+1} = \begin{bmatrix} \frac{\partial z_x}{\partial v_0} & \cdots & \frac{\partial z_x}{\partial v_{14}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_\alpha}{\partial v_0} & \cdots & \frac{\partial z_\alpha}{\partial v_{14}} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} = I \qquad (16)$$

D. Handle asynchronous sensor data

Most of time, data from sensor is not synchronous, and it is a problem for EKF because we don't have another driving input between two observation measurements. However, in ***robot_localization pkg*** we keep track of all the states, we can make prediction step at any time because driving input is derived from states.

For example, if we have two IMU measurements then

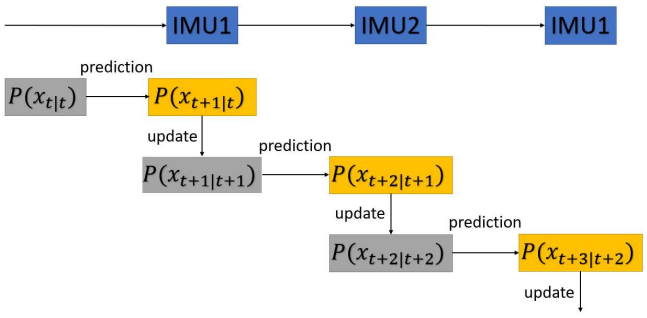

Fig. 1: flow chart with two IMU measurements

E. Update partial state variables

Another common problem is that not every sensor provides full states. For example, encoder can only provide linear velocity. So, we need to do partially update for observed states only.

To do this, we just need to decompose original state and covariance matrix into observed states only, then do update step to correct those state variables, and after update step put them back to original matrix.

IV. Results

Following experiment use ***drive_0015*** from **KITTI dataset** [3]. IMU data is already included in the dataset. To get Visual Odometry, I use library created by Nicolai Høirup Nielsen [4]. Python Code for the experiment can be found in my GitHub link [5].

A. Ground truth from GPS

I use provided GPS data as ground truth to compare the result.

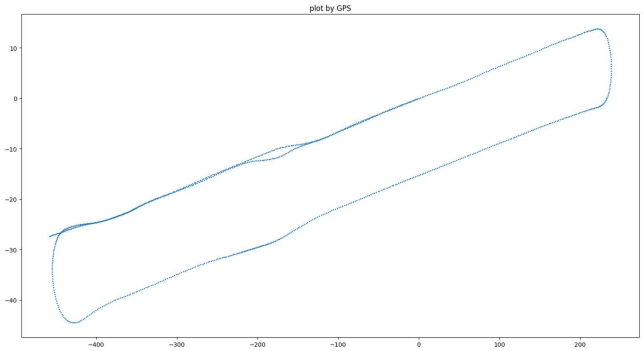

Fig. 2: plot by GPS data (without equal axis)

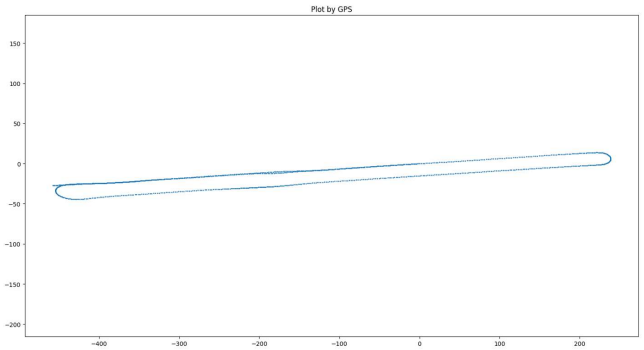

Fig. 3: plot by GPS data (with equal axis)

Note that the plot is from GPS data, therefore, it in on different coordinates system and scaling. However, we can still observe that the trajectory is a loop.

B. *Trajectory by IMU only*

This is trajectory estimated by IMU data only, and we can observe that it's barely a loop.

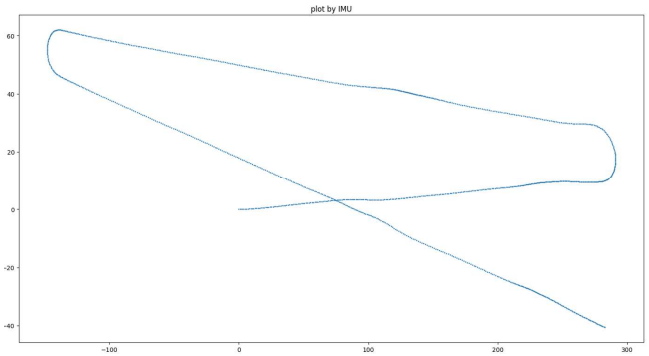

Fig. 3: plot by IMU data only (without equal axis)

C. *Trajectoy by Visual Odometry only*

We can see that the trajectory is totally different from IMU only.

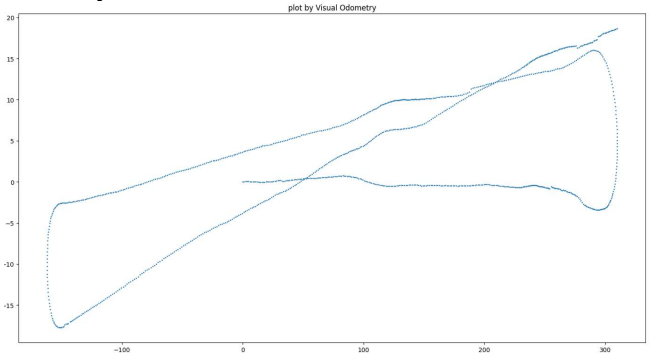

Fig. 4: plot by VIO only (without equal axis)

D. *Sensor fusion*

Now, use method discussed in *III* to do sensor fusion. Before doing that, we need to set up a few hyperparameters first.

For state $\boldsymbol{x}$

$$\boldsymbol{x} = \left[x, y, z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, a_x, a_y, a_z, \alpha_x, \alpha_y, \alpha_z\right]^T$$

my initial state covariance matrix is

```
def get_prediction_noise_covariance(self):
    return np.array([
        [0.05, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0.05, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0.06, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0.03, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0.03, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0.06, 0,    0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0.025, 0,    0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0.025, 0,    0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0.04, 0,    0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0,    0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0,    0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0.02, 0,    0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0,    0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01, 0],
        [0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0.01],
    ])
```

Fig. 5: initial state covariance matrix

Both IMU and VIO provide $\left[v_x, v_y, v_z, \omega_x, \omega_y, \omega_z\right]$, so define noise covariance matrix to

```
VO_covariance = np.array([
    [0.2,  0,    0,    0,    0,    0],
    [0,  0.3, 0,    0,    0,    0],
    [0,  0,    0.2, 0,    0,    0],
    [0,  0,    0,    0.01, 0,    0],
    [0,  0,    0,    0,    0.7, 0],
    [0,  0,    0,    0,    0,    0.011]
])
IMU_covariance = np.array([
    [0.01,  0,    0,    0,    0,    0],
    [0,  0.01, 0,    0,    0,    0],
    [0,  0,    0.06, 0,    0,    0],
    [0,  0,    0,    0.04, 0,    0],
    [0,  0,    0,    0,    0.05, 0],
    [0,  0,    0,    0,    0,    0.029]
])
```

Fig. 6: IMU and VIO noise covariance matrix
*covariance matrix vary for different data (setup).

With defined setup above, get the result as

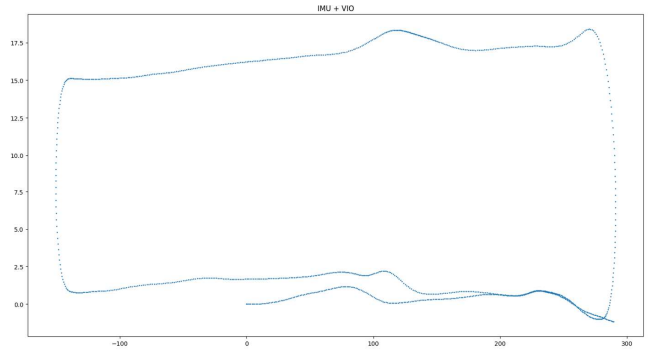

Fig. 7: plot by IMU + VIO (without equal axis)

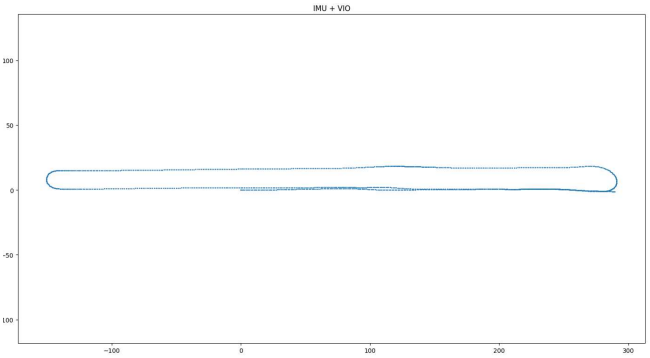

Fig. 8: plot by IMU + VIO (with equal axis)

Even though we can see small deviation in straight line while zooming in, the overall performance is pretty good, the trajectory loops now.

In summary, ***robot_localization pkg*** provides very powerful tool to do sensor fusion. It keeps sensor fusion simple if sensor provides one of the states in $\boldsymbol{x}$.

V. Reference

[1] Thomas Moore, Daniel Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System"

[2] Guy Campion, Georges Bastin, Brigitte D'Andrea-Novel, "Structural Properties and Classification of Kinematic and Dynamics Models of Wheeled Mobile Robots"

[3] Wiki, "Sum of normally distributed random variables", https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables

[4] Nicolai Høirup Nielsen, https://github.com/niconielsen32

[5] GitHub link https://github.com/hsyen23/ECE225_sensor_fusion.git