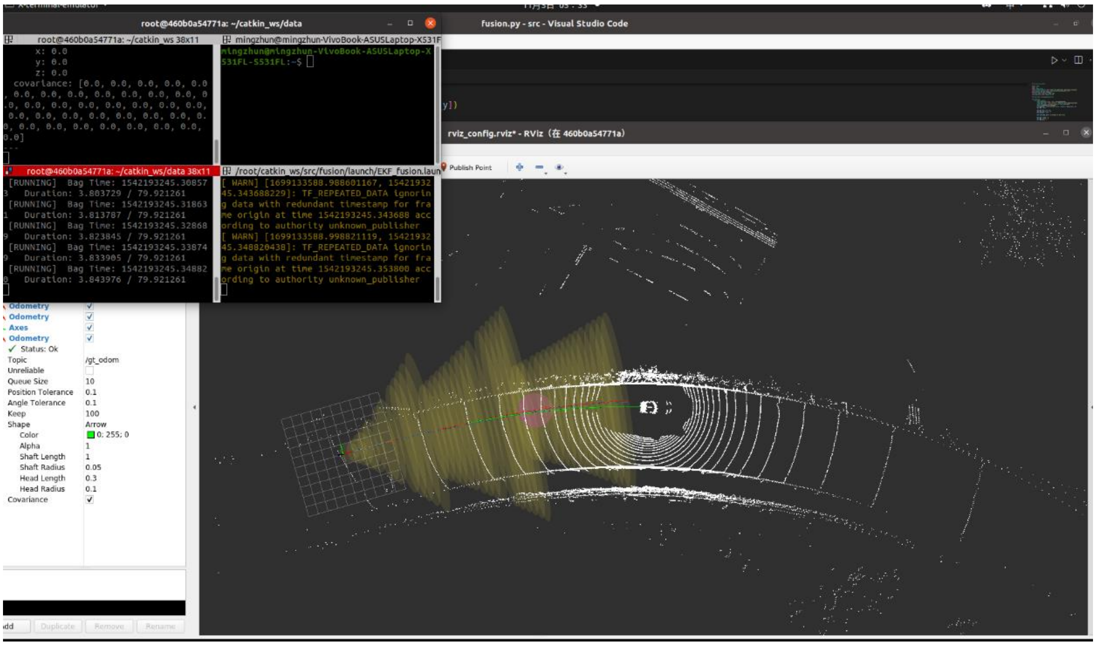
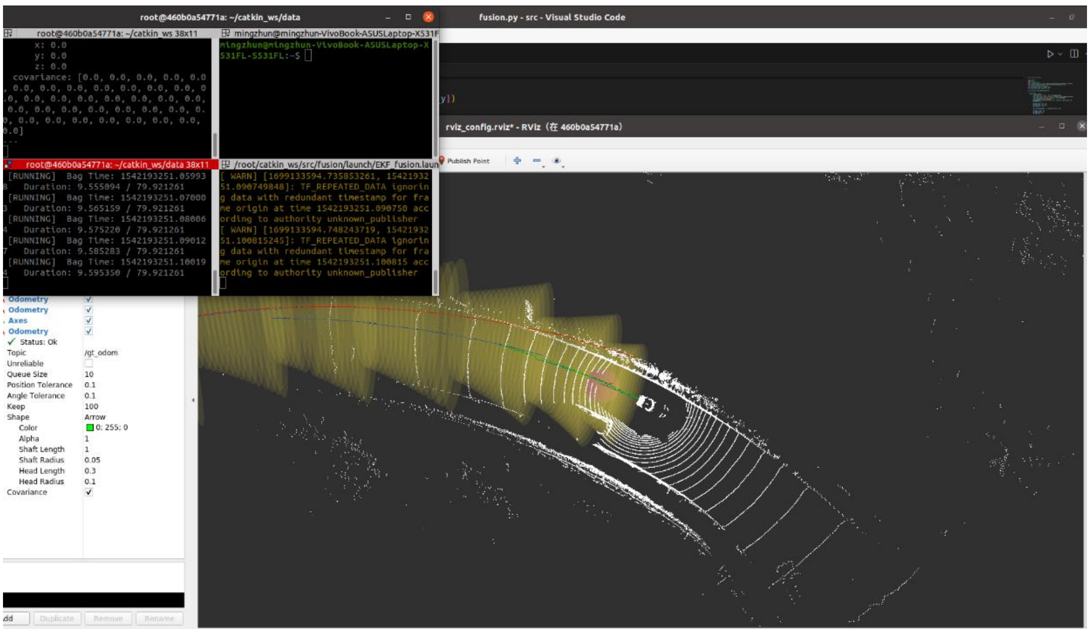
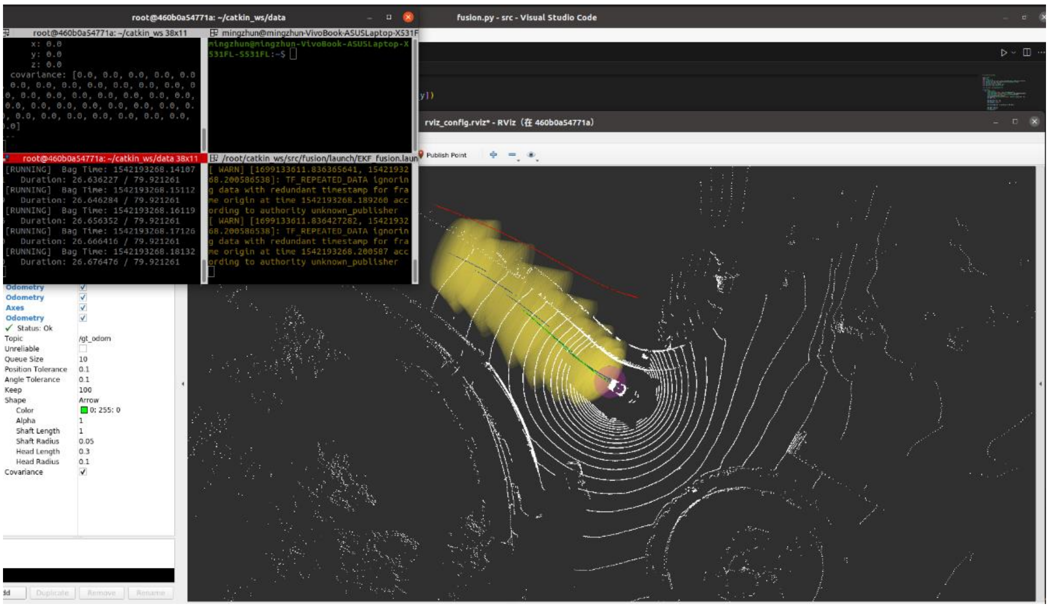
SDC Homework 4 - Application of EKF

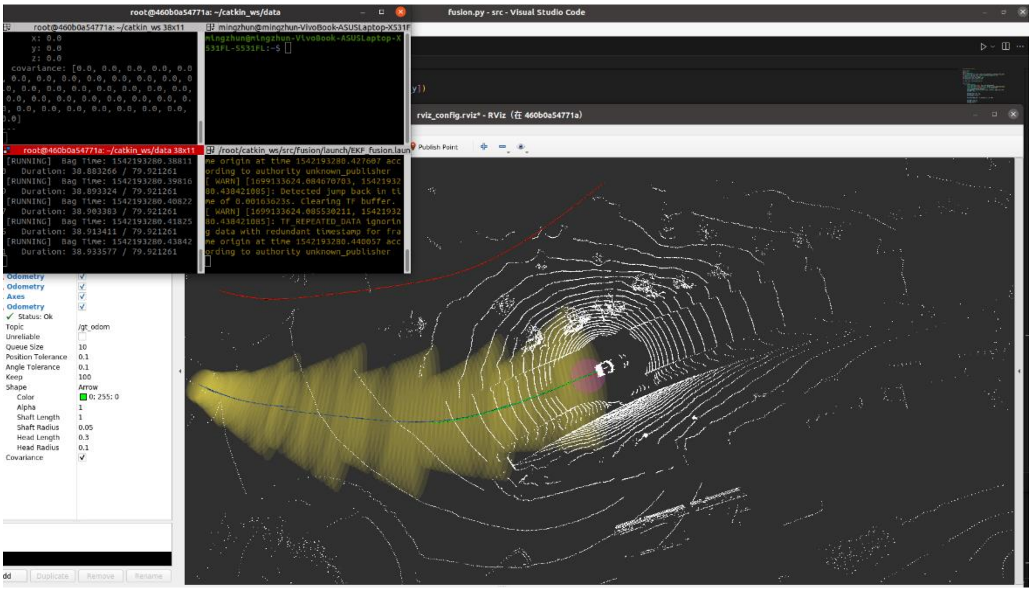
312512005　黃名諄

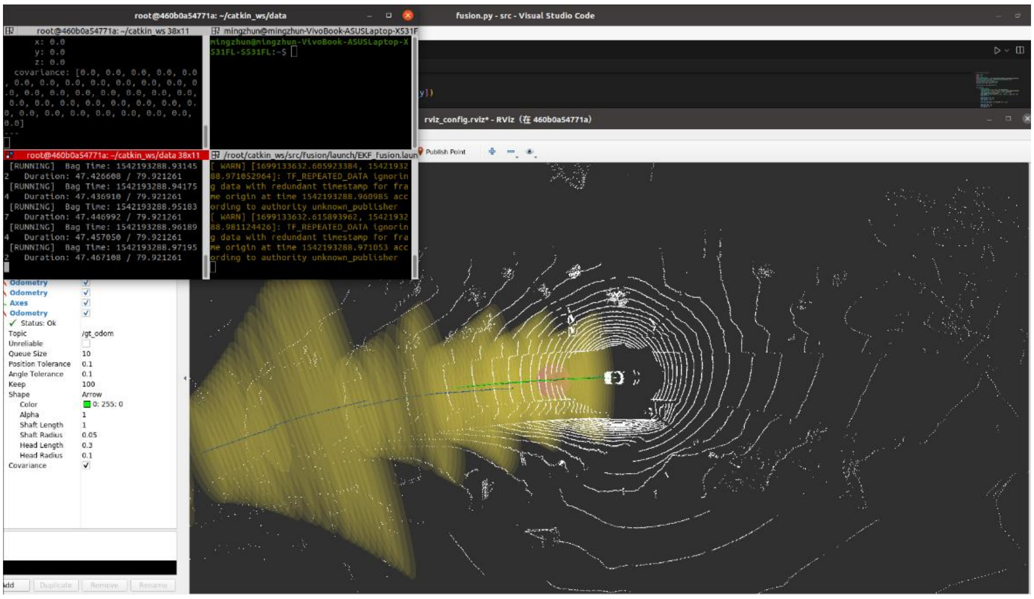
1. Screenshot of running program:

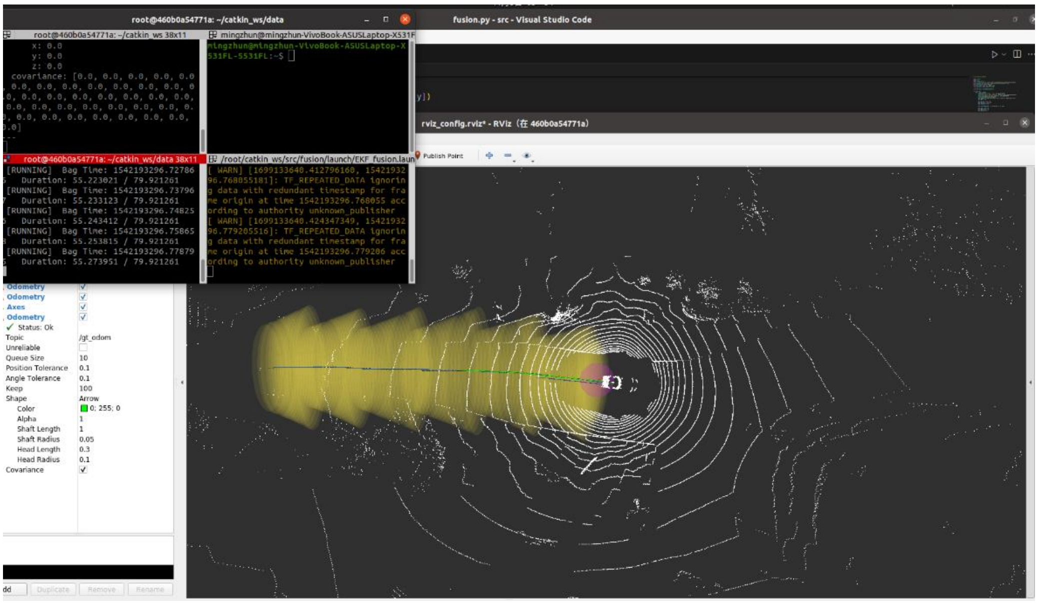


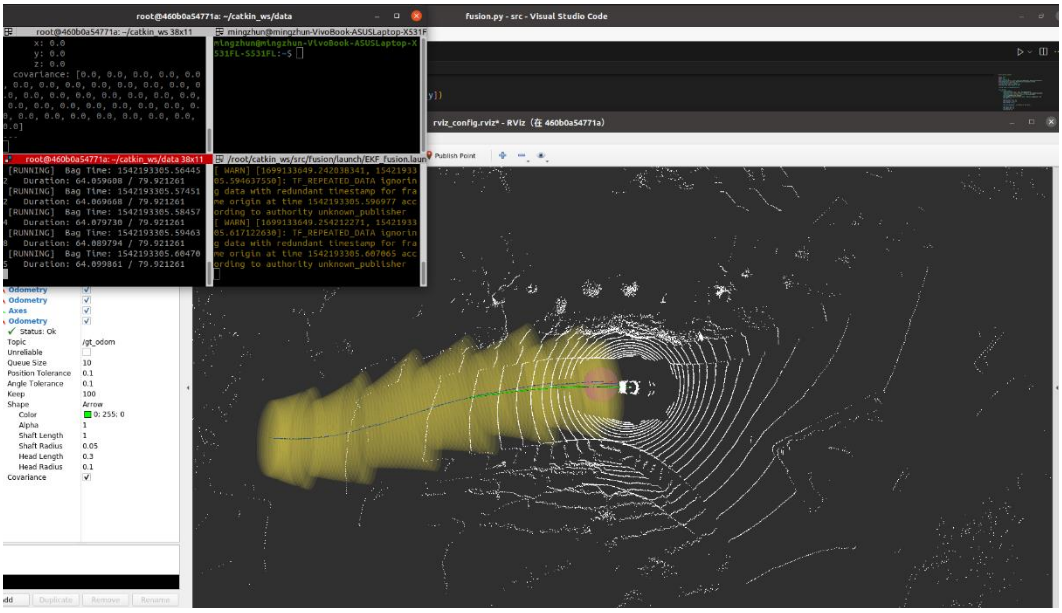


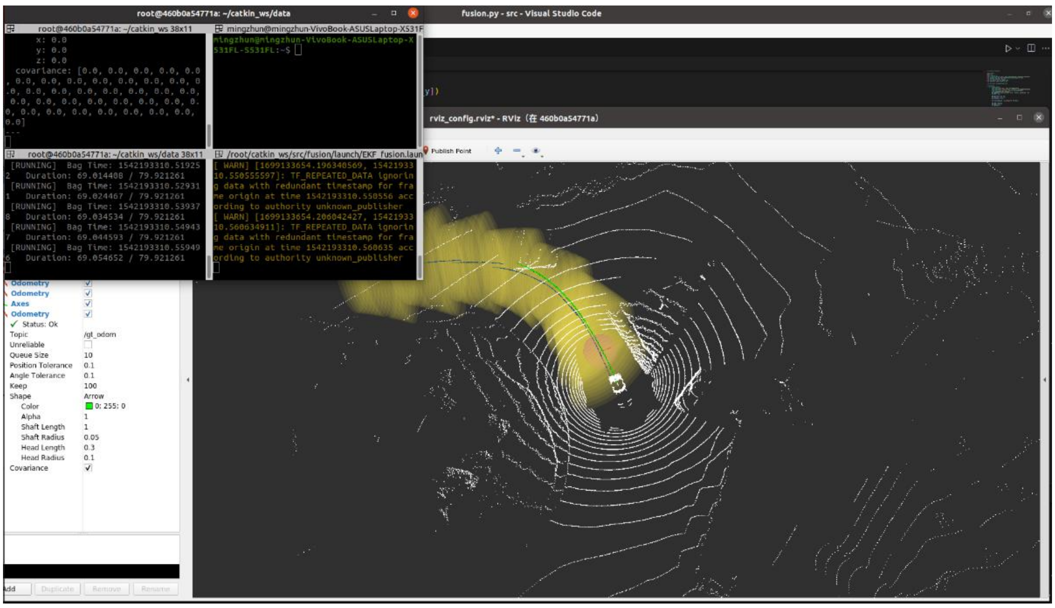

root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
rviz_config.rviz* - RViz（在 460b0a54771a）
Publish Point

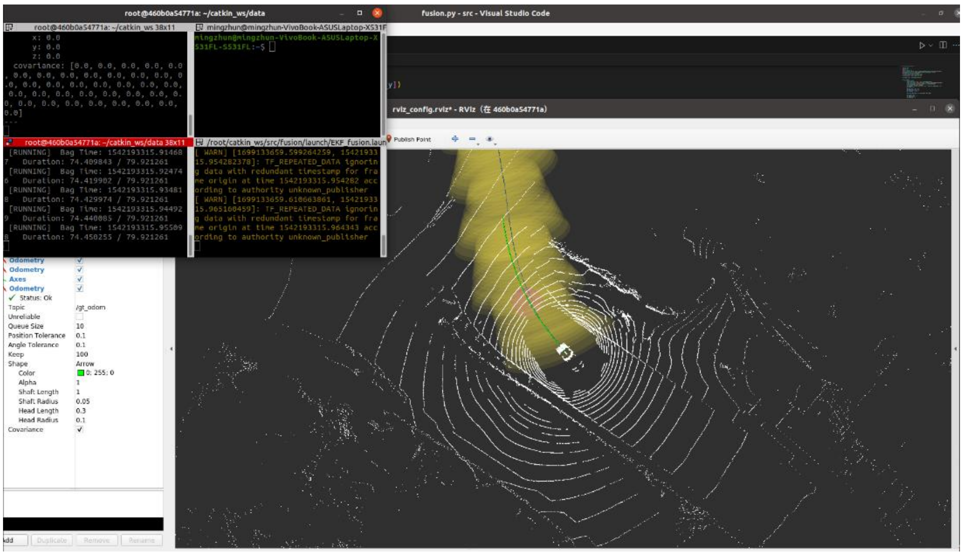

root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
rviz_config.rviz* - RViz（在 460b0a54771a）
Publish Point

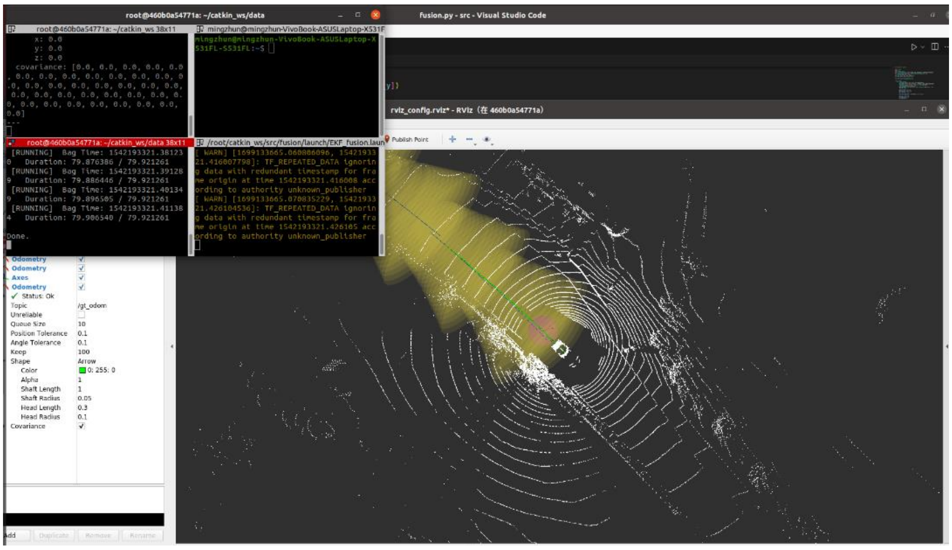

root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
rviz_config.rviz* - RViz（在 460b0a54771a）
Publish Point


root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
root@460b0a54771a: ~/catkin_ws 38x11
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531F
x: 0.0
y: 0.0
z: 0.0
covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~$
rviz_config.rviz* - RViz (在 460b0a54771a)
root@460b0a54771a: ~/catkin_ws/data 38x11
/root/catkin_ws/src/fusion/launch/EKF_fusion.laun
Publish Point
[RUNNING] Bag Time: 1542193296.727865 Duration: 55.223021 / 79.921261
[RUNNING] Bag Time: 1542193296.737967 Duration: 55.233123 / 79.921261
[RUNNING] Bag Time: 1542193296.748255 Duration: 55.243412 / 79.921261
[RUNNING] Bag Time: 1542193296.758658 Duration: 55.253815 / 79.921261
[RUNNING] Bag Time: 1542193296.778795 Duration: 55.273951 / 79.921261
[WARN] [1699133640.412796160, 1542193296.768055181]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193296.768055 according to authority unknown_publisher
[WARN] [1699133640.424347349, 1542193296.779205516]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193296.779206 according to authority unknown_publisher
Odometry
Odometry
Axes
Odometry
Status: Ok
Topic /gt_odom
Unreliable
Queue Size 10
Position Tolerance 0.1
Angle Tolerance 0.1
Keep 100
Shape Arrow
Color 0; 255; 0
Alpha 1
Shaft Length 1
Shaft Radius 0.05
Head Length 0.3
Head Radius 0.1
Covariance
Duplicate
Remove
Rename

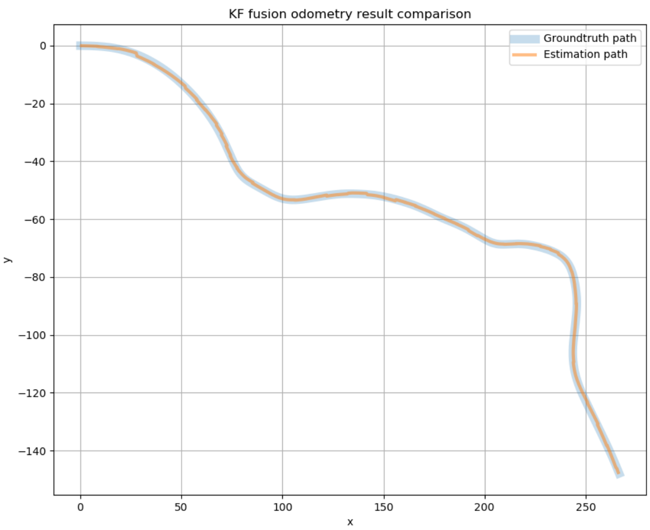

root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
root@460b0a54771a: ~/catkin_ws 38x11
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531F
x: 0.0
y: 0.0
z: 0.0
covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~$
rviz_config.rviz* - RViz (在 460b0a54771a)
root@460b0a54771a: ~/catkin_ws/data 38x11
/root/catkin_ws/src/fusion/launch/EKF_fusion.laun
Publish Point
[RUNNING] Bag Time: 1542193305.564452 Duration: 64.059608 / 79.921261
[RUNNING] Bag Time: 1542193305.574512 Duration: 64.069668 / 79.921261
[RUNNING] Bag Time: 1542193305.584574 Duration: 64.079730 / 79.921261
[RUNNING] Bag Time: 1542193305.594638 Duration: 64.089794 / 79.921261
[RUNNING] Bag Time: 1542193305.604705 Duration: 64.099861 / 79.921261
[WARN] [1699133649.242038341, 1542193305.594637550]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193305.596977 according to authority unknown_publisher
[WARN] [1699133649.254212271, 1542193305.617122630]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193305.607065 according to authority unknown_publisher
Odometry
Odometry
Axes
Odometry
Status: Ok
Topic /gt_odom
Unreliable
Queue Size 10
Position Tolerance 0.1
Angle Tolerance 0.1
Keep 100
Shape Arrow
Color 0; 255; 0
Alpha 1
Shaft Length 1
Shaft Radius 0.05
Head Length 0.3
Head Radius 0.1
Covariance
Duplicate
Remove
Rename


root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
root@460b0a54771a: ~/catkin_ws 38x11
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531F
x: 0.0
y: 0.0
z: 0.0
covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~$
rviz_config.rviz* - RViz (在 460b0a54771a)
root@460b0a54771a: ~/catkin_ws/data 38x11
/root/catkin_ws/src/fusion/launch/EKF_fusion.laun
Publish Point
[RUNNING] Bag Time: 1542193310.519252 Duration: 69.014408 / 79.921261
[RUNNING] Bag Time: 1542193310.529311 Duration: 69.024467 / 79.921261
[RUNNING] Bag Time: 1542193310.539378 Duration: 69.034534 / 79.921261
[RUNNING] Bag Time: 1542193310.549437 Duration: 69.044593 / 79.921261
[RUNNING] Bag Time: 1542193310.559496 Duration: 69.054652 / 79.921261
[WARN] [1699133654.196340569, 1542193310.550555597]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193310.550556 according to authority unknown_publisher
[WARN] [1699133654.206042427, 1542193310.560634911]: TF_REPEATED_DATA ignoring data with redundant timestamp for frame origin at time 1542193310.560635 according to authority unknown_publisher
Odometry
Odometry
Axes
Odometry
Status: Ok
Topic /gt_odom
Unreliable
Queue Size 10
Position Tolerance 0.1
Angle Tolerance 0.1
Keep 100
Shape Arrow
Color 0; 255; 0
Alpha 1
Shaft Length 1
Shaft Radius 0.05
Head Length 0.3
Head Radius 0.1
Covariance
Duplicate
Remove
Rename


root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
rviz_config.rviz* - RViz（在 460b0a54771a）


root@460b0a54771a: ~/catkin_ws/data
fusion.py - src - Visual Studio Code
rviz_config.rviz* - RViz（在 460b0a54771a）

2. Result.png:


KF fusion odometry result comparison
Groundtruth path
Estimation path
0
−20
−40
−60
−80
−100
−120
−140
y
0
50
100
150
200
250
x

3. Discussion:

1. How do you design the EKF (motion model, observation matrix, states, etc.)?

- States:

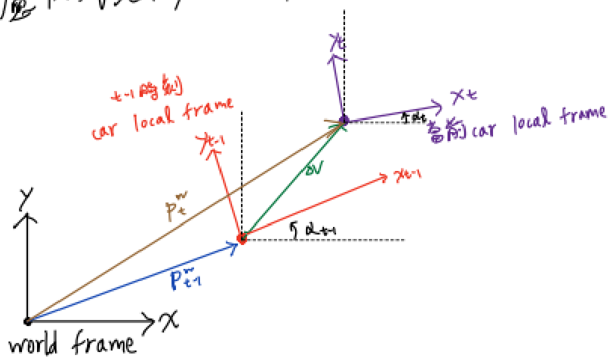
  只靠 radar odometry and GPS 難以取得速度資訊，因此 EKF 內我使用 only pose : x, y, yaw 當作 state 去預測更新

```python
# etc...
self.pose = np.array([x, y, yaw])
```

- Motion model:

  在使用 EKF 時需要找到 nonlinear 方程，使用 radar odometry data 處理後當 control input 至 EKF Motion model 內，我的設計概念是藉由上一時刻(t-1)在 world frame 下的 pose，預測當前時刻(t) 在 world frame 下的 pose，而 control input 則是相對於上一時刻(t-1)之 car local frame 下當前時刻(t)之 pose 位置，也就是改變量在上一時刻(t-1)之 car local frame 下之表示，具體概念細節如下:

考慮俯視下，二維平面



α_{t-1} : t-1 時刻, local frame yaw

α_t : t 時刻, local frame yaw

ΔV : 表 local frame t-1 原點相對 local frame t 原點之向量

P_t^w : local frame t 之原點在 world frame 下之表示

P_{t-1}^w : local frame t-1 之原點在 world frame 下之表示

* goal: find transformation of P_w^{t-1} to P_w^t

由向量加法可知，$P_t^w = P_{t-1}^w + \Delta V^w$

ΔV^w 表向量 ΔV 在 world frame 下之表示

其中，選擇 ΔV^{t-1} 可當做兩筆 odometry data 之變化量當作 control

丟入 EKF 之 prediction step 中

ΔV^{t-1} 表 ΔV 在 local frame t-1 下之表示

EKF中，P_w^{t-1} 是上一次儲存之 pose

，藉 ΔV^{t-1} control to predict P_t^w (pose at t)

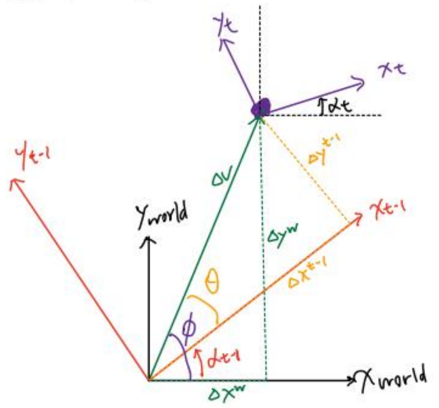
原先的 data 是 global frame 下，可取得兩筆 data 在 world frame 下的改變量，但我需要將此改變量轉成相對於上一時刻(t-1)之 car local frame 下之表示再送入 EKF 中做預測，具體細節如下推導:

* global change ⟶ local change

將原得到的 data 相減，表示的是相差之量在 world frame 下之表示

也就是 ΔV^w，但需轉為相對 t-1 時 local frame 下的改變量再傳入 EKF

以下為原理推導：



α_{t-1} is yaw of local frame t-1 (已知)

將當前 odometry data 和 t-1 時 odometry data 相減

$\Rightarrow$ 得 Δx^w, Δy^w

$\because \phi = atan2(\Delta y^w, \Delta x^w)$

$\therefore \theta = \phi - \alpha_{t-1}$

$\|\Delta V\| = \sqrt{(\Delta x^w)^2 + (\Delta y^w)^2}$

$$\Rightarrow \begin{array}{l} \Delta x^{t-1} = \|\Delta V\| \times \cos\theta \\ \Delta y^{t-1} = \|\Delta V\| \times \sin\theta \end{array}$$

而 $\Delta yaw = \alpha_t - \alpha_{t-1}$

因此，$u = \begin{bmatrix} \Delta x^{t-1} \\ \Delta y^{t-1} \\ \Delta Yaw \end{bmatrix}$，表時刻t下之pose在local frame t-1 下之表示
當作 control 送入EKF

```python
# Calculate current pose in the last car local frame as control input od EKF
delta_x = odom_x - self.last_odom_pose[0]
delta_y = odom_y - self.last_odom_pose[1]
angle_world = atan2(delta_y,delta_x)
angle_local = angle_world - self.last_odom_pose[2]
delta_distance = sqrt(delta_x**2+delta_y**2)

diff_x = delta_distance*cos(angle_local)
diff_y = delta_distance*sin(angle_local)
diff_yaw = odom_yaw - self.last_odom_pose[2]

self.last_odom_pose[0]= odom_x
self.last_odom_pose[1]= odom_y
self.last_odom_pose[2]= odom_yaw

#rospy.loginfo(odom_yaw)
control = np.array([diff_x, diff_y, diff_yaw])#???
```

接著要找 EKF 內 Motion model 的非線性轉換關係並找到其 Jacobian，其實也只是配合座標轉換的向量加法關係，詳細推導如

* EKF 's motion model

EKF之 state $\mathbf{X}$ 設為 pose, np.array([x, y, yaw])

如一開始之目標，將 $P_t^w = P_{t-1}^w + \Delta V^w$

$$\mathbf{X}_t = \begin{bmatrix} x_t \\ y_t \\ Yaw_t \end{bmatrix}, \quad \mathbf{X}_{t-1} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ Yaw_{t-1} \end{bmatrix}, \quad P_t^w = \begin{bmatrix} x_t \\ y_t \end{bmatrix}, \quad P_{t-1}^w = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix}$$

control: $u = \begin{bmatrix} \Delta x^{t-1} \\ \Delta y^{t-1} \\ \Delta Yaw \end{bmatrix}$　其中 $\Delta V^{t-1} = \begin{bmatrix} \Delta x^{t-1} \\ \Delta y^{t-1} \end{bmatrix}$，表t時刻下 x_t 和 y_t 在 local frame t-1 下之表示

已知 car local frame 和 world frame 是旋轉+平移關係

⇒ 兩座標間轉換可用 transform matrix T_{t-1}^w 表示

$$P_t^w = T_{t-1}^w \Delta V^{t-1}, \quad T_{t-1}^w = \begin{bmatrix} R_{t-1}^w & P_{t-1}^w \\ 0^T & 1 \end{bmatrix}$$

$\alpha = Yaw_{t-1}$

$$\Rightarrow \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & x_{t-1} \\ \sin\alpha & \cos\alpha & y_{t-1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x^{t-1} \\ \Delta y^{t-1} \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{aligned} x_t &= x_{t-1} + \cos\alpha \Delta x^{t-1} - \sin\alpha \Delta y^{t-1} \rightarrow f_1 \\ y_t &= y_{t-1} + \sin\alpha \Delta x^{t-1} + \cos\alpha \Delta y^{t-1} \rightarrow f_2 \end{aligned}$$

而 Yaw 部分：$Yaw_t = Yaw_{t-1} + \Delta Yaw \rightarrow f_3$

下：

則 motion model 可整理成：

$$\begin{bmatrix} x_t \\ y_t \\ Yaw_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ Yaw_{t-1} \end{bmatrix} + \begin{bmatrix} \cos\alpha & -\sin\alpha & \\ \sin\alpha & \cos\alpha & \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \Delta x^{t-1} \\ \Delta y^{t-1} \\ \Delta Yaw \end{bmatrix}}_{u}$$

此即我的 motion model

接著需要 $\underbrace{\text{Jacobian matrix}}_{\text{存在 self.A}}$ to predict $\underbrace{\text{state covariance}}_{\text{self.S}}$ matrix

$$\text{Jacobian matrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_{t-1}} & \frac{\partial f_1}{\partial y_{t-1}} & \frac{\partial f_1}{\partial \alpha} \\ \frac{\partial f_2}{\partial x_{t-1}} & \frac{\partial f_2}{\partial y_{t-1}} & \frac{\partial f_2}{\partial \alpha} \\ \frac{\partial f_3}{\partial x_{t-1}} & \frac{\partial f_3}{\partial y_{t-1}} & \frac{\partial f_3}{\partial \alpha} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & (-\sin\alpha)\Delta x^{t-1} - (\cos\alpha)\Delta y^{t-1} \\ 0 & 1 & (\cos\alpha)\Delta x^{t-1} - (\sin\alpha)\Delta y^{t-1} \\ 0 & 0 & 1 \end{bmatrix}$$

$\Rightarrow$ state covariance prediction:

$$self.S = (self.A)(self.S)(self.A)^T + self.R$$

將上述 motion model 在 EKF code 中實現如下:

```
#use transition matrix of world and last car local frame to get the nonlinear motion model
self.u_transition_matrix = np.array([ [cos(self.pose[2]), -sin(self.pose[2]), 0],
                                      [sin(self.pose[2]), cos(self.pose[2]), 0],
                                      [0,0,1] ])
#A is Jacobian matrix here
self.A = np.array([ [1, 0, (-u[0]*sin(self.pose[2])-u[1]*cos(self.pose[2]))],
                    [0, 1, (u[0]*cos(self.pose[2])-u[1]*sin(self.pose[2]))],
                    [0, 0, 1] ])

self.pose = self.pose + np.dot(self.u_transition_matrix, u)
self.S = np.dot(np.dot(self.A, self.S) , self.A.T) + self.R
```

- observation matrix

  對於 observation model，我還是使用 hw3 中的線性關係

$$\begin{bmatrix} z_x \\ z_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ yaw \end{bmatrix} + \delta_t$$

```python
def update(self, z):
    # Base on the Kalman Filter design in Assignment 3
    # Implement a linear or nonlinear observation matrix for the measurement input
    # Calculate Jacobian matrix of the matrix as self.C

    #use linear observation model as hw3
    K = np.dot(np.dot(self.S, self.C.T), np.linalg.inv((np.dot(np.dot(self.C, self.S), self.C.T) + self.Q)))
    self.pose = self.pose + np.dot(K, (z - np.dot(self.C, self.pose)) )
    self.S = np.dot((np.identity(3)-np.dot(K, self.C)), self.S)
    return self.pose, self.S
```

```python
# Observation matrix
self.C = np.array([[1,0,0],[0,1,0]])
#self.C = np.identity(3)
```

使用的觀測 data 是 gps，但其只有 x,y 沒有 yaw 的資訊，在這個 update 步驟中我直接使用 gps 的 x,y 觀測而沒有使用到估算的 yaw，這是因為其實我有嘗試過使用兩筆 data 來計算 atan2(dy1,dx1)

```python
# use 2 gps data to find a approximate yaw
dx1 = gps_x- self.gps_x_last
dy1 = gps_y - self.gps_y_last
gps_distance = sqrt(dx1**2+dy1**2)

# if car stop whin a range, then don't change the approximate yaw
if gps_distance > 1:
    self.gps_yaw = atan2(dy1,dx1)
measurement = np.array([gps_x, gps_y, self.gps_yaw])
self.gps_x_last = gps_x
self.gps_y_last = gps_y
'''
```

當作粗略估計的測量 yaw 值丟入 EKF 做 update，但並沒有更好的效果，原因我認為是其不是真的觀測數據，而是近似的 yaw，本身可靠度就不高，並不能給予更好的修正，結果比較如下方:

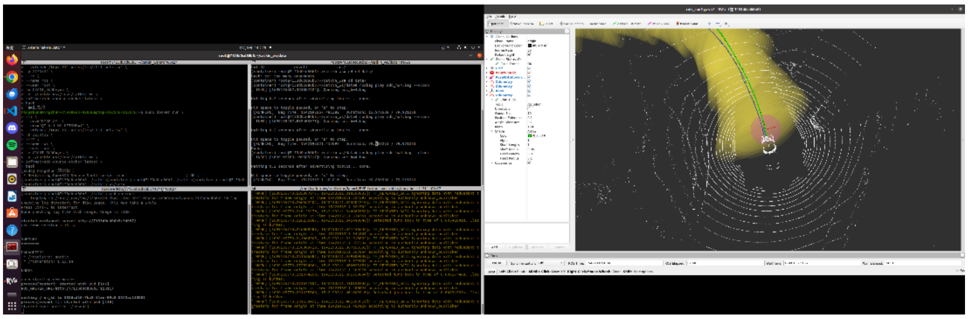


figure 1 demo with use gps approximate yaw

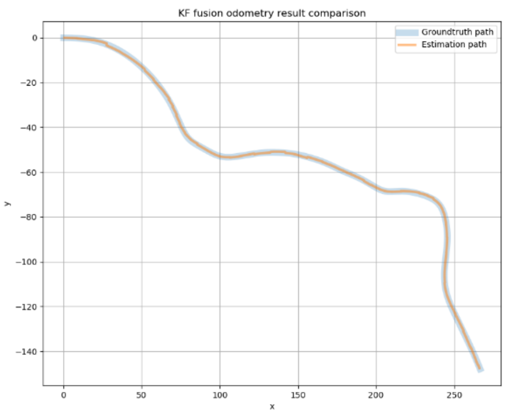


figure 2 result with use gps approximate yaw

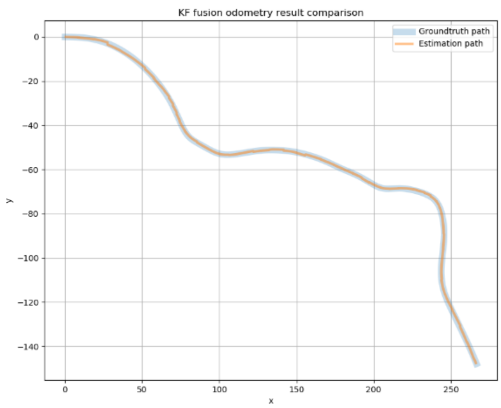


figure 3 result use gps directly

因此我最後才使用如 hw3 之設計來直接使用 gps 的 x,y 觀測 data。

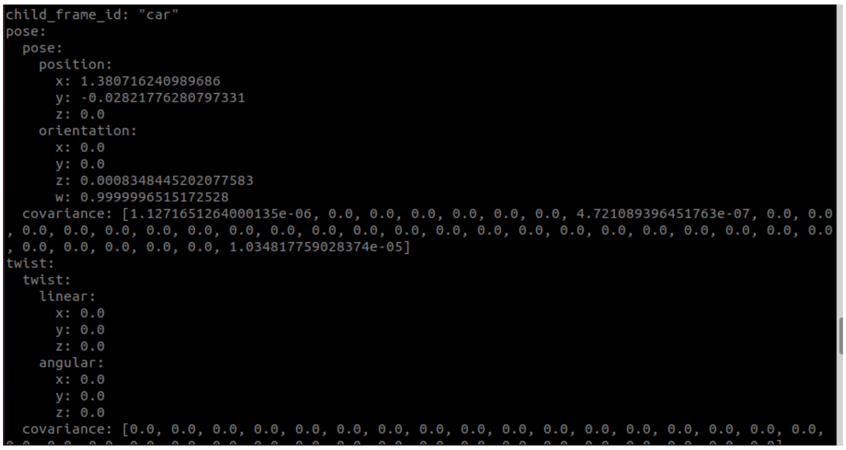
2. What is the covariance matrix of GPS, radar odometry and what does it mean?

    covariance matrix 表示的是 sensor data 的 uncertainty，越大表示不確定性越大，而 EKF 結果會偏向於不確定性小的 data。兩 sensor 之 data 中都有包含其 covariance 資訊，但我在實作中有發現一些問題，對其做了一些調整來使結果更好，會在以下詳細討論

- modify the covariance matrix:

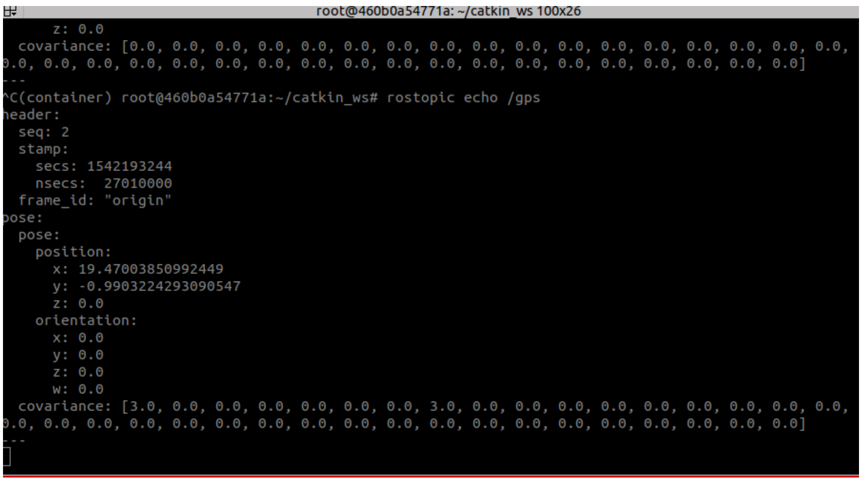
    我有將兩 sensor 各自的 data 調出來看，可發現 radar odometry covariance 大概在 10^{-5}~10^{-7}，非常的小，但 gps covariance 在 3，兩 sensor covariance 比例相差很大，但實際看起來 gps 應該要準一些，所以修改兩者 covariance，改變兩者 uncertainty 比例可得不同效果

    radar odometry data:



```
child_frame_id: "car"
pose:
  pose:
    position:
      x: 1.380716240989686
      y: -0.02821776280797331
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0008348445202077583
      w: 0.9999996515172528
  covariance: [1.1271651264000135e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.721089396451763e-07, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 1.034817759028374e-05]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

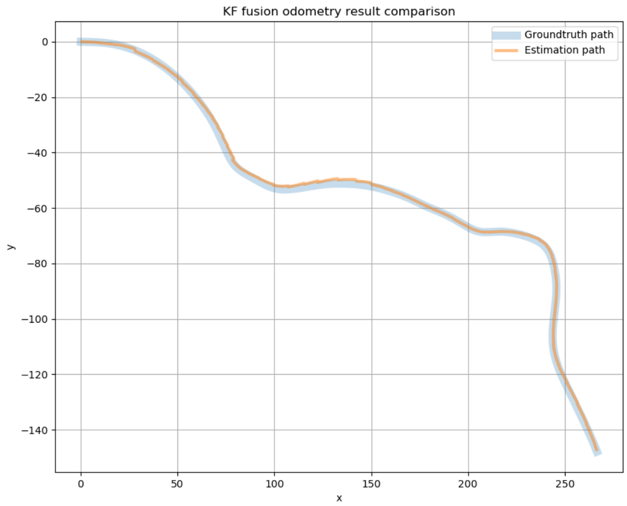
gps data:



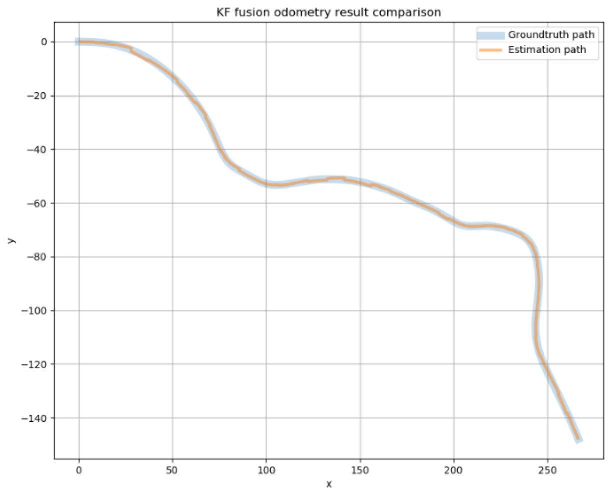
```
root@460b0a54771a: ~/catkin_ws 100x26
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
^C(container) root@460b0a54771a:~/catkin_ws# rostopic echo /gps
header:
  seq: 2
  stamp:
    secs: 1542193244
    nsecs:  27010000
  frame_id: "origin"
pose:
  pose:
    position:
      x: 19.47003850992449
      y: -0.9903224293090547
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
  covariance: [3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

因此我對兩者分別乘上不同的倍率比較結果，使用原始 data 的 covariance 結果不是最好的，最後試出來 radar odometry covariance 乘 1000 和 gps covariance 乘 10 會有最好的定位效果，幾個組合結果比較如下:

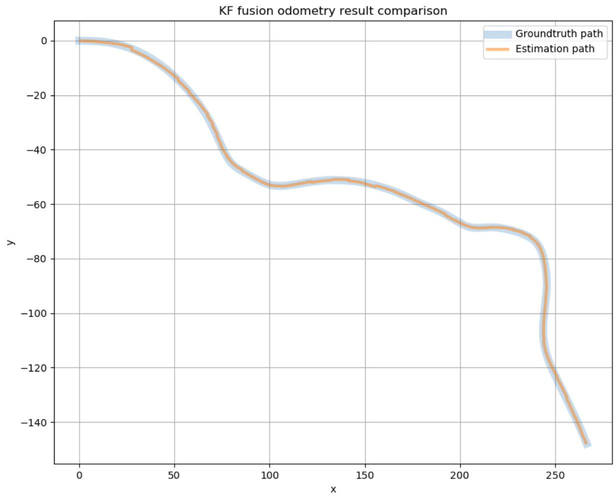
I. use original covariance:



II. use radar odometry covariance*1000 with original gps covariance



III. use radar odometry covariance*1000 with gps covariance*10



因此最後使用的 covariance matrix 如下:

- radar odometry covariance matrix (use in motion model)

```
else:
    # Update error covriance
    self.EKF.R =  np.array([[odom_covariance[0,0],odom_covariance[0,1],odom_covariance[0,5]],
                           [odom_covariance[1,0],odom_covariance[1,1],odom_covariance[1,5]],
                           [odom_covariance[5,0],odom_covariance[5,0],odom_covariance[5,5]]
                           ])*1000 #???
```

- gps covariance matrix (use in observation model)

```
else:
    # Update error covriance
    self.EKF.Q = np.array([[gps_covariance[0][0], gps_covariance[0][1]],
                           [gps_covariance[1][0], gps_covariance[1][1]]
                           ]
                           )*10  #???
    self.EKF.update(z = measurement)
```