SDC Homework 5 – Calibration

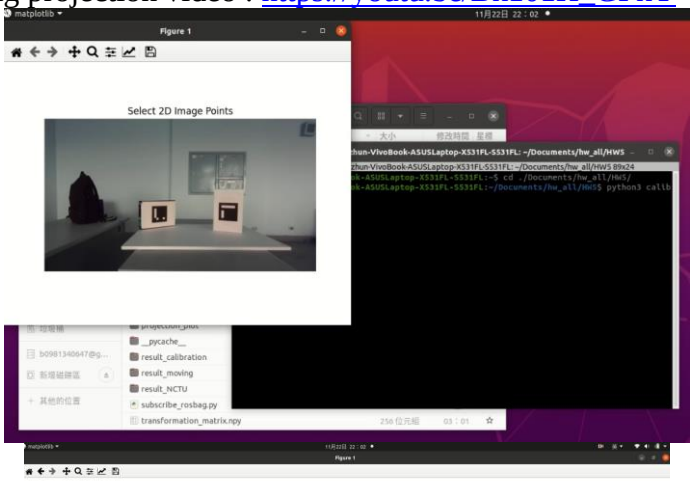
312512005 黃名諄

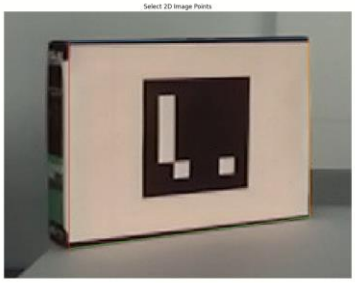
1. Screenshot of running code

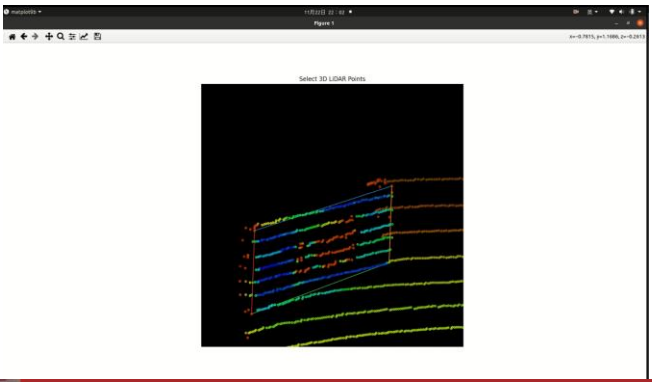
   Full code running video : https://youtu.be/32GIBamMmDA

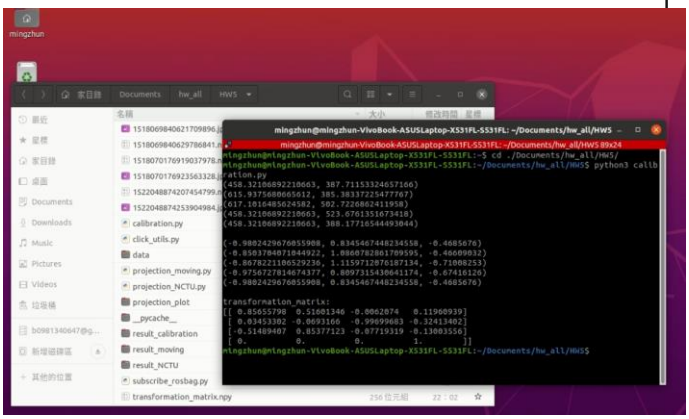
   sdc-moving.bag projection video : https://youtu.be/PhLDoMoWqJc

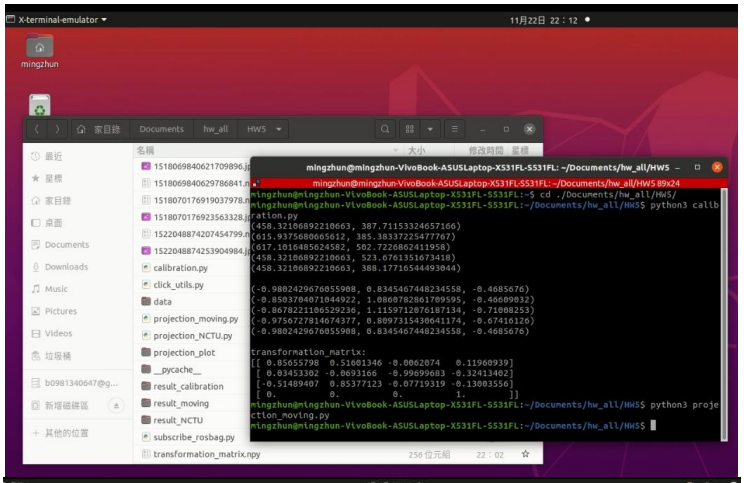
   sdc-NCTU.bag projection video : https://youtu.be/Bk101H_GPkY

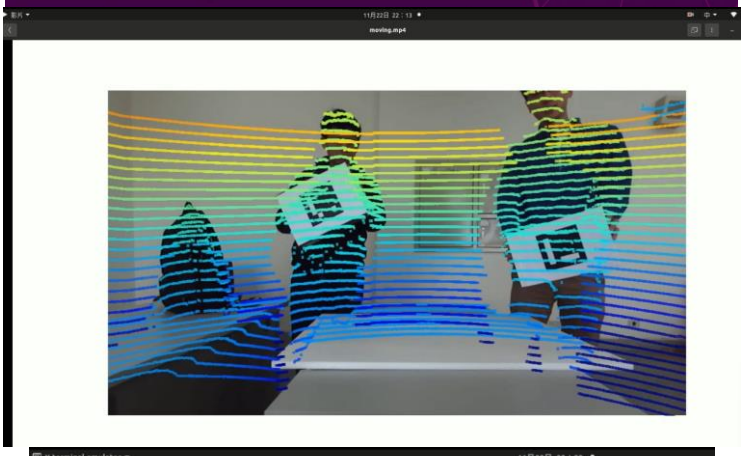


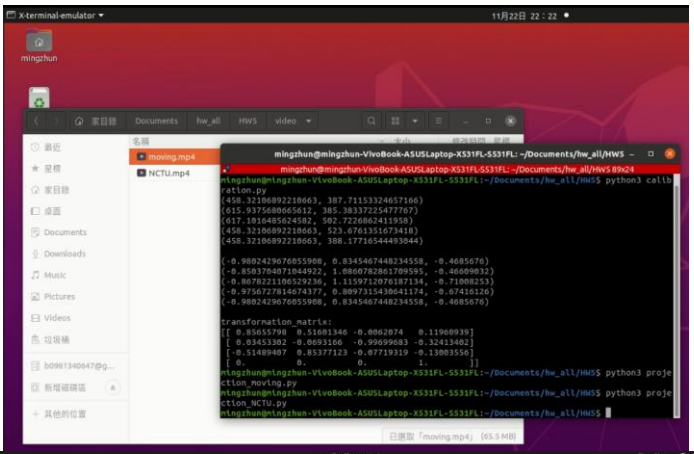


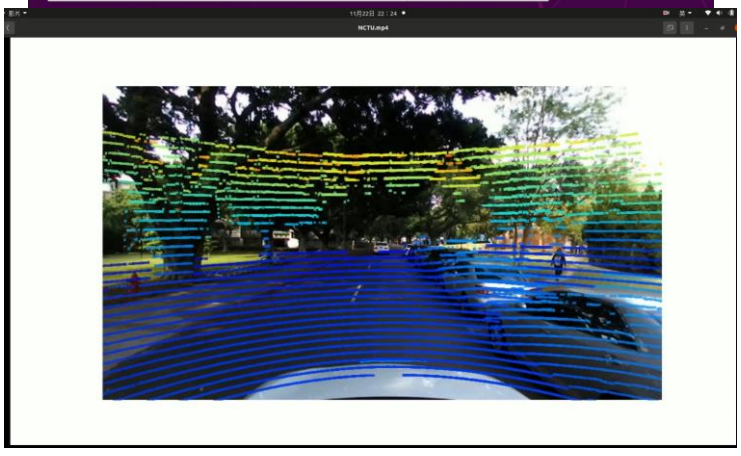





X-terminal-emulator
11月22日 22：12
mingzhun
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL: ~/Documents/hw_all/HW5
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL: ~/Documents/hw_all/HW5 89x24
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~$ cd ./Documents/hw_all/HW5/
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~/Documents/hw_all/HW5$ python3 callb
ration.py
(458.32106892210663, 387.71153324657166)
(615.9375680065612, 385.38337225477767)
(617.1016485624582, 502.7226862411958)
(458.32106892210663, 523.6761351673418)
(458.32106892210663, 388.17716544493044)
(-0.9802429676055908, 0.8345467448234558, -0.4685676)
(-0.8503784071044922, 1.0860782861709595, -0.46609932)
(-0.8678221106529236, 1.1159712076187134, -0.71008253)
(-0.9756727814674377, 0.8097315430641174, -0.67416126)
(-0.9802429676055908, 0.8345467448234558, -0.4685676)
transformation_matrix:
[[0.85655798 0.51601346 -0.0062074   0.11960939]
 [0.03453302 -0.0693166 -0.99699683 -0.32413482]
 [-0.51489407 0.85377123 -0.07719319 -0.13003556]
 [0.   0.   0.   1.   ]]
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~/Documents/hw_all/HW5$ python3 proje
ction_moving.py
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL:~/Documents/hw_all/HW5$
transformation_matrix.npy
moving.mp4

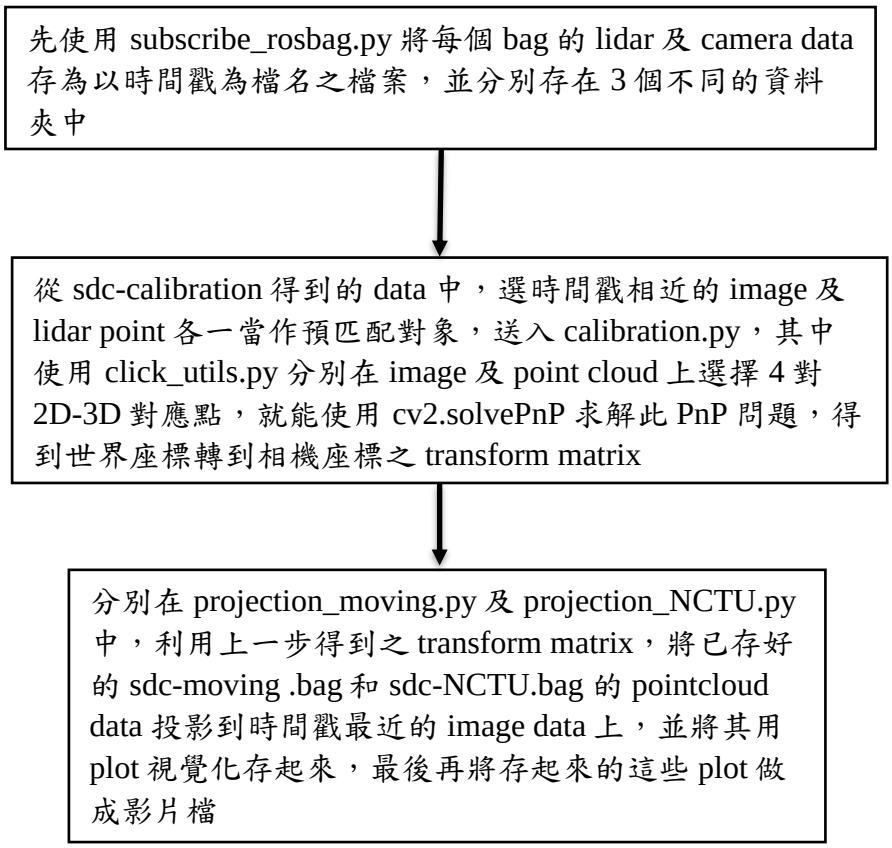



X-terminal-emulator
11月22日 22：22
mingzhun
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL: ~/Documents/hw_all/HW5
mingzhun@mingzhun-VivoBook-ASUSLaptop-X531FL-S531FL: ~/Documents/hw_all/HW5 89x24
moving.mp4
NCTU.mp4



2. Code explanation:

整體程式架構流程如下:



具體程式細節及問題如下說明:

I. calibration.py:

```
#pick the image point
image = cv2.imread('1518069840621709896.jpg')
picked_2D_points = click.click_points_2D(image)

#pick the 3D Lidar point
points = np.load('1518069840629786841.npy')
picked_3D_point = click.click_points_3D(points)

#intrinsic matrix of camera
height, width, channels = image.shape
fx=fy=698.939
intrinsic_matrix = np.array([[fx, 0, width/2],
                             [0, fy, height/2],
                             [0, 0, 1]])
dist_coeffs = np.zeros(5)
```

我先人工挑選時間戳相近的 lidar point file 和 image file，使用此兩筆 data 作為拿來選擇對應 2D-3D 點的對象，使用 click_utils.py 內的 function2u03 點出 4 對對應點。並依照助教給的參數資訊，及 intrinsic matrix 公式建立 intrinsic matrix 以及 camera's distortion coefficients

intrinsic matrix:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 698.939 & 0 & image\ width/2 \\ 0 & 698.939 & image\ height/2 \\ 0 & 0 & 1 \end{bmatrix}$$

```
#solve pnp problem to get transform matrix
retval,rvec,tvec= cv2.solvePnP(picked_3D_point, picked_2D_points, intrinsic_matrix, dist_coeffs,flags=cv2.SOLVEPNP_EPNP)
rvec, tvec = cv2.solvePnPRefineLM(picked_3D_point, picked_2D_points, intrinsic_matrix,dist_coeffs,rvec,tvec)

#use the result of solve pnp problem to build the transform matrix
rotation_matrix, _ = cv2.Rodrigues(rvec)
transformation_matrix = np.concatenate((rotation_matrix, tvec),axis=1)
transformation_matrix = np.concatenate((transformation_matrix, np.array([[0, 0, 0, 1]])), axis=0)
print('transformation_matrix:')
print(transformation_matrix)
np.save('./transformation_matrix.npy', transformation_matrix)
```

接著將選好的 2D 點、3D、intrinsic matrix 及 distortion coefficients 帶入 cv2.solvePnP 求解 PnP 問題，因有 4 個點對，經測試後選擇 SOLVEPNP_EPNP 來求解會較正確及穩定，解出之結果還會經過 cv2.solvePnPRefineLM 來使用 non-linear Levenberg-Marquardt minimization 做 Pose refinement，最後有了 R 及 t，可建構出能轉換 lidar coordinate 至 camera coordinate 的 transformation matrix:

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

並將其存起來給後續投影做使用。

II. projection_moving.py 及 projection_NCTU.py

兩者邏輯相同，只是讀 data 的路徑不同，因此此處以 projection_NCTU.py 來做解釋

```
def main():
    lidar_folder = './result_NCTU/lidar/'
    image_folder = './result_NCTU/camera/'
    plot_output_folder = './projection_plot/NCTU'
    video_folder = './video/'
    if not os.path.exists(plot_output_folder):
        os.makedirs(plot_output_folder)
    if not os.path.exists(video_folder):
        os.makedirs(video_folder)
```

首先先前已將 bag 中的 data 存下來了，此處就是定義好從哪個路徑讀這些存下來的 data，以及後續投影後的 plot 和轉為影片檔後要存的資料夾路徑

```
#sort the image and lidar results by timestamp
lidar_files = sorted(os.listdir(lidar_folder), key=lambda x: int(x.split('.')[0]))
image_files = sorted(os.listdir(image_folder), key=lambda x: int(x.split('.')[0]))

fig = plt.figure(figsize=(12.8, 7.2), dpi=100)
ax = fig.add_subplot()
```

因 subscribe 下來的檔名為 timestamp，因此先將這些 data 用其檔名的 timestamp 來排序，並建立畫布供後續投影視覺化使用

```python
for i,lidar in enumerate(lidar_files):
    # get the timestamp of the lidar file
    lidar_timestamp = int(lidar[:-4])  # remove.npy

    # find the image whose timestamp is closest to the lidar file
    closest_image_file = min(image_files, key=lambda x: abs(int(x[:-4]) - lidar_timestamp))

    # load the lidar and image data
    lidar_data = np.load(os.path.join(lidar_folder, lidar))
    image_data = cv2.imread(os.path.join(image_folder, closest_image_file))
    # because cv2.load is BGR, we need to convert the image data into RGB
    image_data = cv2.cvtColor(image_data, cv2.COLOR_BGR2RGB)

    projection(image_data,lidar_data,ax,i,plot_output_folder)
```

接著觀察之前存好的 lidar data 和 camera data，會發現兩者 timestamp 對不上且 image data 數也較 lidar points data 多，代表 camera 頻率比 lidar 高，所以我的處理方式為，為每一筆 lidar points data file 找到 timestamp 最相近的 image file，將兩者 load 進來，此處還需對 image 做 BGR to RGB 的轉換，原因於問題 3 解釋，將此 image file 當作投影對象把 lidar points 投影上去，因此對每組 lidar points data 做投影，結果會有跟 lidar file 數一樣多的 plot，投影的 function 如下說明

```python
def projection(img,pointcloud,ax,index,plot_output_folder):

    transformation_matrix = np.load('transformation_matrix.npy')
    height, width, channels = img.shape
    fx=fy=698.939
    intrinsic_matrix = np.array([[fx, 0, width/2],
                                 [0, fy, height/2],
                                 [0, 0, 1]])
```

將 calibration.py 得到的 transformation matrix 讀進來使用，並一樣建立好 intrinsic matrix

```python
points_world = pointcloud[:,0:3]
uv = []
#using the transformation matrix to project the points onto the image
for i in range(len(points_world)):
    #world coordinates to camera coordinates
    points_camera = transformation_matrix @ (np.append(points_world[i], 1))
    points_camera = points_camera[:-1]

    #camera coordinates to image planes
    image_point = intrinsic_matrix @ (points_camera*(1/points_camera[-1]))
    uv.append(image_point[0:2])
uv = np.array(uv)
```

取得 point 的 x_w , y_w , z_w 座標值，透過 transformation matrix 轉至 camera coordinate 下 x_c , y_c , z_c ，數學表示如下:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

接著從 camera coordinate 透過 intrinsic matrix 轉至成像平面上

$$\begin{bmatrix} \mathrm{u} \\ \mathrm{v} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{z_c} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

得到成像平面上座標值 u,v，對每個 lidar point 都做上述的投影存在 list uv 內

```
ax.cla()   #clear the plot
ax.imshow(img)
ax.set_xlim(0, 1280)
ax.set_ylim(720, 0)
ax.scatter([uv[:, 0]], [uv[:, 1]], c=pointcloud[:, 2], marker=',', s=10, edgecolors='none', alpha=0.7, cmap='jet')
ax.set_axis_off()

#save the plot
output_file = os.path.join(plot_output_folder, '{:d}.png'.format(index))
plt.savefig(output_file)
```

接著如助教給的方式，將圖片及投影點 plot 在畫布上，並將結果圖存起來。

而跑迴圈每組 lidar points data 都會建立出一張投影後的圖，原本是使用 plt.pause()來使投影圖能在畫布上不斷更新顯示，但是試了很多次都達不到助教的連續效果，因此決定將所有圖存好後製成影片檔控制其 fps 使其能和助教的影片長度相同

main()中:

```
#transform the projection plots into video
fps = len(lidar_files)/137
video(plot_output_folder,video_folder,fps)
```

fps 是輸出的投影圖數/助教影片中的時間長度

video 製作 funtion 如下:

```
def video(plots_folder, video_folder,fps):
    video_path = video_folder + 'NCTU.mp4'
    output_video = video_path

    #get the plots list after sorting
    plots = sorted(os.listdir(plots_folder),key=lambda x: int(x.split('.')[0]))

    #get the shape of the plot
    frame = cv2.imread(os.path.join(plots_folder, plots[0]))
    height, width, channels = frame.shape


    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    video = cv2.VideoWriter(output_video, fourcc, fps, (width, height))

    for plot in plots:
        path = os.path.join(plots_folder, plot)
        video.write(cv2.imread(path))

    cv2.destroyAllWindows()
    video.release()
```

我之前是用數字 index 當檔名存影圖，所以一樣先對 plot 做排序，使其符合時間上的先後順序，接著使用 cv2.VideoWrite 將先前存好的投影圖製作成影片檔，以上是投影視覺化的整體步驟

3. Explain why we need to use sdc-calibration.bag for calibration, and what issues may arise if we use sdc-moving.bag or sdc-NCTU.bag?

Ans:

可以清楚地發現 sdc-calibration.bag 和 sdc-moving.bag、sdc-NCTU.bag 最大的差異就是靜態及動態的差別，做 2D-3D 點匹配時，若 data 的空間關係是隨時間變動的，那很難去確定說找到的 image 和 lidar points data 是在同一個空間關係下的結果，也就是兩者選到的點不是真正意義上對應的點對解出來的很大機會是錯誤的結果，而 sdc-calibration.bag 的 data 空間物體關係是不隨時間改變的靜態 data，因此空間關係比較不會變動，能較確定選到的點對是真正的對應點，也因此 PnP 算出的 transformation matrix 會是較正確的。

4. Other issues:

i. The lidar and camera data frequency are different:

將 bag 中 lidar 及 camera data 存起來後，可發現 timestamp 是對不上的，且 image file 比 lidar points file 多很多，說明了 camera 的更新平率較高，這樣不能輕易得到一對一的 lidar points file 和 image file 去做投影，所以我選擇配合較少的 lidar file，讓每一筆 lidar points file 去找 timestamp 最近的 image file 來做投影，解決兩 sensor data 更新頻率不同造成的問題。

ii. Color of the image:

我發現投影後 plot 的 image 的顏色跑掉了，後來查找資料發現 opencv 的 cv2.imread 讀進來時是 BGR 形式而 plt.imshow()是 RGB 形式，若讀進來後直接畫在 plot 上，顏色會錯誤，在讀進來後要先將 BGR 轉成 RGB 形式，後續投影 plot 的 image 就能有正確的顏色了。

iii. PnP accuracy based on the 2D-3D points

在做 calibration 時發現用人工選 image 點及 points 點對應點，一定會有偏差，造成 PnP 解出之 transformation matrix 每次都有些差異，且點的稍為偏差，就可能得到較不對的 transformation matrix，使投影結果很依賴於人工取點的精準性，因此我覺得要有好的 transformation matrix 需要有更好的取對應點方法而不依賴人工選點。