SDC Homework 5 – Calibration

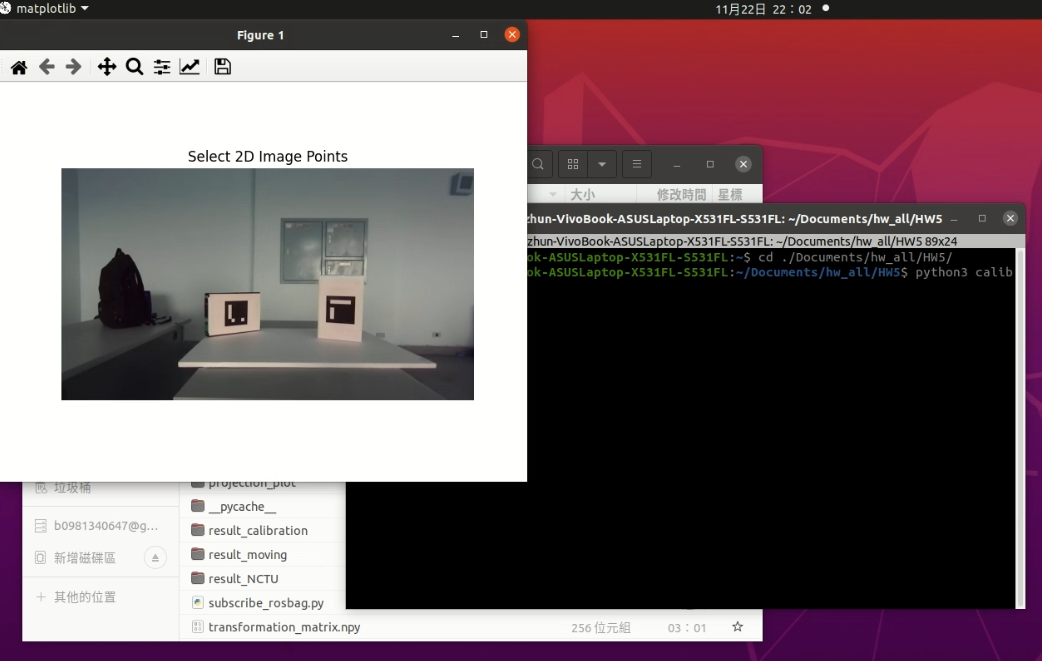
312512005 黃名諄

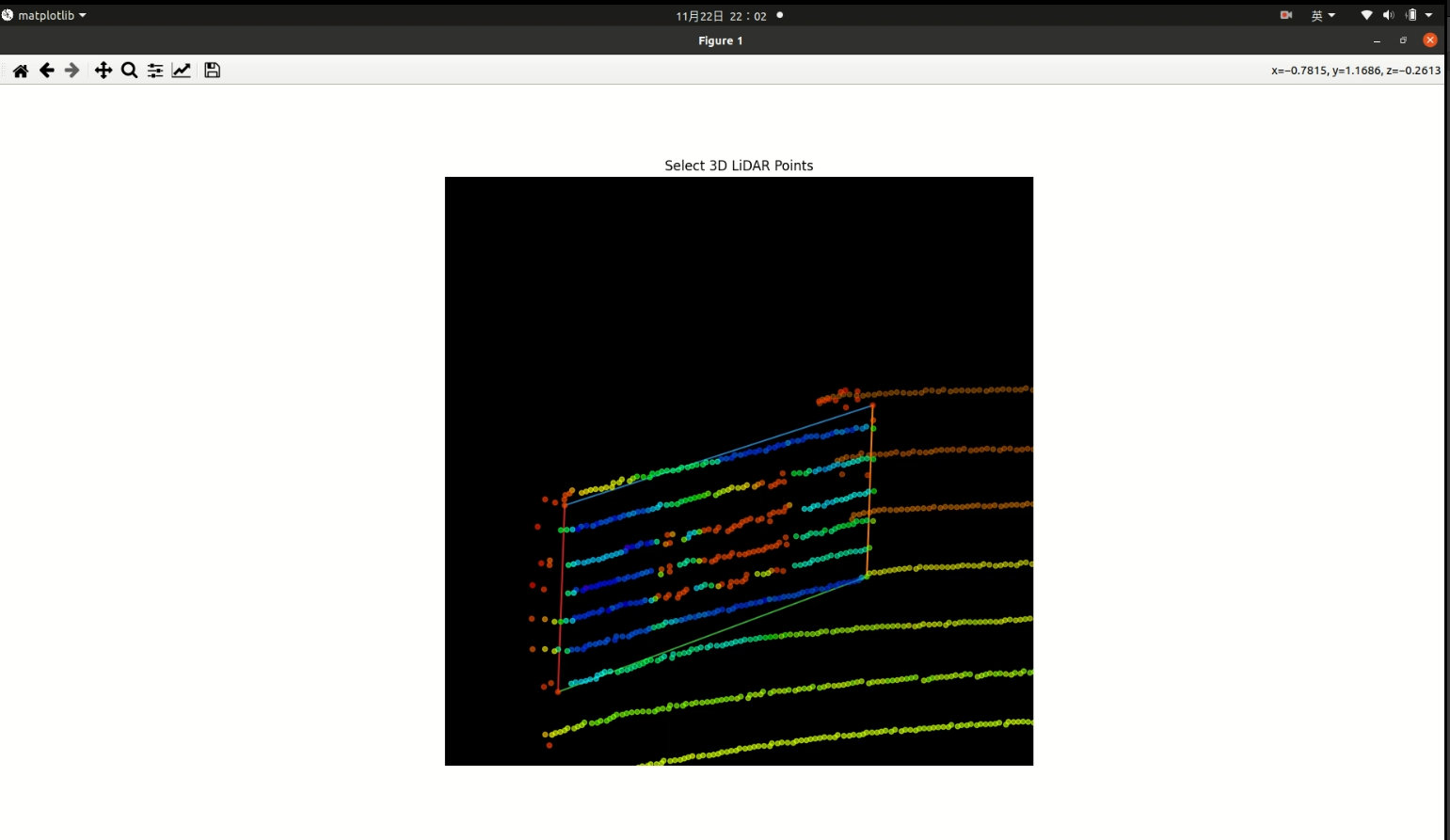
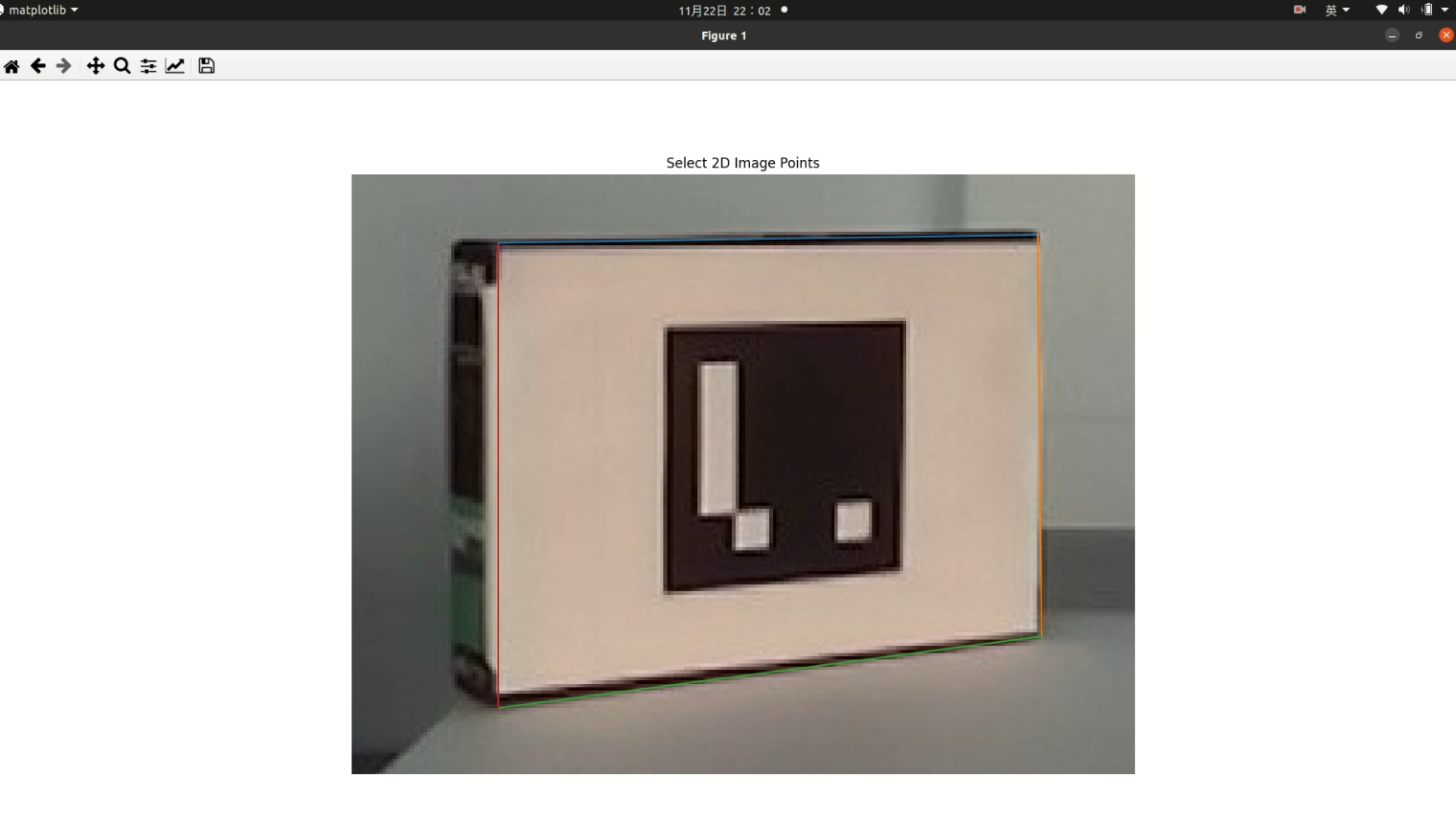
1. Screenshot of running code

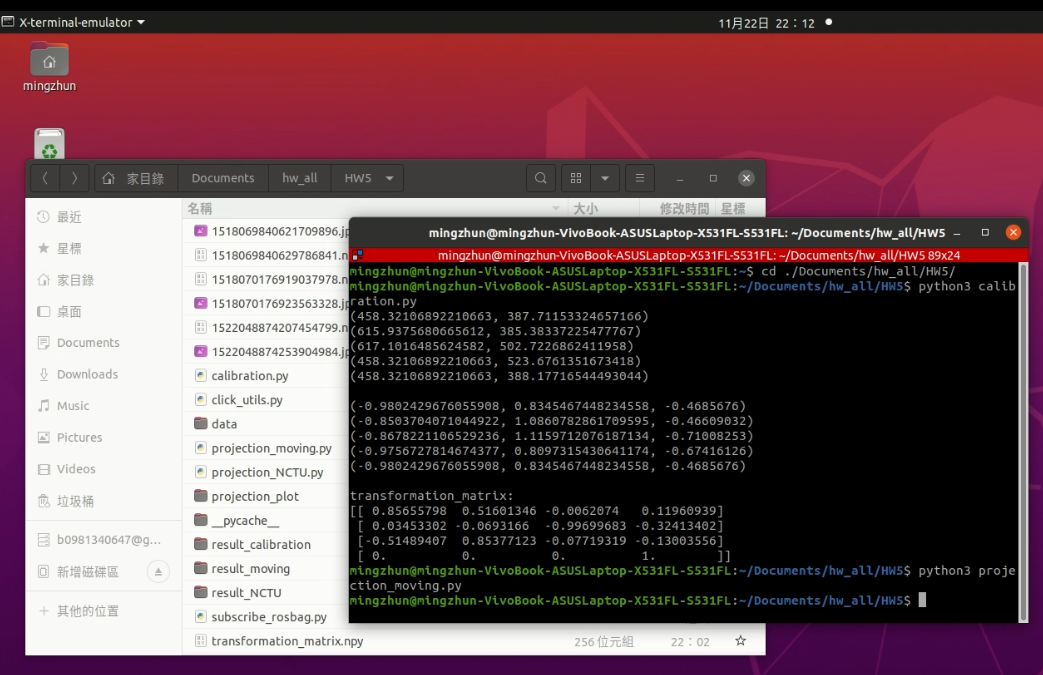
Full code running video : <https://youtu.be/32GIBamMmDA>

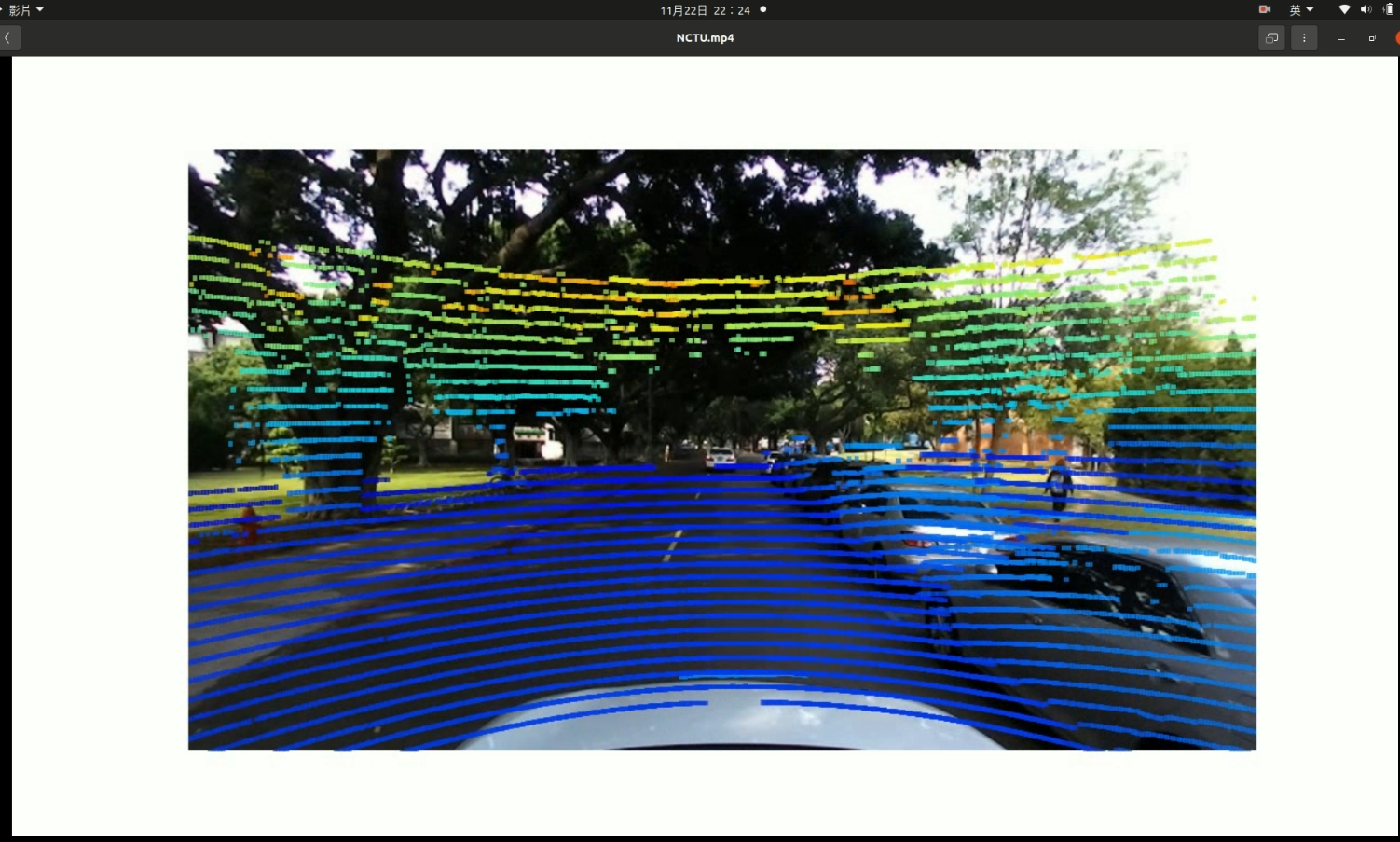
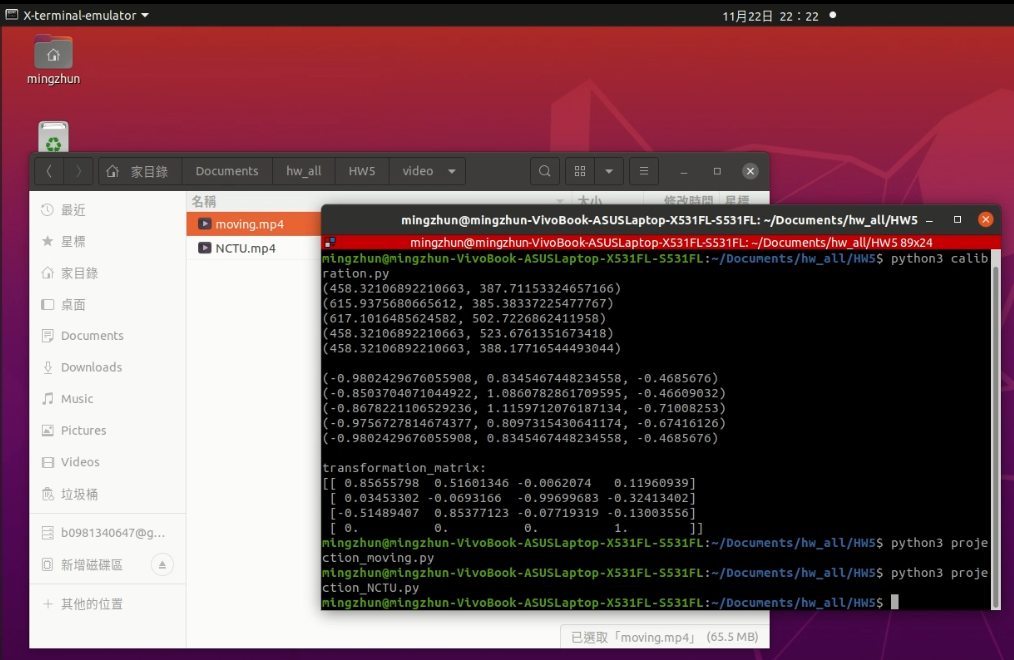
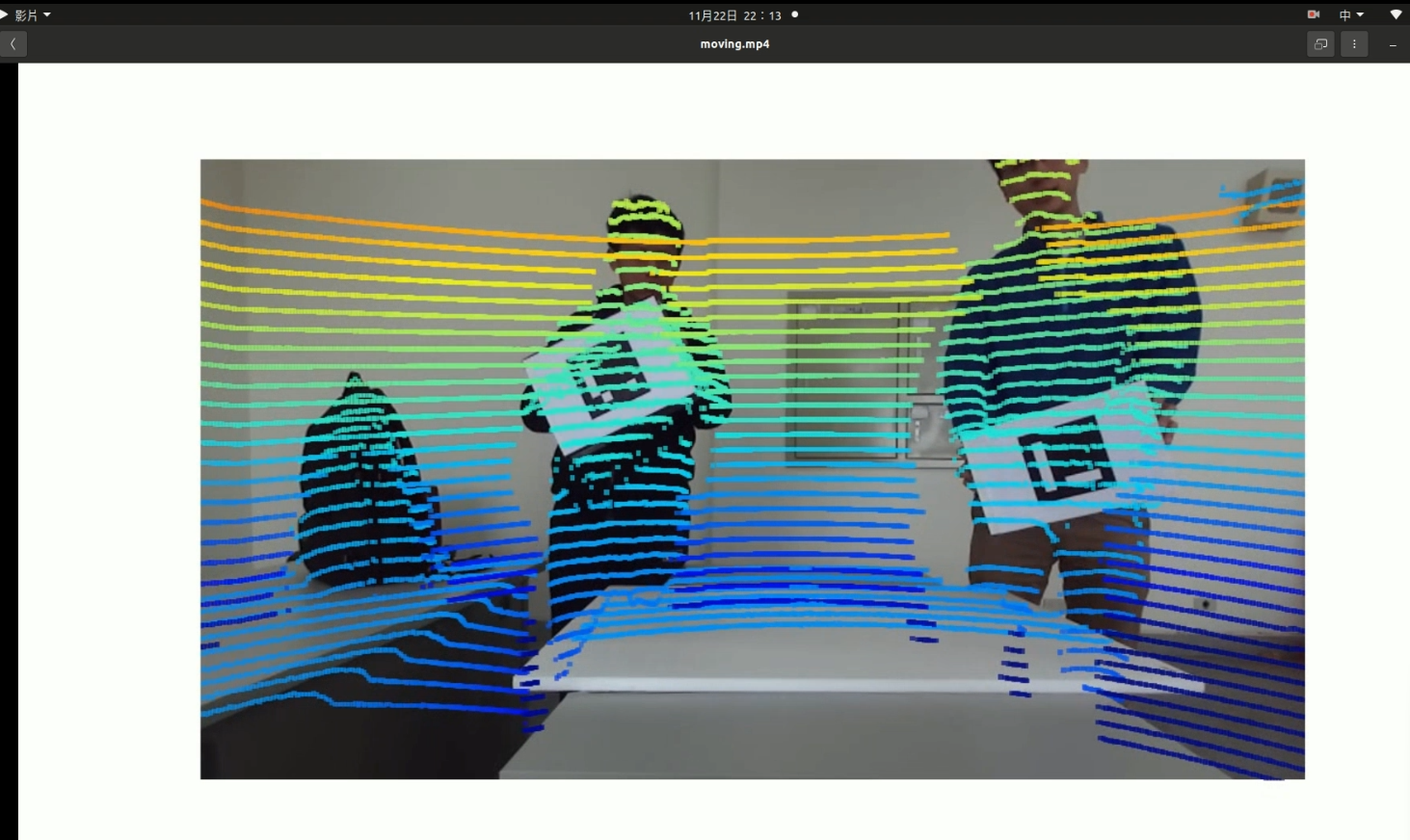
sdc-moving.bag projection video : <https://youtu.be/PhLDoMoWqJc>

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









1. Code explanation:

整體程式架構流程如下:

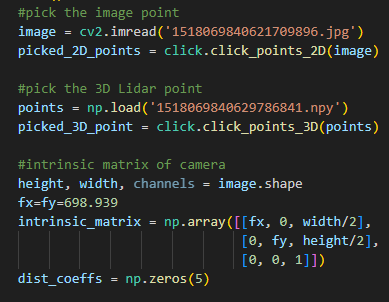
先使用subscribe\_rosbag.py將每個bag的lidar及camera data存為以時間戳為檔名之檔案，並分別存在3個不同的資料夾中

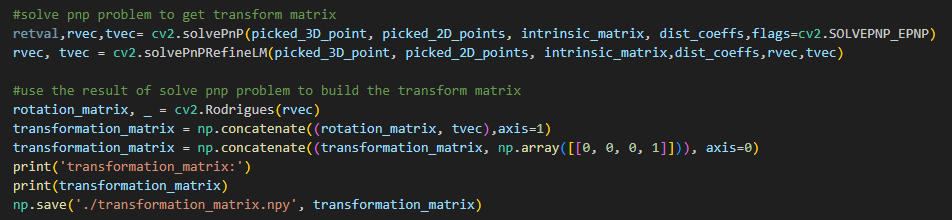
從sdc-calibration得到的data中，選時間戳相近的image及lidar point各一當作預匹配對象，送入calibration.py，其中使用click\_utils.py分別在image及point cloud上選擇4對2D-3D對應點，就能使用cv2.solvePnP求解此PnP問題，得到世界座標轉到相機座標之transform matrix

分別在projection\_moving.py及projection\_NCTU.py中，利用上一步得到之transform matrix，將已存好的sdc-moving .bag和sdc-NCTU.bag 的pointcloud data 投影到時間戳最近的image data上，並將其用plot視覺化存起來，最後再將存起來的這些plot做成影片檔

具體程式細節及問題如下說明:

1. calibration.py:

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

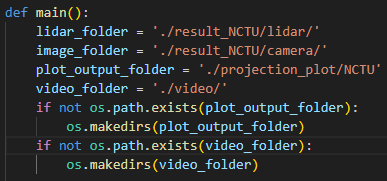
intrinsic matrix:

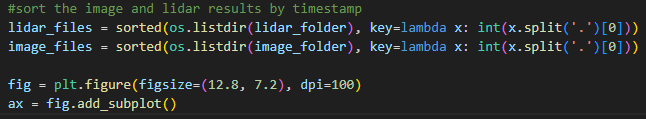
接著將選好的2D點、3D、intrinsic matrix及distortion coefficients帶入cv2.solvePnP求解PnP問題，因有4個點對，經測試後選擇SOLVEPNP\_EPNP來求解會較正確及穩定，解出之結果還會經過cv2.solvePnPRefineLM來使用non-linear Levenberg-Marquardt minimization做Pose refinement，最後有了R及t，可建構出能轉換lidar coordinate至camera coordinate 的transformation matrix:

並將其存起來給後續投影做使用。

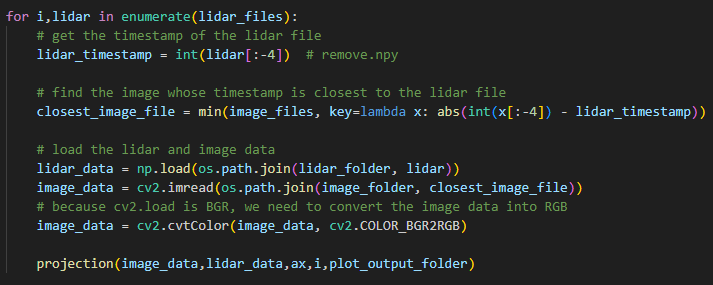
1. projection\_moving.py及projection\_NCTU.py

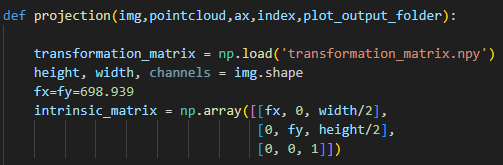
兩者邏輯相同，只是讀data的路徑不同，因此此處以projection\_NCTU.py來做解釋

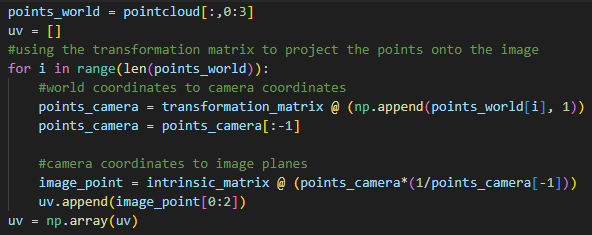


首先先前已將bag中的data存下來了，此處就是定義好從哪個路徑讀這些存下來的data，以及後續投影後的plot和轉為影片檔後要存的資料夾路徑

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

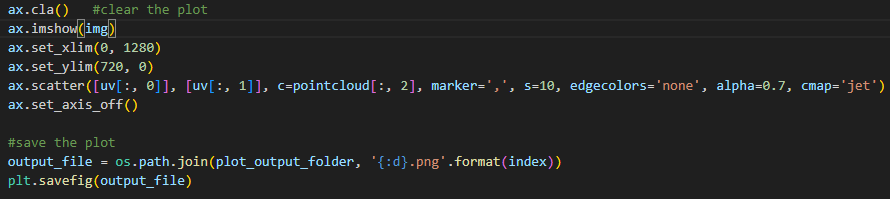


接著觀察之前存好的lidar data和camera data，會發現兩者timestamp 對不上且image data數也較lidar points data多，代表camera頻率比lidar高，所以我的處理方式為，為每一筆lidar points data file找到timestamp最相近的image file，將兩者load進來，此處還需對image做BGR to RGB的轉換，原因於問題3解釋，將此image file當作投影對象把lidar points投影上去，因此對每組lidar points data做投影，結果會有跟lidar file數一樣多的plot，投影的function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

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

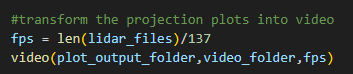
取得point的座標值，透過transformation matrix轉至camera coordinate下，數學表示如下:

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

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

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:

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:



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

1. 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。

1. Other issues:
   1. 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。

* 1. 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。

* 1. 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。