

CV Final Project Report

ITRI Group 12

Team Name: 叫這個隊名可以嗎 XD

Team Members：羅恩至、陳炯濤、溫威領、黃詩瑜、羅崇榮

1. Methodology

Method 1

實作方法：二值化畫 contours ->使用 Harris Corner 偵測角點->使用 deep learning model 取得深度資訊->pinhole model 建出 3D 點雲

Step 1. 二值化

二值化使用大津二值化法，根據大圖片或是每個 bounding box 的 histogram，來計算閾值，將每個 bounding box 處理好後，貼到原圖大小的黑圖上，結果如圖(以下均以 seq1 中的 1681710717_544461636 當例子)

```
_,gray_thr=cv2.threshold(crop.astype(np.uint8),0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

Figure.1 使用大津二值化法將圖片二值化



Figure.2 原圖(1681710717_544461636)

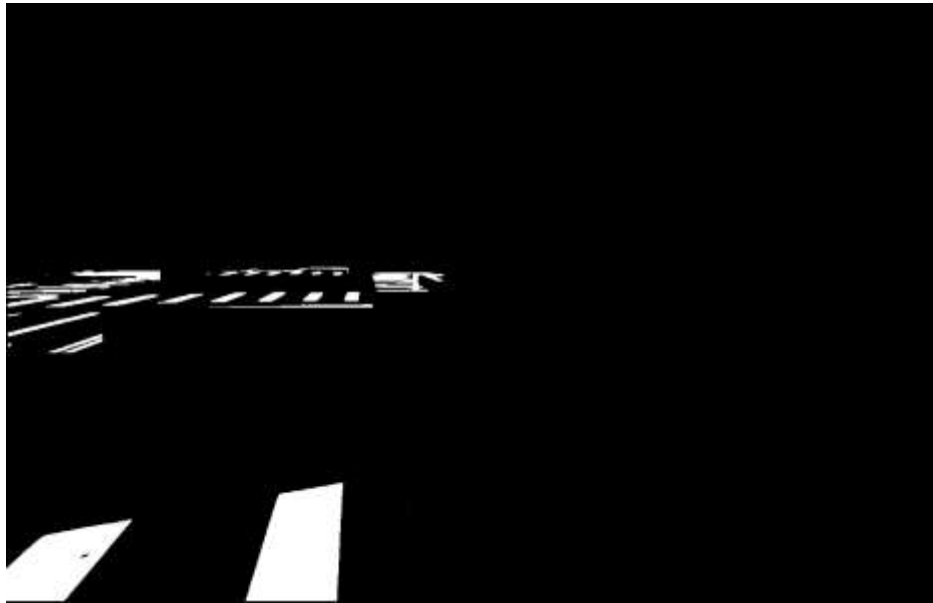


Figure.3 大津二值化_bounding box

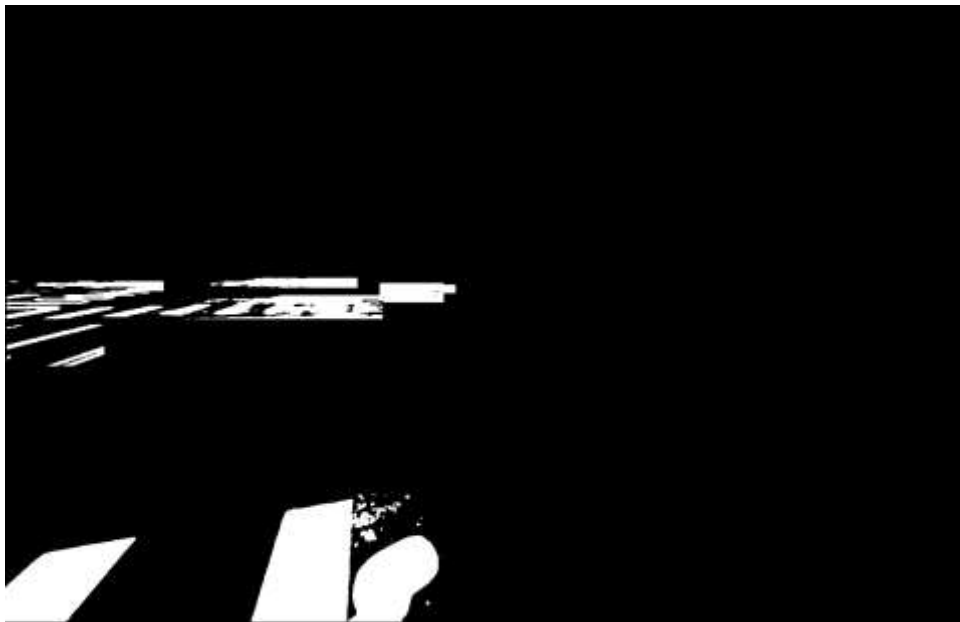


Figure.4 大津二值化_全圖

由於每個 bounding box 可能有對應的局部特徵，所以用統一的二值化效果較差，因此使用局部的二值化，即便如此，圖中仍有一些椒鹽噪點或型態不完整，所以使用 Dilate+erode 和 erode+dilate 處理這個問題。

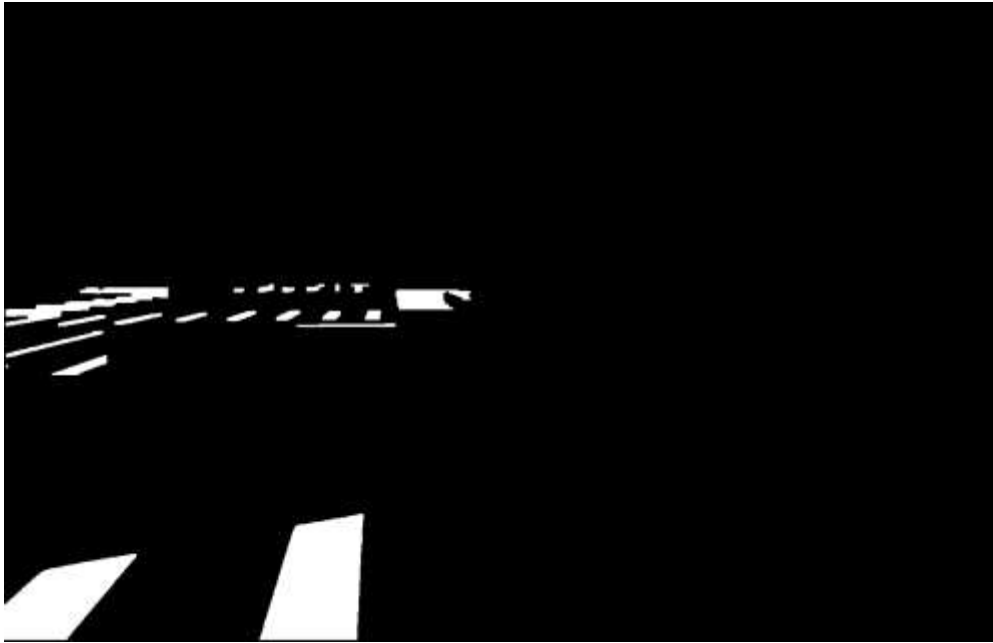


Figure.5 After dilate erode and erode dilate

由於影片一些部份有陰影，且部份視角的車子部份容易導致二值化與找角點時與標線混合在一起，因此在圖片前處理中加入過濾車子 mask 與陰影部份，把原始圖片輸入先找出 Histogram 中出現最多次的像素，判定為道路顏色，將像素值低於該像素的，全部改為道路顏色，如此一來車子就會跟道路融合從而不會判定為標線，也不會發生陰影太多導致 Histogram 像素分佈過低，而把道路跟標線融合。



Figure.6 未經前處理左下的部份有車子 mask 雜訊



Figure.7 經過前處理後，車子 mask 雜訊已被消除

Step 2. Harriscorner

在二值化完成後，使用 Harris Corner 方法找出角點，CV2 已有 Harris Corner 的相關套件，只要呼叫即可傳回各像素為角點的機率，並將大於閾值的像素設為角點，由於角點的附近應也有較高的機率為角點，因此將過於靠近的角點濾除，最後得到的結果如下 Figure.8 所示。

```
dst = cv2.cornerHarris(gray_ok,2,3,0.04)
```

```
greatmarker=np.argwhere(dst>thr_corner*dst.max())
```



Figure.8 角點結果圖

Step 3. Zero-shot Transfer by Combining Relative and Metric Depth[1]

得到 2D 角點座標資訊後，需將角點轉換成 3D 點雲，因此需要先取得每個角點該像素所在位置的深度資訊。為了獲取深度資訊，我們實作 Zero-shot Transfer by Combining Relative and Metric Depth [1]這篇論文的做法，利用他們所提出的深度資訊預測模型(ZoeDepth)，來得到每張圖片的深度資訊。

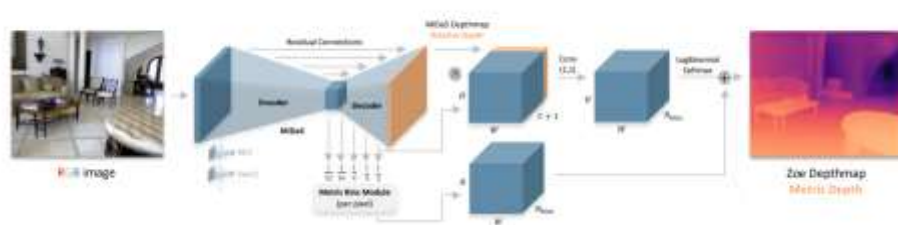


Figure.9 ZoeDepth 架構圖[1]

模型架構如上圖所示，共包含了兩個階段。第一階段為將輸入的 RGB 圖片過 encoder 與 decoder，進行相對深度的估計；第二階段則是將所提出的 metric bins module 附加在 decoder 上，得到用來做絕對深度預測的 head，以進行絕對深度的估計。透過相對深度與絕對深度的資訊，模型便能以 end to end 的方式輸出最終的深度圖。



Figure.10, 11 原圖(左)與 ZeoDepth 所得之深度圖(右)

可以看到深度的預測還不錯，大致的街景輪廓都能分辨出來(紫色為距離較近、黃色距離較遠)。

在深度圖中，每個像素的值即代表模型預測的深度。因此我們在各個角點的像素位置中，提取該像素所對應的深度值(z)，便取得角點的深度資訊。

Step 4. Pinhole Model

我們使用 pinhole model 的方式，透過助教所提供的相機內參，來得到各個角點在地圖上的實際座標資訊，之後再透過 launch 檔中提供的相機轉換參數，將所有角點投射到 baselink 上，以重建出最終的 3D 點雲。

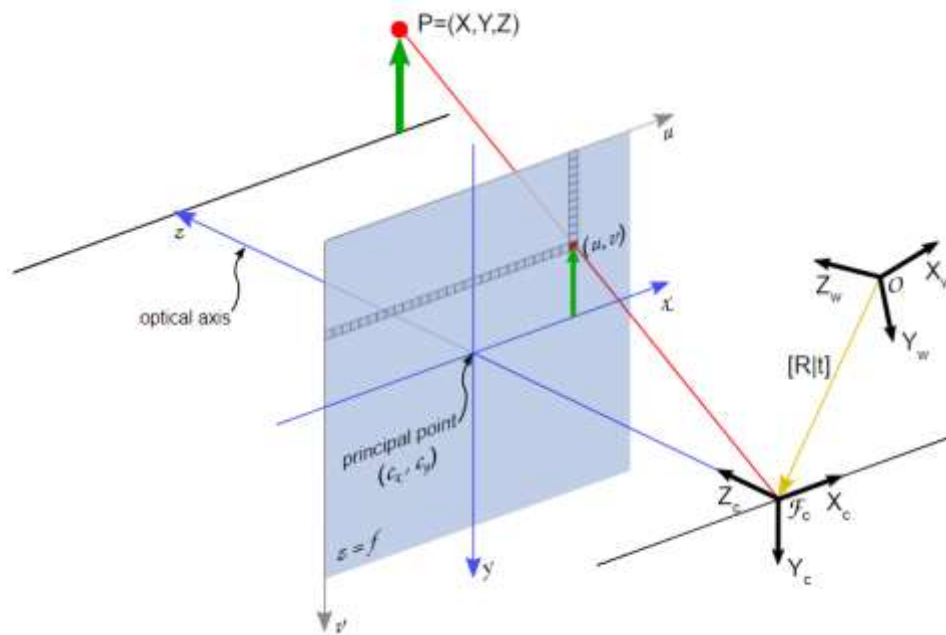


Figure.12 pinhole model 示意圖[2]

```

fx, fy = projection_matrix[0][0], projection_matrix[1][1]
cx, cy = projection_matrix[0][2], projection_matrix[1][2]

point_cam = []
for (x, y, z) in corners:
    x_cam = (x-cx) * z / fx
    y_cam = (y-cy) * z / fy
    z_cam = z
    point_cam.append([x_cam, y_cam, z_cam])

return point_cam

```

Figure.13 pinhole model 座標轉換程式碼

座標轉換的作法如上圖所示，透過 projection matrix 的 c_x, c_y, f_x, f_y ，得到各點雲的 x, y 座標，搭配先前透過 ZoeDepth 預測的深度(z)，便推出各點雲的實際位置。

之後再將點雲轉至 `base_link`，轉換方式會先從 `launch` 檔案提供的 `extrinsic parameters` 提取出 $x, y, z, q_x, q_y, q_z, q_w$ ， q_x, q_y, q_z, q_w 四元數可以計算出 3×3 的旋轉矩陣，再加入 x, y, z 資訊變成 4×4 的矩陣。將三維的座標點

乘與 4x4 矩陣即可轉換至 base_link。有些鏡頭無法一次轉到 base_link，而各鏡頭會按照以下的順序轉換：

fl : fl->f->base_link

b : b->fl->f->base_link

fr : fr->f->base_link

f : f->base_link

經過轉換之後的 3D 點雲圖如下：

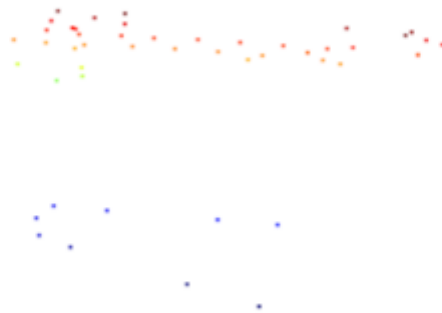


Figure.14 最終的 3D 點雲

Method 2: Rule Base Method

此任務是希望從 dataset 中資訊來建立路標的 3D point cloud，本部分嘗試方法的基本想法如下，在 dataset 每個 seq 中的 other_data 都有各時間點的 speed 和 imu 資訊，因此可以從這些資訊來計算車子的移動軌跡和得知每個 frame 中車子位置座標，並從相對應 frame 中 image 來確認有哪些路標物件，最後根據大部分道路的標線規則將已先建立好的路標點雲(如 Figure.15)在當前位置貼上。

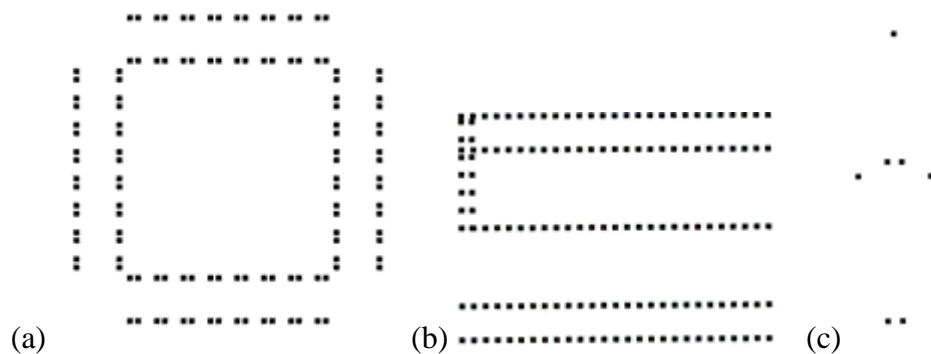


Figure.15-a、15-b、15-c 分別為斑馬線、道路與停止線、箭頭的點雲

以 seq2 為例，以下是 visualize 的結果，可以看到 Figure.16 中藍色線條為 seq2 的車子行進軌跡，而綠色點是根據軌跡貼上的點，一般來說路標都在同一平面，因此只要找到 xy 平面的點座標，再把 z 設為同一平面即可。

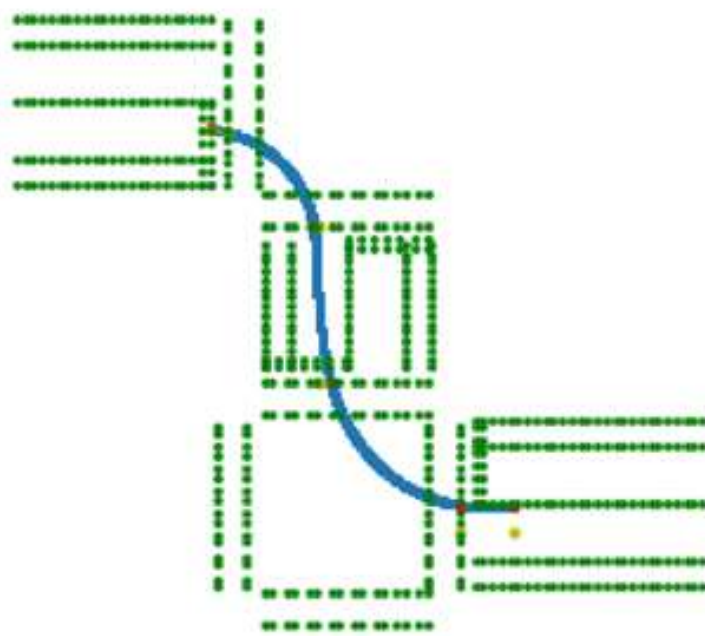


Figure. 16

Figure.17-a、17-b 是從綠點 xy 座標建立的點雲，而 Figure.17-c、17-d 為對應的 sub_map ground truth，可以看到兩者點雲的結果還蠻相似的。

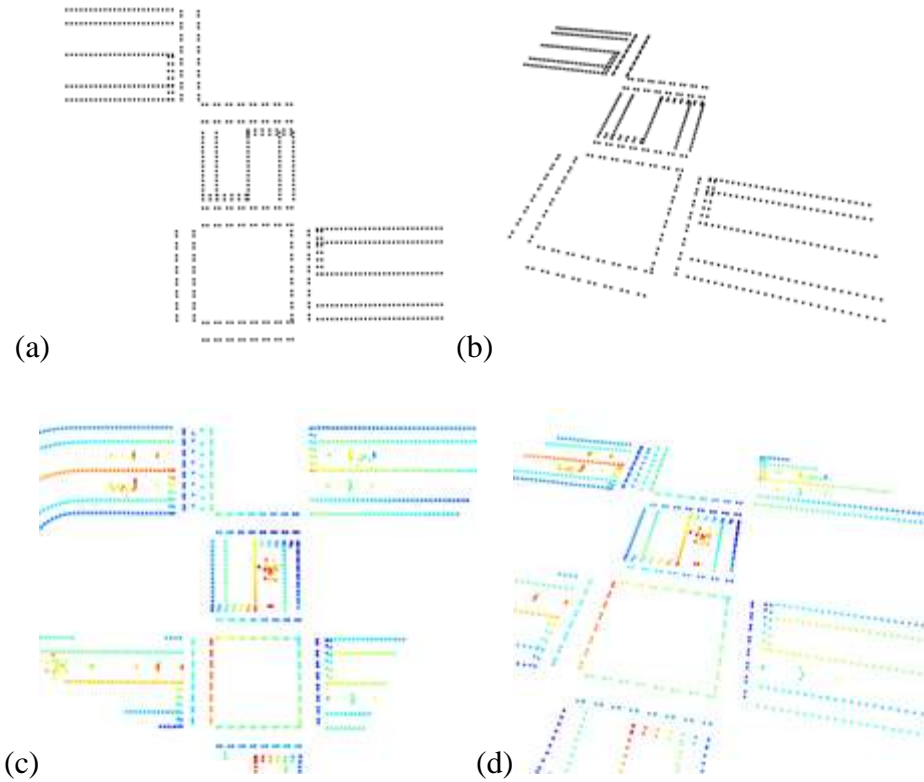


Figure.17-a、17-b、17-c、17-d

最後再取各個 frame 時間的位置 10 公尺內的點，作為當下 frame 時間點的 point cloud，如 Figure.18-a 所示，是 Figure.17-a 的其中一部份。但預設的標線點座標仍與 sub_map ground truth 的座標有些差異，因此將 predict 點座標複製數份做各方向小幅度的 offset 位移 (Figure.18-b)，可以使 ICP 有更好的 matching，經由這個步驟可將 seq2 做 ICP(assert!=0)的 threshold 從 1.8 降為 0.35，ME 也從原先 2.05141 降為 0.52867。

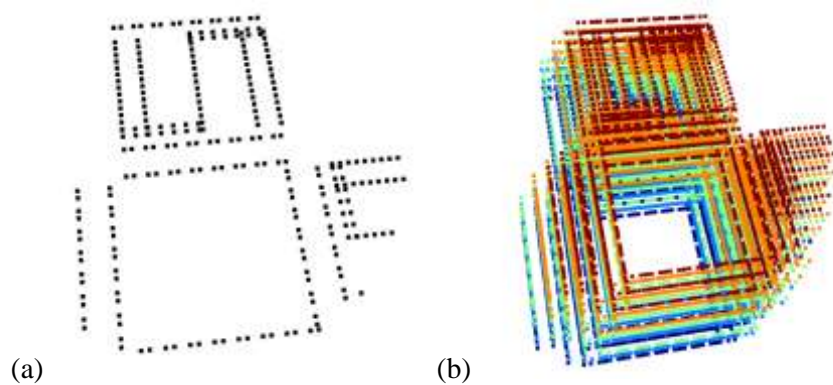


Figure.18-a、18-b

此方法預設的道路限制很多且效果不如預期，並非正統作法因此並不採用，單純實驗性質。此方法最終 ME 如下： $seq1 = 0.49661$ 、 $seq2 = 0.52867$ 、 $seq3 = 0.48224$ 。

Method 3: Slam

實作方法：找二維角點->用 slam [3]重建出 3D 點雲，此亦為我們使用的最終版本。

Step 1

因為使用 slam 的關係，讀取影像的順序要是前後連貫的，不能單純依照 all_timestamp.txt 內的順序，因此先將所有時間的影像按照 camera info 的 fl/b/fr/f 鏡頭做分類，再讀取每個鏡頭的 timestamp，這樣影像的順序才會是前後連貫的。

Step 2

將該影像依照 detect_road_marker.csv 提供的範圍做分區，分別抓取角點後再將分區內的角點全部蒐集在一起成為該影像的角點。找角點的方式，首先將彩色影像轉換為灰度圖像，以便後續處理，接下來對圖像執行最小最大濾波，這可以幫助消除圖像的陰影。

最大最小值濾波：首先排序周圍像素和中心像素值，將中心像素值與最大和最小值比較，如果比最小值小，則替換中心像素為最小值，如果比最大值大，則替換中心像素為最大值。

因為影像背景為柏油路較暗，而且斑馬線等物體較亮，會先執行最小濾波再執行最大濾波。執行完濾波後，獲得的值不在 0-255 範圍內，因此必須歸一化，使用背景減法獲得最終陣列，該方法將原始圖像減去濾波圖像，以獲得去除陰影的最終圖像。

消除圖像的陰影後，使用 `cv2.threshold` 對圖像進行二值化處理，將圖像轉換為黑白二值圖像，二值化是為了將圖像轉換為只包含黑色和白色像素的形式，以方便後續的形態學操作。我們對圖像進行膨脹和侵蝕，有助於消除圖像中的噪點，平滑輪廓。接下來，使用 `cv2.findContours` 找到二值圖像中的斑馬線等輪廓，然後將圖像轉換回彩色圖像，以便在圖像上進行可視化。之後遍歷所有的輪廓，選取那些面積大於一定數值的輪廓，作為我們找到的角點座標。

Step 3

將每張影像找到的角點進行特徵提取，使用 ORB 特徵檢測器的 `compute` 方法來計算特徵點的描述子，將結果存儲在 `kpts` 和 `des` 變數中，再將找到的特徵點進行匹配。每兩個前後影像的特徵點使用 Brute-Force Matcher 的 `knnMatch` 方法對當前影像的特徵描述子和上一張影像的特徵描述子進行匹配，並返回匹配結果中的特徵點的座標。

使用 `slam` 將特徵點轉成 3 dimensional，依照不同 `fl/b/fr/f` 鏡頭給予對應的 `camera_matrix` 作為 `intrinsic parameter`。下一步，我們需要對點集進行縮放和變換。首先，將點集的中心平移到原點，然後根據平均距離進行縮放。接下來，計算從一個影像到另一個影像的變換矩陣，接收兩組點的座標，然後使用這些座標來建立一個矩陣，並進行奇異值分解（SVD），將本質矩陣進行正規化，根據分解結果計算正規化的相機投影矩陣。最後就是重建單個三維點，這需要接收兩個影像中的對應點和兩個相機的投影矩陣，然後根據這些資訊進行三角測量等資訊計算並返回重建的三維點的座標。

Step 4

將點雲轉至 `base_link`，轉換方式與 Method 1 作法雷同，在此不再贅述。

Visualization:

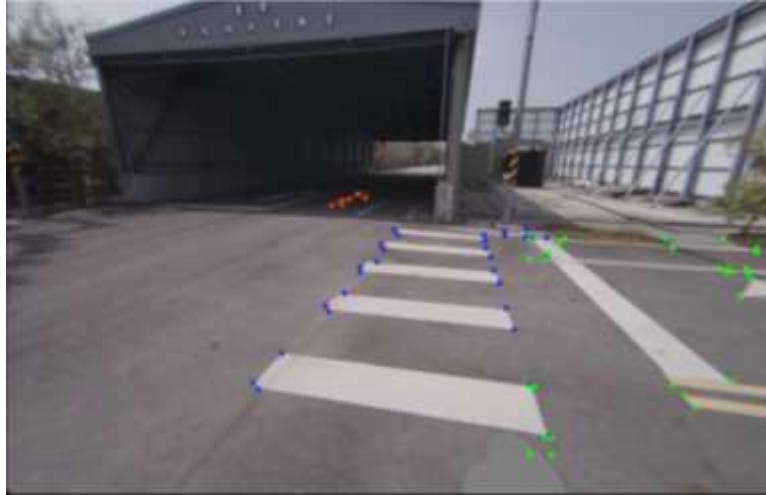


Figure.19 抓取到的 2D 角點

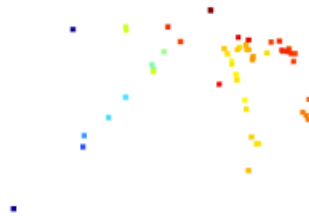


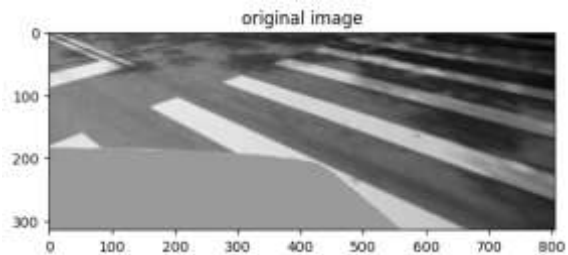
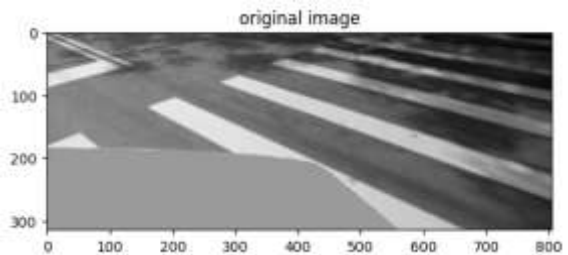
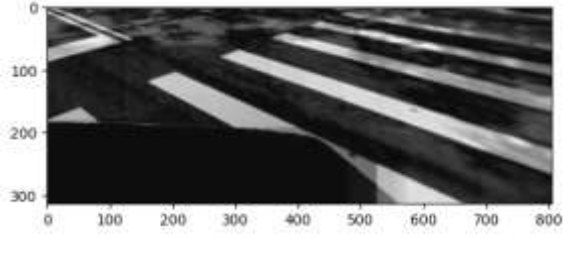
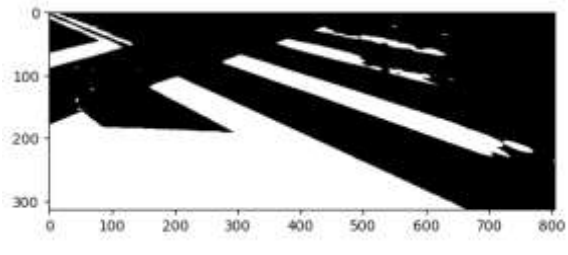
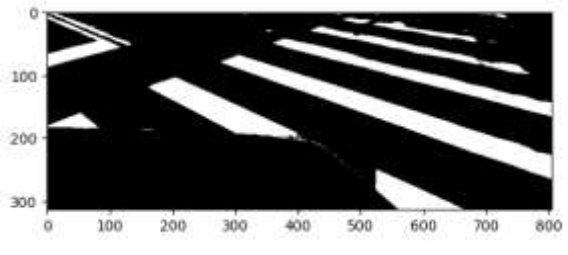
Figure.20 最終的 3D 點雲

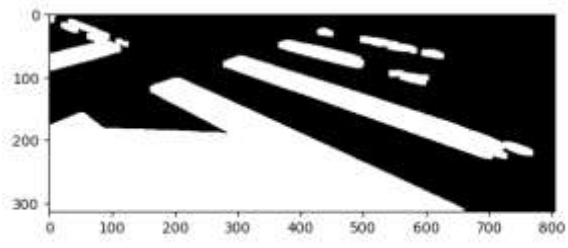
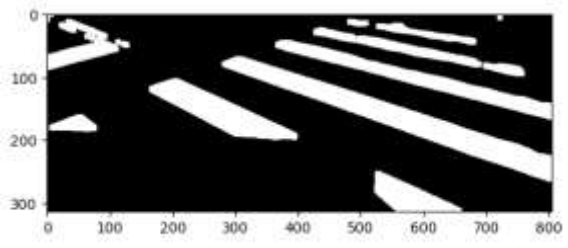
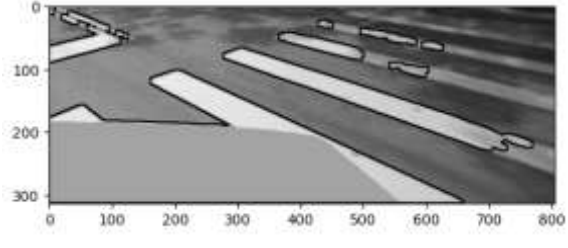
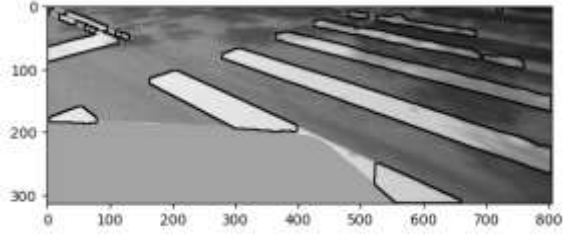
2. Experiments & Analysis

針對有無使用最大最小值濾波進行實驗：

左方為無使用最大最小值濾波，只對影像依序進行

cv2.threshold->cv2.erode/cv2.dilate->cv2.findContours 的結果，右方則是在
cv2.threshold 之前先經過最大最小值濾波。

無濾波	有濾波
original image	original image
	
X	min_max_filtering
X	
cv2.threshold	cv2.threshold
	

cv2.erode/cv2.dilate	cv2.erode/cv2.dilate
	
cv2.findContours	cv2.findContours
	

表一 最大最小值濾波實驗結果

從上方表格可看出經過濾波後的影像，找出的斑馬線輪廓更為精準，代表右上區域的陰影對原圖的影響確實有被去除。

3. Ablation Study

Method	ME Score
Method 1	3.28878
Method 2	0.45588
Method 3	0.41685

表二 三種方法 ME 結果比較

可以看到 Method 3 使用 slam 的結果有最好的 Performance，因此我們採用此方法作為最終結果。

分析三者的結果，Pinhole model 因為只針對每個 frame 中的角點去做點雲重建，而未考慮前後 frame 的資訊，且用來推估實際位置的深度資訊(z 值)為深

度學習模型預測出來的，數值可能和相機實際與角點的距離仍有差距，因此重建出來的點雲圖較不理想，像是重建完的斑馬線點雲變成波浪狀，而 ME 的分數也是三者最高，數值高達 3.28878。第二種方法雖然可以見出較為完整的點雲，但如先前提到，此方法預設的道路限制較多，且非正統作法，因此不予採用。第三種方法，除了在偵測角點多使用了最大最小值濾波讓偵測出來的 contours 更為完整之外，加上改用 slam 來重建 3D 點雲，由於 slam 的做法有考慮 frame 與前後 frame 的關係，因此重建出來的點雲明顯較為完整，ME Score 也是三者最佳，達到 0.41685。

4. Reference

- [1] Bhat et al. ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth. In CVPR, 2023.
- [2] Pinhole model https://docs.nvidia.com/vpi/appendix_pinhole_camera.html
- [3] Slam-python <https://github.com/filchy/slam-python/tree/master>
- [4] <https://zhuanlan.zhihu.com/p/335777554>
- [5] <https://medium.com/@ckwang19/slam%E5%AD%B8%E7%BF%92%E4%B9%8B%E8%B7%AF-1-%E7%B0%A1%E4%BB%8B-1465bc319f3>
<https://zhuanlan.zhihu.com/p/388164543>