

Final Project: DeepVo

R07921001 李尚倫 R07522625 李佳蓮 R07522653 陳健倫

Our work: https://github.com/Shining-Zone/VFX_Final

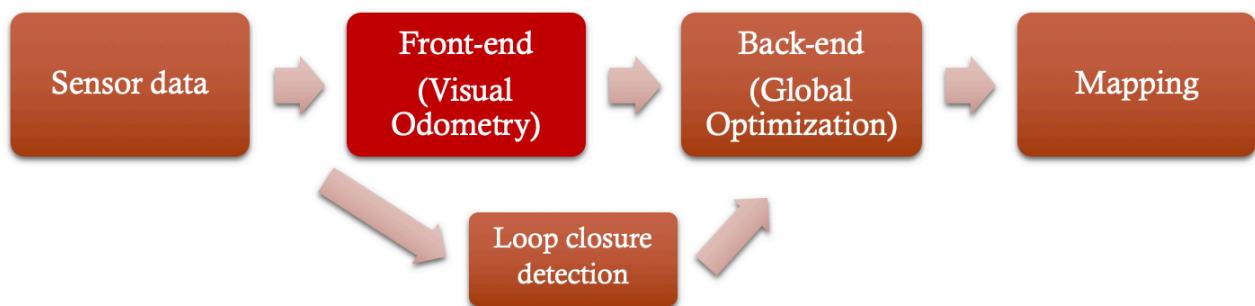
I. Project description

在這次的 Final，我們參考這篇 Paper: DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks[1]，發表在 2017 年的 ICRA，是 robotics 界頂級的會議，其中提出了嶄新的方法，透過 end-to-end learning 的方式來解 visual odometry 的問題，是 learning based 解決 visual odometry 的始祖，但作者並沒有釋出 official 的 source code。而我們的目標就是想實作這篇 Paper 還原出 paper 裡的實驗結果，並透過 ROS Rviz 來重現三圍中的座標 pose。

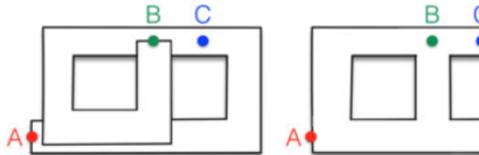
II. Implementation

- 實作論文 DeepVo 的 Model，自製 DataLoader 來做資料處理，train 出自己的 model
- 自製 ROS node，將預測結果在 Rviz 三維空間中呈現。
- 自行實際錄影，生成 6 支影片用 Blender 切割作為 Dataset 來做實驗。

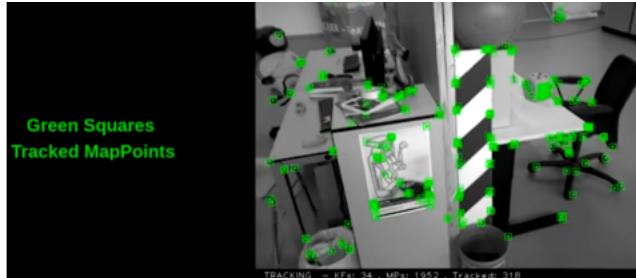
III. Recall SLAM (or SFM)



SFM(Structure from motion)或在 robotics 領域內我們稱 SLAM(Simultaneous localization and mapping)，大致上的演算法設計都經過上述 pipeline，有 sensor data input 進來，進行前端的里程計計算，得知機器人當下的 pose 與 local map，之後再經過回環檢測和後端全局優化，得到一個全局一致的地圖。



(回環檢測 loop closure, 左邊為沒有加 · 右邊有加)



ORB-SLAM

ORB-SLAM 是一個最為經典的 visual SLAM 代表，

其中紅色部分就是 local map，而藍色部分就是相機的 pose。

IV. Visual odometry: traditional vs learning based

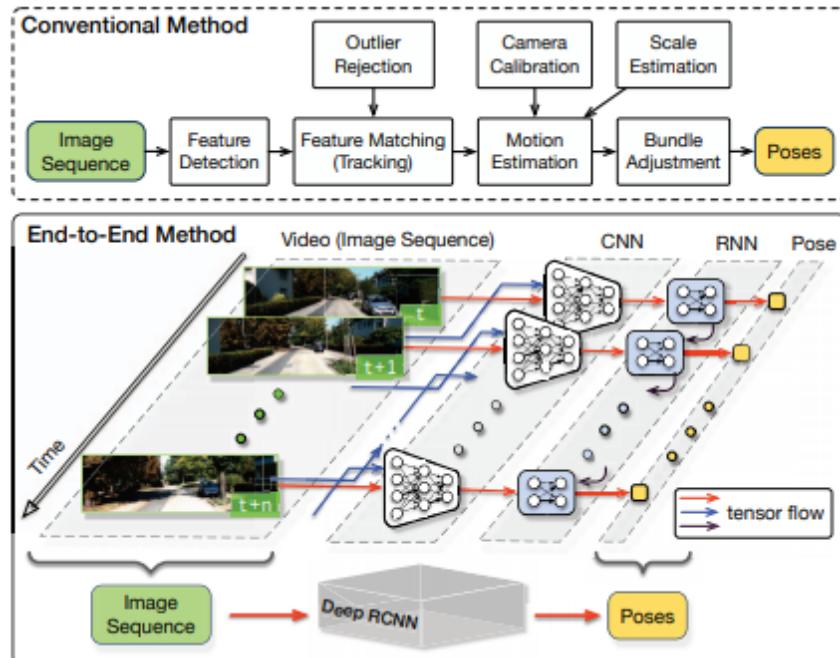


Figure 1 Visual Odometry

傳統的 Visual odometry 是個應過精密設計好的演算法，pipeline 大致上為 Feature Detection 後，進行 Feature Description，然後 Feature Matching，最後進行 Motion Estimation 並且優化。而各個步驟有很多種成熟的方法可以代換，進而衍生出了許多不同的 VO 演算法。

Feature Detection: Harris, SIFT, SURF, FAST, ORB...

Feature Description: Harris, SIFT, BRIEF...

Feature Matching: Brute-Force, FLANN...

Motion Estimation: Epipolar constraint, Triangulation, Perspective-n-Point(PnP),

Iterative Closest Point(ICP)...

Non-linear Optimization: Bundle Adjustment(BA), ...

Pose: Quaternion, Euler angels, Rotation matrix ...

而 DeepVO 則是提出了新的一種方式，透過 end to end training 的方式把中間用 RCNN 的黑盒子罩著，希望能夠餵一串影像進去，得到一串 pose 出來。

V. DeepVO Model

下圖是 DeepVO 這個架構，是用 CNN+RNN 作為 RCNN 的一個 Model 來做 Training，CNN 的部分使用網路現有的 Flownet 來作為 Pretrain Model，RNN 的部分則是使用 pytorch 內建的 LSTM，Optimizer 使用 Adam，learning rate 為 0.0005，batch size 為 8，Epoch 為 250，比較需要注意的是這邊影像處理時，0-255 的顏色區間必須改為 -0.5 到 0.5 之間，這是因為 Flownet 在訓練時就是對照片做這樣的轉換，因此也要跟著做，Pretrain model 才有更大的用意。

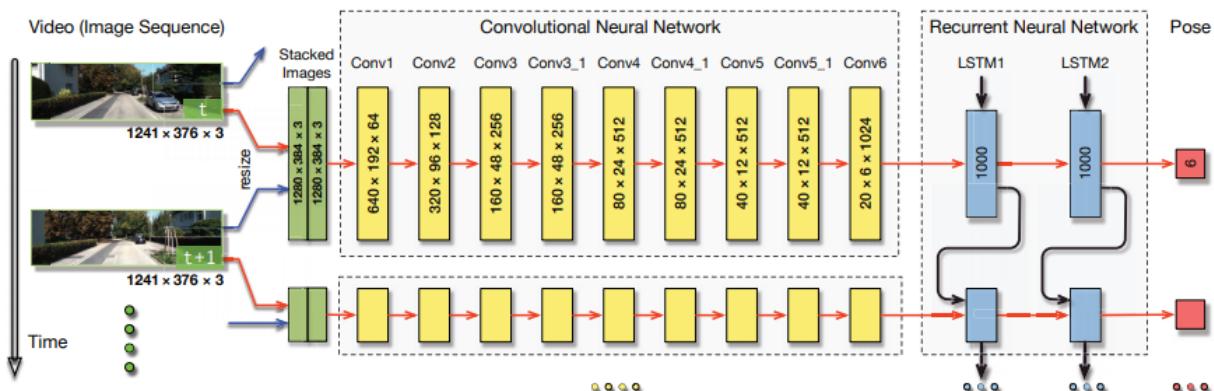


Figure 2 DeepVO model

VI. DataLoader

本 Model 在預測時的 Pose，會有坐標系的問題，所以在 Data load 進來的時候，必須在 ground truth 的部分做微處理。假設我們現在一個 batch 有 7 張 frame，就是

$1(\text{batch}) * 7(\text{frame}) * 3(\text{RGB}) * \text{img_w} * \text{img_h}$ ，在這樣的狀況下，我們把第一個 pose 視為原點，就是沒有旋轉，同時 translate 為 0，而後六張 frame 必須減去第一張 frame 的位置，變成相對坐標系的數值，再來就是要把第 frame 張的照片剪掉 frame-1 張的資訊，變成以上一張為基底座標的方式，才能拿來做

訓練，也就是每一個 predict 都是把上一張照片作為原點坐標系去預測下一張所相對移動多少，而這就是此 Model 比較需要注意的小細節。

解決 Pose 的問題後，再來就是照片的部分，實際做法我們是把[1:]跟[:-1]這樣的照片 concatenate 在一起，也就是將 2~7 張與 1~6 張疊在一起(如下面示意圖)，作為這個 model 的 input，dimension 為 $1(\text{batch}) * 6(\text{frame}) * 6(\text{RGB+RGB}) * \text{img_w} * \text{img_h}$ ，可以看到改變的地方是 frame 從 7 變成 6，因為去了頭尾的張數，而 RGB 的通道變成 6，原因就是多了一組照片，所以疊的地方是 RGB 的通道。如此一來 input 為 $\text{batch} * 6(\text{frame}) * 6 * \text{img_w} * \text{img_h}$ 、output 與 label 為 $\text{batch} * 6(\text{frame}) * 6(\text{pose})$ ，這樣



就能拿來做訓練。

Figure 3 Dataloader 擷取示意圖

而在照片擷取的部分還有的細節就是，我們可以用 overlap 的技巧來增加 data 量，假如說資料有 1-15 張，overlap=1，每個 batch 所要拿的 frame 數為 3，那麼就是 1-3、3-5、4-6，這樣子拿下去，就能讓 Model 有更多的 Data 來訓練。

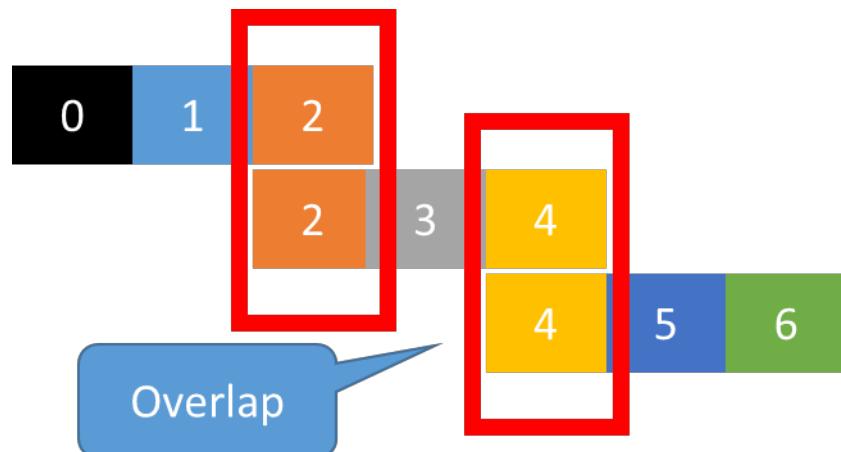


Figure 4 overlap 示意圖

VII. KITTI dataset

KITTI Dataset: Odometry benchmark

A. Geiger, P. Lenz, C. Stiller, R. Urtasun. "Vision meets robotics: The KITTI dataset", *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231-1237, 2013.

	Method	Setting	Code	Translation	Rotation	Runtime	Environment	Compare
1	RLO		code	0.00 %	0.0000 [deg/m]	0.05 s	GPU @ 1.0 Ghz (C/C++)	<input type="checkbox"/>
M. Dimitrievski., M. Dimitrievski., D. Hamme., D. Hamme., P. Velaert., W. Philips. and W. Philips.: Robust Matching of Occupancy Maps for Odometry in Autonomous Vehicles. Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: VISAPP, (VISIGRAPP 2016) 2016.								
2	V-LOAM	☒		0.55 %	0.0013 [deg/m]	0.1 s	2 cores @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
J. Zhang and S. Singh: Visual-Lidar Odometry and Mapping: Low drift, Robust, and Fast. IEEE International Conference on Robotics and Automation (ICRA) 2015.								
3	LOAM	☒		0.57 %	0.0013 [deg/m]	0.1 s	2 cores @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
J. Zhang and S. Singh: LOAM: Lidar Odometry and Mapping in Real-time. Robotics: Science and Systems Conference (RSS) 2014.								
4	IMLS-SLAM++	☒		0.61 %	0.0014 [deg/m]	1.3 s	1 core @ >3.5 Ghz (C/C++)	<input type="checkbox"/>
5	SOFT2	☒		0.65 %	0.0014 [deg/m]	0.1 s	2 cores @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
I. Cvitić, J. Česić, I. Marković and I. Petrović: SOFT-SLAM: Computationally Efficient Stereo Visual SLAM for Autonomous UAVs. Journal of Field Robotics 2017.								
6	IMLS-SLAM	☒		0.69 %	0.0018 [deg/m]	1.25 s	1 core @ >3.5 Ghz (C/C++)	<input type="checkbox"/>
J. Deschaud: IMLS-SLAM: Scan-to-Model Matching Based on 3D Data. 2018 IEEE International Conference on Robotics and Automation (ICRA) 2018.								
7	MC2SLAM	☒		0.69 %	0.0016 [deg/m]	0.1 s	4 cores @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
F. Neuhaus, T. Koss, R. Kohnen and D. Paulus: MC2SLAM: Real-Time Inertial Lidar Odometry using Two-Scan Motion Compensation. German Conference on Pattern Recognition 2018.								
36	VINS-Fusion	☒	code	1.09 %	0.0033 [deg/m]	0.1s	1 core @ 3.0 Ghz (C/C++)	<input type="checkbox"/>
T. Qin, J. Pan, S. Cao and S. Shen: A General Optimization-based Framework for Local Odometry Estimation with Multiple Sensors. 2019.								
42	ORB-SLAM2	☒	code	1.15 %	0.0027 [deg/m]	0.06 s	2 cores @ >3.5 Ghz (C/C++)	<input type="checkbox"/>
R. Mur-Artal and J. Tardos: ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. IEEE Transactions on Robotics 2017.								
54	RTAB-Map	☒	code	1.26 %	0.0026 [deg/m]	0.1 s	1 core @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
79	VISO2-S	☒	code	2.44 %	0.0114 [deg/m]	0.05 s	1 core @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
A. Geiger, J. Ziegler and C. Stiller: StereoScan: Dense 3d Reconstruction in Real-time. IV 2011.								
102	VISO2-M		code	11.94 %	0.0234 [deg/m]	0.1 s	1 core @ 2.5 Ghz (C/C++)	<input type="checkbox"/>
A. Geiger, J. Ziegler and C. Stiller: StereoScan: Dense 3d Reconstruction in Real-time. IV 2011.								
106	DeepVO			24.55 %	0.0489 [deg/m]	1 s	1 core @ 2.5 Ghz (Python)	<input type="checkbox"/>

KITTI dataset 是一個 2013 年發表的 benchmark，上面有許多著名的 odometry 或 slam 演算法，而 DeepVO 也榜上有名，雖然名次非常靠後，但其之後延伸出的 paper 有不少還不錯的 learning based 演算法，有排在前段班。

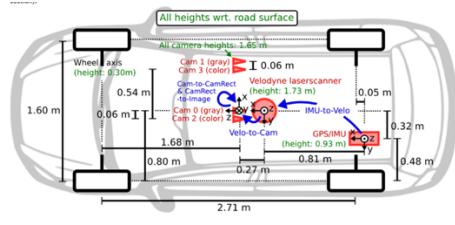


train on KITTI dataset video: 00, 01, 02, 05, 08, 09

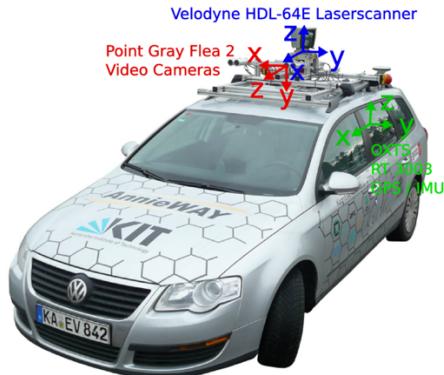
valid on KITTI dataset video: 04, 06, 07, 10

test on KITTI dataset video: 04, 05, 07, 09, 10

而我們使用其中城市街景的 dataset 00, 01, 02, 05, 08, 09 的 camera3 的 rgb image 來當 training data，其餘則當作 validation。

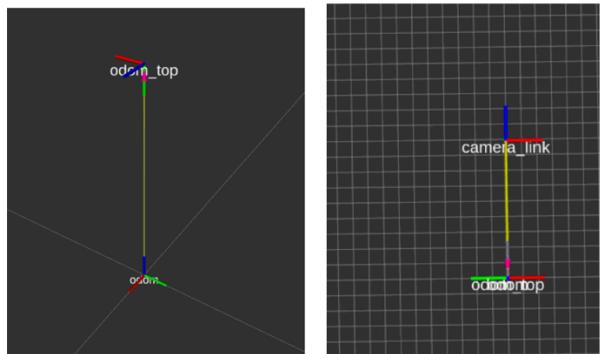


This figure shows our fully equipped vehicle:



Monocular Visual Odometry

- We only using Cam3 image
- Camera pose Z axis is forward



Original pose and camera pose on Our visualizer on Rviz

而相機的做表則以垂直鏡頭的方向為 z 方向，左右為 x 方向，上下為 y 方向，以此作為我們預測出來的 pose 的 coordinate，並同理套用至 visualizer 的座標假設中。

VIII. Our dataset

我們共收集六組影片，其中

room 使用手機拍攝徒步移動軌跡；

ntu、campus1、campus2 使用手機固定於腳踏車上，並繞行台大一周；

NTU1、NTU2 使用 GoPro 安全帽並騎機車繞行台大一周，如下圖所示。



Figure 5 單車繞行台大校園 (ntu)

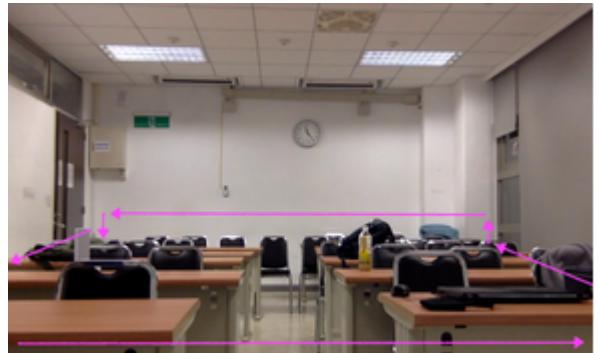


Figure 6 教室內步行一圈 (room)



Figure 7 椰林大道直線 1 (campus1)



Figure 8 椰林大道直線 2 (campus2)



Figure 9 機車繞行台大一周 (split to ntu3, ntu4)



然後我們使用 Blender video sequence editor 對切割影像作切割，並調整影像的尺寸以及取樣頻率 (frame step)等等，和作業三的前置步驟相似。

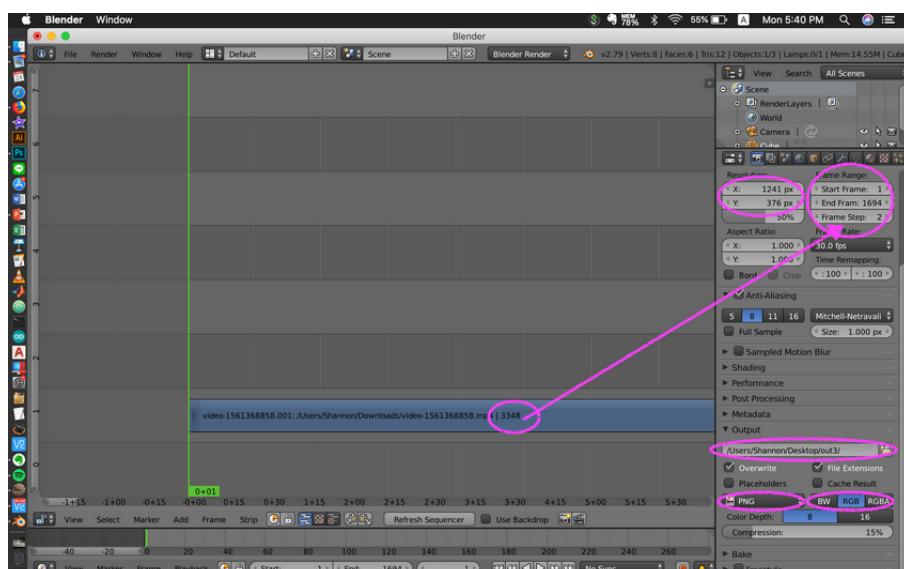
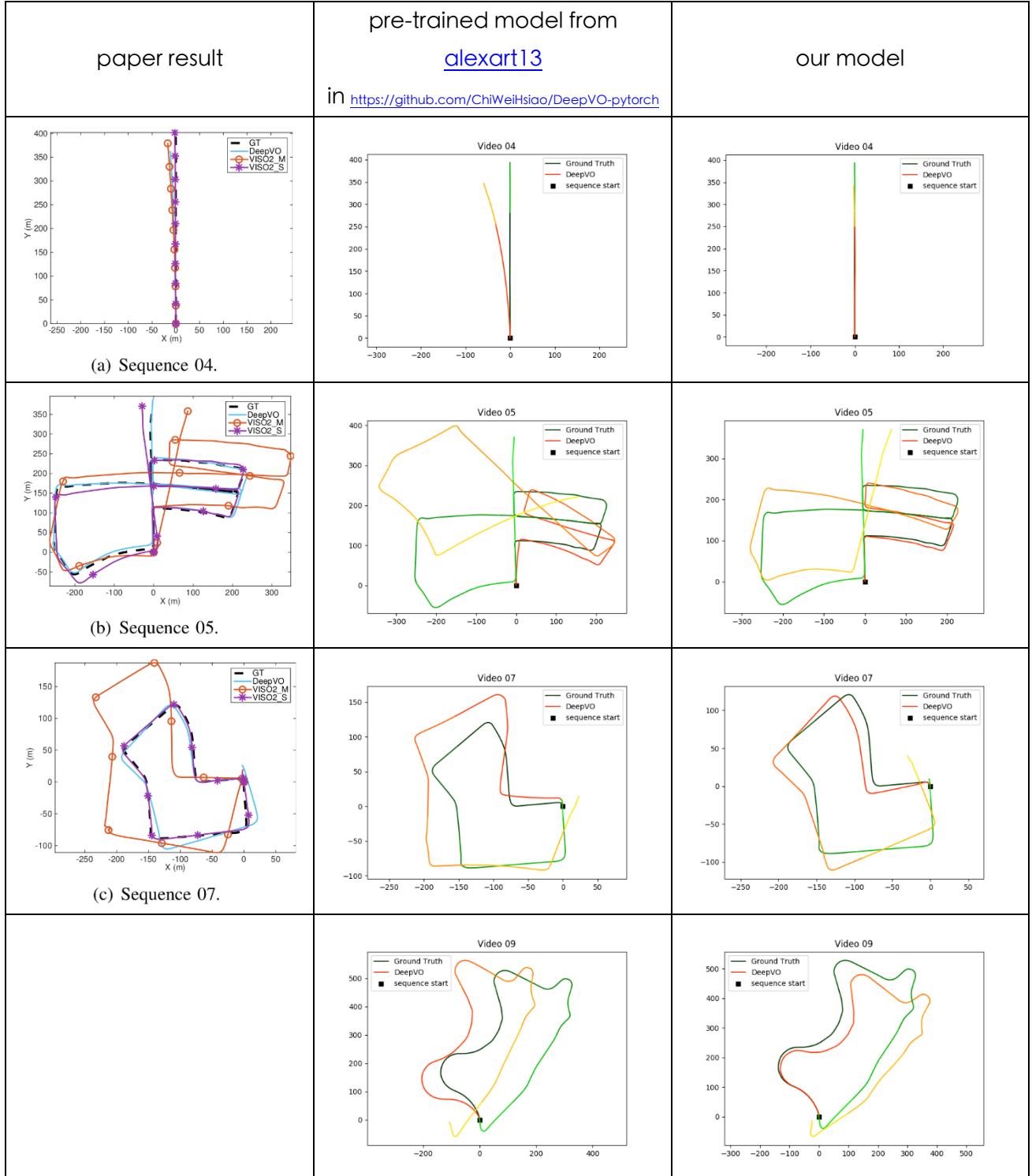
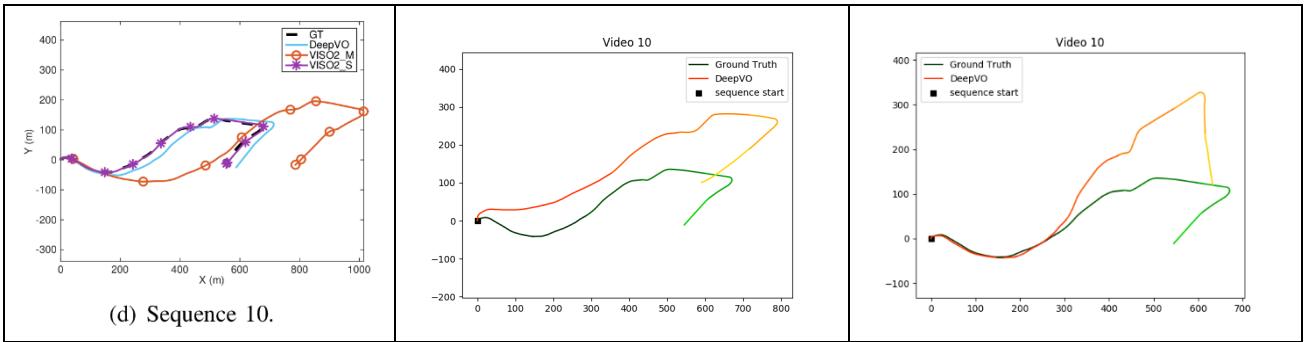


Figure 10 Blender 參數設定

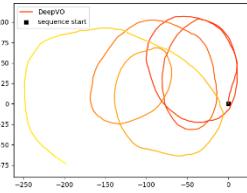
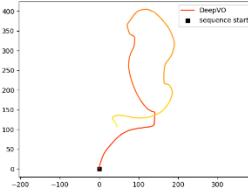
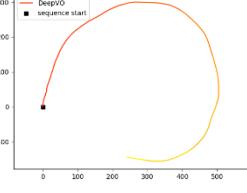
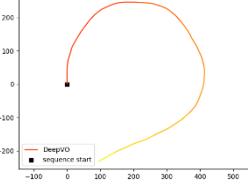
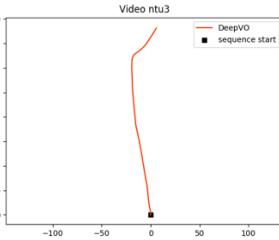
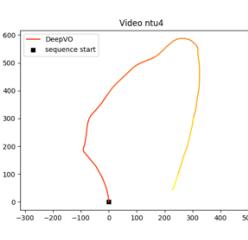
IX. Result

Test on KITTI dataset





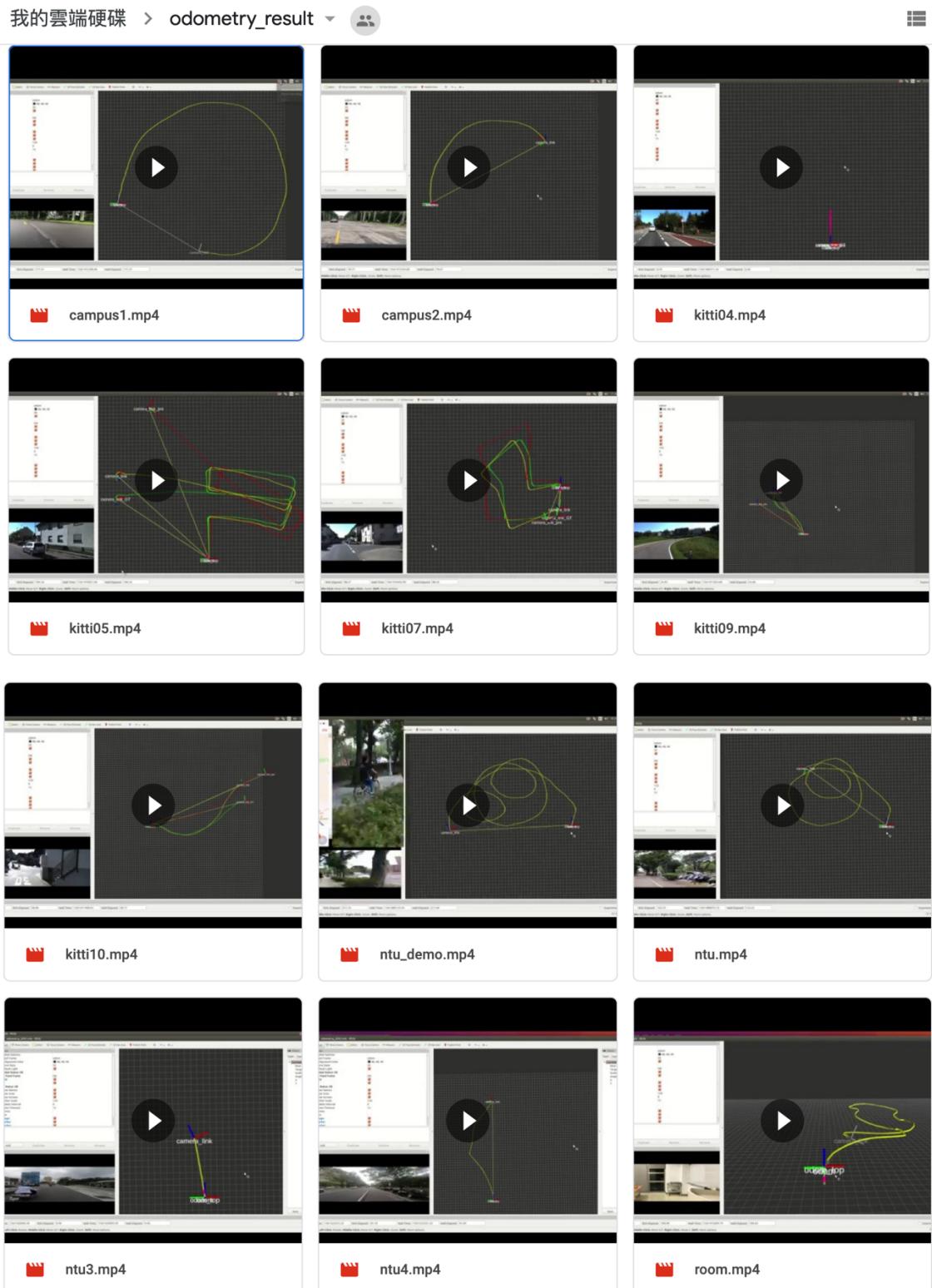
Test on our dataset

ntu	ntu GT	room	room GT
			
campus1	campus1 GT	campus2	campus2 GT
			
ntu3	ntu3 GT	ntu4	ntu4 GT
			

Visualize on those test result by our visualizer

Link to demo videos: <https://drive.google.com/open?id=16Mqq-QOYdFPORCvmaqvTxSjoXwKzzQ8O>

由於為 linux 螢幕錄影，部分影片須用 VLC player 才能播（三角錐）



X. Conclusion

- Deep VO 對於影像品質非常敏感，因為其 train on 一個蠻完美的開車街景 dataset(KITTI)，所以對於手持騎車或走路，預測結果均不如預期，需要有行車記錄器那種很穩定的行車影像。
- 承上點，不過這可能僅僅是因為這篇是第一篇，後續這個領域還有許多論文產出，甚至有 Unsupervised 的方法出現，或是有融入 IMU data 一起 training 的，算是還在研究中的新方向。
- 承上點，以 robustness 而言，目前還是使用傳統的幾個經典方法如 ORB2-SLAM 或 RTabmap 等等效益會更大。

XI. What do we implement by ourselves

- 訓練時的 Script 都是自行撰寫，過程有一些部分參考範例 Code。
- 意即 training 時的 Dataloader、Model 的建構皆是自行撰寫，並且也自己從頭 train。
- 三維度的 Visualization 是自行撰寫，需要讀取 Predict 的 txt file。
- 自行錄製影片並且用 Blender 切割成 dataset。
- 用自製的 dataset 做預測，並且與參考 code 做比較。

XII. What do we not implement by ourselves

- downloader.sh、myVisualize.py、helper.py、test.py 是直接複製網路上的並做一點微調而已。
- 獲得 dataset 部分，就是直接使用 downloader.sh。
- Helper.py 內部有一些轉換的函式，像是 Euler angle 轉換我們是直接使用。
- myVisualize.py 是他的二維繪圖，我們直接使用方便與三維的做比較。
- test.py 使用的原因在於 predict 的部分，由於會有坐標系和 Dataloader 的問題，所以我們是使用自己訓練好的 Model 搭配他的 dataloader 以及對 Predict 的後處理函式來撰寫真正 Predict 的結果。

XIII. 作業心得

本 Project 實作了 DeepVo 這個 Model，而我認為最困難的點就是在於座標轉換的地方，然後還有 rnn 的 sequence to sequence training。至於 model 的撰寫以及 dataloader 的部分 implement 不難，但是也因為在 predict 的時候，貌似需要 batch>1 才能做 predict，所以我們的 loader 沒辦法做 predict，但礙於時間不夠只好在最後直接使用網路的 loader 來做讀檔進行 Testing。

而 visualizer 上因為原本已經有多 ROS 的開發經驗所以蠻知道該如何處理的，有別於一般演算法可以直接使用 rosbag real-time 的一邊發 data 一邊進行 odometry 的運算與 output，我們需要讓我們 prediction 的結果和 raw data 的 image 自己進行同步的顯示。

而自己製作 dataset 的部份更是有了一些意外的收穫，除了採集過程很冗長、困難外，也認知到 DeepVO 在早期時其實還真的只是個雛形，並沒有想像中這麼厲害，不及傳統經過精密設計的 visual odometry 那麼 robust 和實用。

對於日後的研究而言，我想我還是會走傳統的方法，並 fusion 不同的 sensor(2d laser scanner, IMU)來做出一個好的 odometry。

XIV. How to Run the Program

1. 詳情請看 git hub 上的 README.md，有較完整的解說，下面為大致流程簡述。
2. 執行 getdataset.sh 獲得 KITTI 的資料。
3. 執行 Preprocess 來對資料作前處理。
4. 執行 myMain.py 來做訓練。
5. 執行 myTest.py 來做預測。
6. 執行 myVisualize.py 來做 2D 預測的繪圖。

XV. Reference

- [1] S. Wang, R. Clark, H. Wen and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 2043-2050.
- [2] reference: <https://github.com/ChiWeiHsiao/DeepVO-pytorch>
- [3] visualizer odometry: <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>
- [4] visualizer image: http://wiki.ros.org/image_transport/Tutorials/PublishingImages
- [5] visualizer marker: <http://wiki.ros.org/rviz/Tutorials/Markers%3A%20Basic%20Shapes>