中原大學
資訊工程學系
專題實驗報告

基於影像之 ROS 機器人室內巡檢辨識系統
Image based on ROS robot indoor inspection recognition system.

指導老師:余執彰教授

專題生:

10727102 康愷辰

10727207 張　婷

10727249 林恆毅

中華民國 110 年 10 月

目錄

一、摘要

現今 AI 人工智慧技術的蓬勃發展,機器人技術在近年來已有大幅提升，並開始出現各式應用，人工智慧對於機器人領域有著密不可分的關西，尤其在對理解像機器學習、機器視覺、影像處理等，AI 人工智慧的研究中，使機器人應用在眾多領域。

本專題主旨為在 Linux 系統下，基於機器人作業系統(Robot Operating System ROS)之室內巡檢之開發，透過 SLAM (Simultaneous Localization and Mapping)技術結合里程計(odometry)與雷射掃描數據(scan)建構周圍環境地圖。使用 AMCL(Adaptive Monte Carlo Localization)技術在建構完成的地圖上進行車體定位，使用者下達目標位置後，機器人透過 DWA 演算法計算最佳路徑，到達目標位置時進行物件辨識->偵測目標物->回報目標物->完成任務 ; 展現出人工智慧與機器人的整合技術。

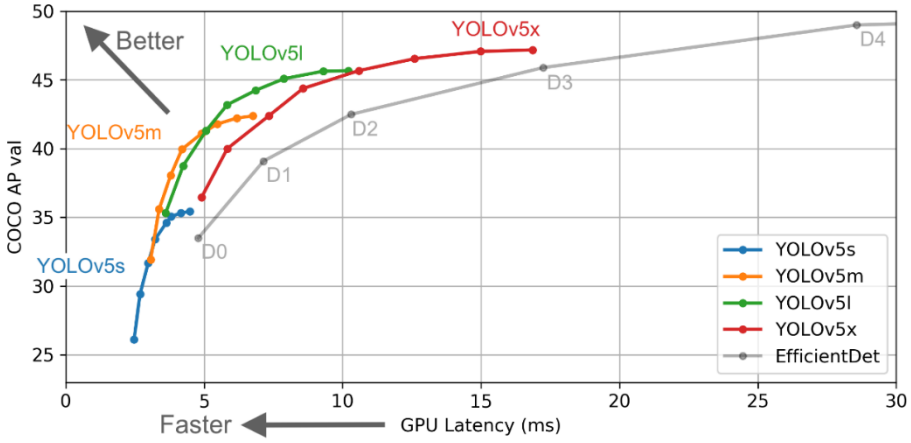
二、研究動機

近幾年來機器人已被應用在各個領域，能夠取代人們在生活及工作中的需求，以更節省資源、更有效率的方式完成任務。現今我們在機場裡使用的安檢掃描器都是 X 光機，就像拍 X 光片一樣，它只能提供平面的二維圖像，而且偶爾 X 射線會被金屬物品擋住，造成視野盲區。無法確定是否有可疑物品從而通過安全檢查，因此我們就在想，如果這些行李被放置在一些機場人煙稀少或是無人的環境中，難免會覺得可疑，於是我們結合 RO 官方網站提供 Gmapping 的程式包能夠拿來做 SLAM (Simultaneous Localization and Mapping)演算法， 使機器人能在缺乏外界提供資訊的環境中完成定位、自行建立地圖無須仰賴 GPS 定位系統。而為了辨識行李箱、後背包、手提包等物品我們將攝像頭加入我們巡檢的系統內以此輔助系統辨識。

三、研究方法與結果

3.1. 模型架構介紹

YOLOv5 為高效、易於使用的目標檢測框架，根據圖(一)YOLOv5 是在 PyTorch 中實現使得在支援方面更加簡單與部署時更加容易，YOLOv5 共有 Yolov5s、Yolov5m、Yolov5l、Yolov5x 四種模型，可由表(一)得知各個模型的.yaml 檔所規範的 depth multiple 值與 width multiple 值其中 YOLOv5s 是深度最淺、寬度最小的模型，剩餘三種皆是在此基礎上不斷加深與加寬，API 精度值也因此而提升(由表(二)得知)，卻也使速度的消耗不斷的增加，而我們為了輕量化訓練過程以及加快辨識速度，我們選擇使用檢測速度最快的 Yolov5s。



圖(一)、各模型對 EfficientDet 比較

表(一)、各模型 yaml 檔 depth multiple、width multiple 值

模型	depth_multiple	width_multiple
Yolov5s.yaml	0.33	0.50
Yolov5m.yaml	0.67	0.75
Yolov5l.yaml	1.00	1.00
Yolov5x.yaml	1.33	1.25

表(二)、各模型在 mAP 與速度方面的表現

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{test} 0.5:0.95	mAP^{val} 0.5	Speed V100 (ms)		params (M)	FLOPs 640 (B)
YOLOv5s	640	36.7	36.7	55.4	**2.0**		7.3	17.0
YOLOv5m	640	44.5	44.5	63.1	2.7		21.4	51.3
YOLOv5l	640	48.2	48.2	66.9	3.8		47.0	115.4
YOLOv5x	640	**50.4**	**50.4**	**68.8**	6.1		87.7	218.8

3.1.1. 資料集準備

數據集選擇與提取:

COCO Dataset 為豐富的物體檢測數據集，並且圖片主要從複雜的日常場景中擷取，加上資料提供的 80 個類別中有超過 50 萬個目標標注，適合讓我們進行其他處理，也能讓生成結果符合現實中的生活場景，而我們下載 COCO 2017 數據集，從 COCO Dataset 中提取所需的類別資料，提取類別為 **backpack**、**handbag** 及 **suitcase**，在經過預處理後圖片數量為 11989 張。

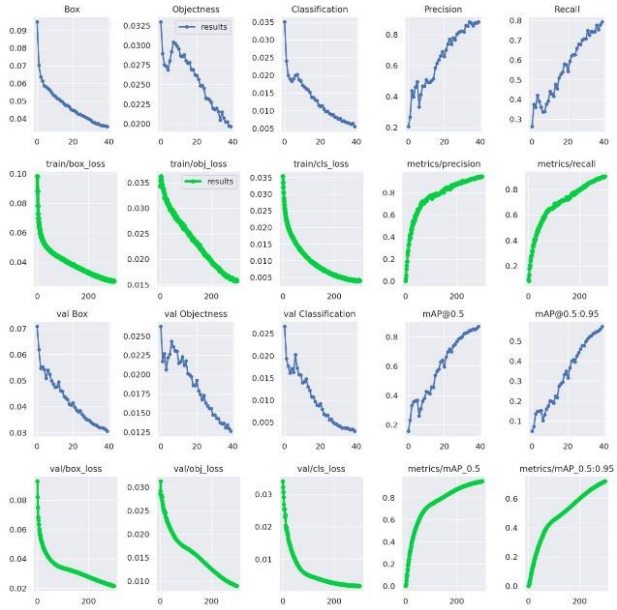
資料格式轉換:

原先提取類別後的標籤檔案為.xml 檔，而 YOLOv5 所需要的格式為一個圖片檔案對應一個同名的.txt 標籤檔案，檔案中每一行表示為一個目標，格式為：class、x_center、y_center、box_width、

box_height，我們利用 python 檔進行格式的轉換以符合 YOLOv5 的格式要求。

3.1.2. 訓練模型

我們使用環境 Docker 進行環境配置修改.yaml 檔以及參數配置後即可運行 train.py 開始訓練，一開始是使用參數值 batch 16、epoch 40 進行訓練，將訓練後的結果圖示與現在所使用的模型(batch 16、epoch 300)之訓練結果圖(二)相比：由圖(二)可得知參數 epoch 40 訓練成果較為不佳，各個數據變化圖的波動與 epoch 300 相比起伏太大，收斂的部分也沒有預期內的好，於是我們使用的參數值為 batch size 16、epoch 300 進行模型的訓練使得訓練成果提高。



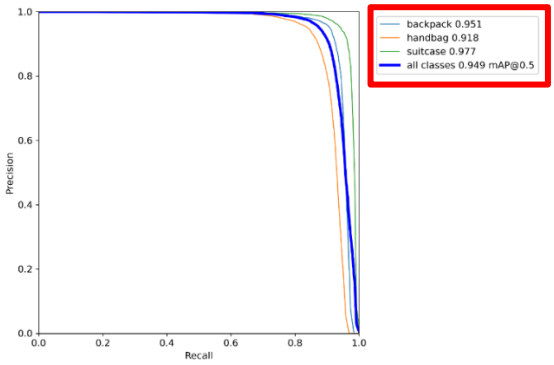
圖(二)、兩者對比結果(藍色為 epoch40，綠色為 epoch300)

表(三)、比較 epoch40 與 300 的辨識結果

結果 / 模型層數	precision	recall	mAP_0.5	mAP_0.5:0.95
epoch 40	0.8813	0.7932	0.8704	0.5728
epoch 300	0.94463	0.90162	0.94869	0.71896

mAP_0.5 與 mAP_0.5:0.95：

AP(Average Precision)為計算 Precision-recall curve 底下的面積(圖(三)所示)，m 表示平均，因此 mAP 為各類別 AP 的平均值。



圖(三)、使用 yolov5 框架訓練之模型的 Precision-recall curve 圖

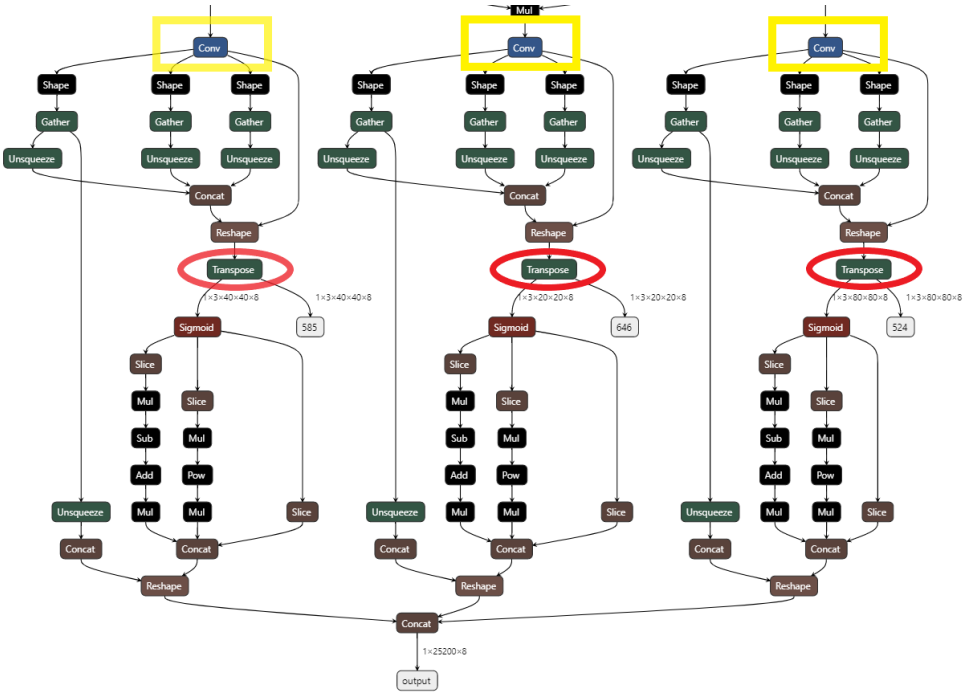
由圖(三)可看出各個結果曲線可看出訓練後的模型取得了不錯的結果，大部分之曲線隨著 y 軸之 epoch 數的增加都趨於收斂，且變化的波動起伏都不大。

3.1.3. 模型轉換

我們將已訓練好的模型透過 OpenVINO 提供的 Model Optimizer 來轉換 Intermediate Representation(IR)的優化模型，其中 IR 檔案是由兩個文件檔所組成，一個是 XML 文件包含了網絡架構，另一個是 bin 文件存放了權重和誤差，有了這兩個文件檔才能透過 Inference Engineer 和 User Application 進行互動。

圖(四)、Intel OpenVINO 模型轉換流程圖

首先我們使用 YOLOv5 官方所提供的 export.py 將訓練好的模型從 PyTorch 轉為 ONNX，接著經由 Netron 查看模型的完整結構並尋找 “Transpose”找到三個輸出節點便可往上查找各自的卷積節點，這三個卷積節點將用於轉換為 IR 檔案時之參數。



圖(五)、部分模型結構圖(紅色橢圓為輸出節點，黃色方框為卷積節點)

接下來我們透過 OpenVINO 所提供的 mo.py 來運行 Model Optimizer 將已轉換為 ONNX 的模型再轉換為 IR 檔，轉換成功後我們便可使用 OpenVINO 工具套件進行推理。

```
[ SUCCESS ] Generated IR version 10 model.
[ SUCCESS ] XML file: /home/tt/Documents/test/best.xml
[ SUCCESS ] BIN file: /home/tt/Documents/test/best.bin
[ SUCCESS ] Total execution time: 25.33 seconds.
[ SUCCESS ] Memory consumed: 369 MB.
It's been a while, check for a new version of Intel(R) Distribution of OpenVINO(TM) toolkit here https://
software.intel.com/content/www/us/en/develop/tools/openvino-toolkit/download.html?cid=other&source=prod&c
ampid=ww_2021_bu_IOTG_OpenVINO-2021-3&content=upg_all&medium=organic or on the GitHub*
```

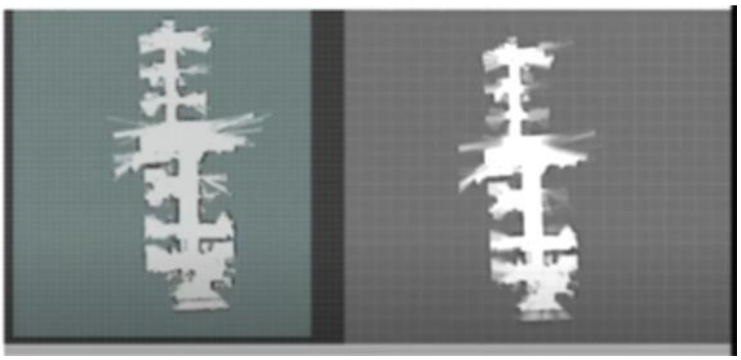
圖(六)、模型轉換為 IR 檔之成功截圖

3.2. 建構地圖

Gmapping 是一種基於 Rao-Blackwellized 粒子濾波器(Particle Filter)的 SLAM 技術，原理為利用粒子的分布去推測機器人可能的移動路徑。而 ROS 的官方網站上已有提供 Gmapping 的 package。在建構地圖時， Gmapping 會將掃描到地圖輸出到/map，我們透過 rviz 把地圖以黑白的方式呈現。白色區域代表空曠區域，黑色則代表障礙物或牆壁，灰色則代表尚未探索的區域，而 RBPF(Rao-Blackwellized Particle Filter)算法利用公式(1)對聯合概率密度函數進行因式分解。

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t}) = p(m \mid x_{1:t}, z_{1:t}) p(x_{1:t} \mid z_{1:t}, u_{0:t}) \quad (1)$$

因此 RBPF 可以先估計機器人的軌跡而後再去根據已知的軌跡計算地圖。



圖(七)、rviz 地圖上形狀

3.3. 自主定位系統

在這個項目中，我們使用 AMCL（自適應蒙特卡羅定位）包在模擬環境中定位機器人。由 amcl 節點提供/global_localization (std_srvs/Empty)服務，所有粒子隨機散佈在地圖的自由空間中。在此之後，機器人然後通過將 Twist()數據發佈到/cmd_val 主題來執行旋轉收斂粒子運動。完成運動後，我們會收集粒子的 convariance 第 0 陣列、第 7 陣列與第 35 陣列將這 3 個值除以 3 得到我們的協方差值:

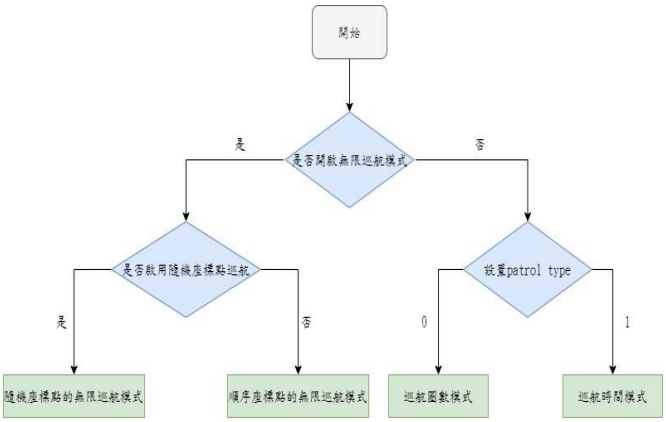
$$convariance = \frac{initial\_cov\_xx + initial\_cov\_yy + initial\_cov\_aa}{3}$$

如果該協方差小於 0.05，則表示機器人已正確定位自身。然後，程序將結束。如果這個協方差大於 0.05，它將重複整個過程（分散粒子，執行運動，檢查協方差.....）。

3.4. 巡檢系統

3.4.1. 設置目標點

在巡檢系統中，我們主要的巡邏方式有分定點巡航以及隨機巡航，定點巡航我們也可以設置巡航的圈數.即按照順序將所有目標點巡邏一遍就是走了一圈。那對於目標點的取得我們利用 rviz 中的 2D NAV GOAL 功能來獲取地圖上的 position 與 orientation 再利用/move_base_simple/goal 這個 topic 去查詢各坐標點的位姿信息再將訊息寫入我們的巡檢系統完成設置目標點。



圖(八)、巡檢流程

3.4.2. 巡檢過程

我們去訂閱 /move_base 這個 action server，move_base 在導航過程中會給予我們狀態反饋，根據反饋的狀態我們來決定何時發送下一個目標點的坐標。再來就是判斷巡邏方式、開始任務 那我們在巡檢過程中有可能找不到目標點所以我們設置若到達目標點時超時了(約 5 分鐘)，那麼就取消這個目標點的導航任務，準備開始前往下一個目標點。

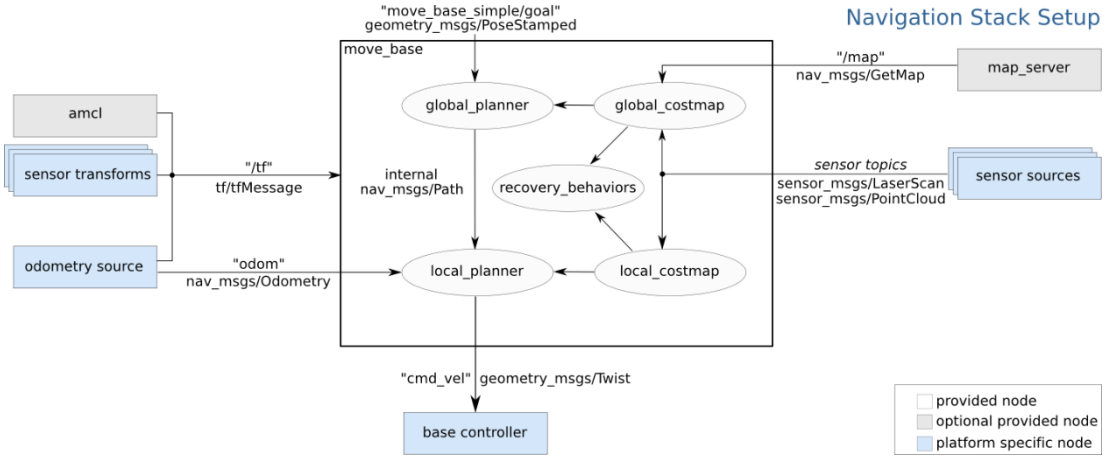
最後我們獲取 move_base 導航時的任務狀態，如果反饋的狀態是 SUCCEEDED，那麼就認為本次導航任務成功。

3.4.3. 辨識物件

當機器人走到目標點時我們才會開啟物件辨識的 node，辨識的節點首先會去訂閱攝像頭的資訊並且接收 image_raw，我們將收到的照片經過 cv_bridge()轉成 opencv 格式，然後開啟模型並且讀入照片，那因為照片的實際大小太大所以我們透過一些預處理將格式縮小成 480*640，之後我們將 yolo 模型所辨識到信心水準 0.6 以上的物件框出來→截圖→儲存資料夾→結束節點。

3.5. 機器人避障

move_base 提供了 ROS 導航的配置，主要分成 4 個部分，global planner: global_planner 會根據你 gmapping 的地圖資訊和障礙物，設立的起始點和終點去做 DWA/TEB 的路徑規劃，local planner: 根據附近的障礙物進行閃避路徑的規劃簡單來說就是 global_planner 原先設立的路徑進行修正而 Navigation 使用兩個 Costmap 來存儲有關障礙物的信息。Global Costmap:在整個環境中紀錄已知的障礙物；local Costmap 為即時的偵測障礙物規劃和避障。



圖(九)、ros navigation 架構圖

四、結論

目前我們已經完成將定位、自主巡檢及物件辨識整合在一起，在建構地圖上我們並沒有採用自主建構地圖而是採用手動去建構一來我們可以省去機器人自主建構會花費較長時間二來考慮到機器人電量問題，所以我們採用手動建構。在定位系統中，我們在環境較為複雜的區域有蠻好的效果，但在比較單調或空間相似度較高的環境中容易定位失敗，所以可能會多定位幾次，若定位成功後機器人在自主巡檢的過程中，能準確地走到目標點，物體辨識部分雖然我們平均才 4、5 偵數不過卻不影響我們在機器人定點時的檢測，如果有物件的話大概 10 秒內能辨識出來，所以整合的效果像是相當成功。

五、未來展望

針對本專題尚可發展及精進的部分，我們列了幾項層面:

1. 制定使用者介面這樣就不需要在 rviz 上找尋目標點及透過 topic 查看訊息花費太多時間。
2. 優化 yolo 模型，使我們的偵數提高至 15 偵避免畫面 delay。
3. 在定位過程中可以手動設置特徵點在地圖上以便機器人去定位現在位置。
4. 節點與節點的訊息傳遞過程將 topic 相互發布訂閱模式改成 service 或 action 方便我們管理。

六、參考文獻

[1]Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics, Volume 23, pages 34-46, 2007

取自 http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf

[2]定位系統。

取自 https://github.com/umerjamil16/ROS-Localisation-using-AMCL

[3]yolov5 四種模型架構。

取自 https://github.com/ultralytics/yolov5

[4]coco 資料集。

取自 https://cocodataset.org/#home。

[5]yolov5 評估指標解釋。

取自 https://towardsdatascience.com/mask-detection-using-yolov5-ae40979227a6。

[6] 模型檢測結果圖片。

取自 https://pixabay.com/zh/photos/luggage-suitcases-baggage-bags-933487/。

[7] OpenVINO 模型轉換流程。

取自

https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

[8] 自主巡檢系統。

取自 https://www.corvin.cn/892.html

[9] ros 功能包。

取自 http://wiki.ros.org/ROS/Tutorials