

國立陽明交通大學

影像與生醫光電研究所

碩士論文

Institute of Imaging and Biomedical Photonics

National Yang Ming Chiao Tung University

Master Thesis

深度確定性策略梯度於田間履帶機器人之追跡及避障

Path Following and Obstacle Avoidance of Field Tracked

Robots by Deep Deterministic Policy Gradient

研究生：李仁祐 (Jen-Yu Lee)

指導教授：陳顯禎 (Shean-Jen Chen)

中華民國一一〇年六月

June 2021

深度確定性策略梯度於田間履帶機器人之追跡及避障
Path Following and Obstacle Avoidance of Field Tracked
Robots by Deep Deterministic Policy Gradient

研究 生：李仁祐

Student: Jen-Yu Lee

指導教授：陳顯禎 教授

Advisor: Prof. Shean-Jen Chen

國立陽明交通大學

影像與生醫光電研究所

碩士論文

A Thesis
Submitted to Institute of Imaging and Biomedical Photonics
College of Photonics
National Yang Ming Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Institute of Imaging and Biomedical Photonics

June 2021
Taiwan, Republic of China

中華民國 一一〇年六月

致謝

◦

深度確定性策略梯度於田間履帶機器人之追跡及避障

研究生：李仁祐

指導教授：陳顯禎 教授

國立陽明交通大學 影像與生醫光電碩士班

摘要

本論文以視覺化即時定位與地圖構建(visual simultaneous localization and mapping, vSLAM)以及 YOLO 物件辨識等技術為基礎，利用 MATLAB Simulink 之深度強化學習(deep reinforcement learning, deep RL)中的無模型(model-free approach)深度確定性策略梯度(deep deterministic policy gradient, DDPG)為學習控制架構，其擅長處理非線性時變控制問題以及具有連續動作空間決策優勢。在機器人作業系統(robot operating system, ROS)框架下，以智能體(intelligent agent)強化學習環境系統(environment)出最佳化控制器(actor)後，驅使機器人依目標函數來行走，在這裡主要是動態決策新的避障局部路徑來跟蹤，讓機器人能迅速避障後回到原有的全局路徑。

我們先於 Gazebo 模擬環境中去驗證所學習訓練出的 DDPG 智能體擁
有「能夠在執行任務上具有高成功率」的前提條件後，將其引入實驗室所開
發的履帶機器人於田間場域中進行實測實驗。此履帶機器人以 Intel
RealSense D435i 深度相機用於 vSLAM 中的定位感測，以及結合 YOLO v3
進行障礙物感知定位的輸入，藉由訓練完成 DDPG 智能體來提供最佳化
actor。目前在 Gazebo 模擬中可完成 80% 成功率之避障追跡，而在履帶機器

人於田間場域之避障追跡實作中，我們以 RTAB-Map (real-time appearance-based mapping)為 vSLAM 演算法，成功率已達到 60%以上。

關鍵字：vSLAM、YOLO、田間履帶機器人、Simulink、深度確定性策略梯度、避障。

Path Following and Obstacle Avoidance of Field Tracked Robots by
Deep Deterministic Policy Gradient

Student: Jen-Yu Lee Advisor: Prof. Shean-Jen Chen
Institute of Imaging and Biomedical Photonics
National Yang Ming Chiao Tung University

Abstract

In this thesis, a deep deterministic policy gradient (DDPG) learning architecture in MATLAB Simulink is utilized to train intelligent agents. The DDPG is a kind model-free approach of deep reinforcement learning (deep RL). Based on the condition that the mature positioning technology and object detection we already have by using vSLAM (visual simultaneous localization and mapping) and YOLO (you only look once), the advantage of good at making decision and dealing with nonlinear time variant control problem of DDPG is used to train an agent as the system controller for controlling the robot to achieve the expected behaviors we want in the robot operating system (ROS) framework. The thing that the trained agent is mainly used to make decision dynamically for the new local path is to let the robot which is driven by agent can be returned rapidly to the reference path after avoiding the obstacle optimally.

Before doing the experiment of obstacle avoidance control on the field tracked robot we developed which will be driven by DDPG trained agent in field, we go first on doing the verification with the trained agent at the simulated environment called Gazebo, make sure of that the success rate of executing task by agent is certainly high. And the sensing of the tracked robot is using Intel RealSense depth camera D435i as both inputs of vSLAM for doing robot localization and YOLO v3 for doing obstacle sensing. Currently, we get 80% success rate in verification of obstacle avoidance control under Gazebo, and 60%

in verification of obstacle avoidance control on the field tracked robot in field by using RTAB-Map (real-time appearance-based mapping) as vSLAM.

Keywords: vSLAM, YOLO, field tracked robot, Simulink, DDPG, obstacle avoidance.

目錄

摘要	i
Abstract	iii
目錄	v
圖目錄	vii
表目錄	x
第一章 序論	1
1-1 前言	1
1-2 研究動機	2
1-3 論文架構	4
第二章 田間履帶機器人及物件辨識定位	5
2-1 ROS 架構下履帶機器人	5
2-1-1 ROS	6
2-1-2 履帶機器人 ROS 架構	7
2-2 SLAM	8
2-2-1 深度相機	8
2-2-2 vSLAM	9
2-3 YOLO 辨識	9
2-3-1 YOLO	9
2-3-2 ROS 感測結合流程	10
第三章 深度確定性策略梯度之理論與模擬	12
3-1 深度強化學習	12

3-1-1 馬可夫決策過程概述	12
3-1-2 基於價值函數算法	13
3-1-3 基於策略函數算法	15
3-2 深度確定性策略梯度之演算法.....	17
3-3 Simulink 控制與學習架構.....	20
3-3-1 Simulink 控制架構.....	21
3-3-2 MATLAB DDPG 架構	22
3-4 TurtleBot3 於 Gazebo 環境模擬	22
第四章 實驗討論與結果	25
4-1 履帶機器人追點控制.....	26
4-1-1 模擬測試	32
4-1-2 實作結果	34
4-2 履帶機器人追跡避障.....	36
4-2-1 模擬測試	43
4-2-2 實作結果	45
第五章 結論與未來工作	49
參考文獻	51

圖目錄

圖 2-1 履帶機器人。	5
圖 2-2 履帶機器人 ROS 架構。	7
圖 2-3 vSLAM 架構[7]。	9
圖 2-4 YOLO-v3-tiny 網路架構於 ubuntu 16.04。	10
圖 2-5 rqt 節點監控 UI：(a) YOLO 物件偵測收發流程；(b)感測結合收發流 程。	11
圖 3-1 MDPs 中智能體與環境交互過程圖[10]。	13
圖 3-2 DQN 演算流程[12]。	14
圖 3-3 DDPG 演算流程[15]。	18
圖 3-4 DDPG 架構圖。	19
圖 3-5 MATLAB 與機器人 ROS 通訊方法。	21
圖 3-6 Simulink 控制架構。	22
圖 3-7 rqt 節點監控 UI。	22
圖 3-8 DDPG 網路架構：(a) actor 網路架構；(b) critic 網路架構[17]。	23
圖 3-9 TurtleBot3 Waffle [18]。	23
圖 3-10 Gazebo 環境運行 YOLO 物件偵測。	24
圖 4-1 履帶機器人運動學模型(kinematic model) [21]。	25
圖 4-2 追點控制 Gazebo 訓練環境。	26

圖 4-3 追點控制第一版訓練統計。	27
圖 4-4 時間懲罰函數。	28
圖 4-5 追點控制第二版訓練統計。	29
圖 4-6 追點控制第三版訓練統計。	30
圖 4-7 追點控制第四版訓練統計。	31
圖 4-8 追點控制第四版獎懲值累積情形。	32
圖 4-9 純追點模擬驗證中機器人從起點走向終點。	32
圖 4-10 純追點模擬驗證中機器人每時刻的狀態。	33
圖 4-11 純追點模擬驗證中機器人每時刻的獎懲值。	33
圖 4-12 純追點模擬驗證中機器人每時刻的終止條件狀態。	34
圖 4-13 室內純追點實作驗證中每時刻：(a)計算出的線速度及角速度；(b)智能體輸出的動作空間；(c)智能體觀察到的狀態空間。	35
圖 4-14 室外純追點實作驗證中每時刻：(a)計算出的線速度及角速度；(b)智能體輸出的動作空間；(c)智能體觀察到的狀態空間。	36
圖 4-15 追點及避障控制 Gazebo 訓練環境。	37
圖 4-16 追點及避障控制第一版訓練統計。	39
圖 4-17 追點及避障控制第二版訓練統計。	40
圖 4-18 追點及避障控制第三版訓練統計。	41
圖 4-19 追點及避障控制第四版訓練統計。	42

圖 4-20 追點及避障模擬驗證中機器人避開障礙物從起點走向終點。	43
圖 4-21 成功的追點及避障模擬驗證中每時刻：(a)智能體觀察到的狀態； (b) 智能體獲得的獎懲值；(c)終止條件狀態。	44
圖 4-22 追點及避障模擬驗證中機器人避障失敗。	44
圖 4-23 失敗的追點及避障模擬驗證中每時刻：(a)智能體觀察到的狀態； (b)智能體獲得的獎懲值；(c)終止條件狀態。	45
圖 4-24 田間場域驗證布置。	45
圖 4-25 田間障礙物。	46
圖 4-26 失敗的 ORB-SLAM 追點及避障實作中智能體每時刻：(a)觀察到的 狀態空間；(b)輸出的動作空間；(c)獲得的獎懲值。	46
圖 4-27 成功的 RTAB-MAP 追點及避障實作中智能體每時刻：(a)觀察到的 狀態空間；(b)終止條件狀態；(c)獲得的獎懲值(zoom in)。	47
圖 4-28 失敗的 RTAB-MAP 追點及避障實作中智能體每時刻：(a)觀察到的 狀態空間；(b)終止條件狀態；(c)獲得的獎懲值。	48

表目錄

表 2-1 深度相機規格。	8
---------------	---

第一章 序論

1-1 前言

農用機器人在農業的應用可提高農作產量、減少人力資源需求，並且使農務成為高科技職業，進而促進農業經濟發展，可藉由無人載具來搭配農業自動化設備來大幅降低農業勞動成本，幫助緩解農業勞動人口的缺口與提升整體蔬果的經濟產量。台灣目前有相當多的單位投入無人空中載具的農業應用發展；相對於無人空中載具，無人地面載具(unmanned ground vehicle, UGV)所需的技術門檻更高，其重要性更勝於無人空中載具。我們實驗室近幾年來致力於田間機器人(field robot)開發，結合人工智慧的趨勢與光電的技術，開發出能夠在田間自主巡航，且執行多種任務的田間機器人，如疏花、去新梢及除蟲等，希望能取代傳統農業上耗時且耗人力的工作，發展新農業。

而如何替複雜的機器人設計一個良好的控制器是極具挑戰的問題，尤其在現實中的控制模態往往為非線性，且機構不精準的情況下，怎麼設計一個智慧且強健的控制器來克服來自自身系統或外來環境的干擾，傳統上常以時變線性化的控制模型來處理，而在本論文中，我們使用非傳統的強化學習(reinforcement learning, RL)方式來設計控制器(controller)，RL 學習過程類似以適應性最佳控制(adaptive optimal control)方式來處理非線性動力系統(nonlinear dynamical system)問題。本研究基於已掌握之視覺化即時定位與地圖構建(visual simultaneous localization and mapping, vSLAM)技術用於實驗

室自行開發的履帶機器人，於定位方面使用兩種方式進行，分別為使用 ORB (oriented FAST and rotated BRIEF)-SLAM 的特徵點擷取方法，以及 RTAB-MAP (real-time appearance-based mapping) [1]在實驗上做比較，而為了動態避障的功能，省略了建圖(mapping)的過程。另外使用 YOLO 進行障礙物的偵測，當偵測到障礙物時結合深度相機，將其轉換為與障礙物與車體之間的距離及夾角，作為狀態空間中的狀態。以深度強化學習(deep reinforcement learning，deep RL)之無模型(model-free)深度確定性策略梯度(deep deterministic policy gradient，DDPG)用於田間履帶機器人的控制器設計，透過獎懲函數(reward function)來引導智能體(intelligent agent)的 critic 去調整最佳的 actor，使其在後續中能驅使機器人去達到有最大的獎懲值的目標狀態，目的是為了讓智能體學習出控制著機器人朝著「符合獎懲函數理念的參考目標狀態逼近」的行為，激勵智能體權衡出探索(exploration)和利用(exploitation)的能力。

1-2 研究動機

近二十年來 UGV 已快速發展，在市面上也漸漸有了供需，範圍遍布各行各業，不論是國內外汽車大廠、農機企業或是機器人研發團隊。也隨著硬體設備的發展與時俱進，有了近年非常流行的機器學習，在神經網路中具有模仿生物神經元活躍與非活躍狀態的函數如 sigmoid 和線性整流函數(rectified linear unit，ReLU)等激活函數[2]，能夠將彼此間關聯不大的特徵以

線性化的函數輸出，故神經網路層又被稱為函數近似算法(function approximation algorithms)，是一種端到端(end-to-end)的方式，不需要借助任何第三方程序，可以直接將輸入映射(mapping)至輸出，而 RL 結合了深度神經網路，即產生 deep RL [3]，deep RL 運用深度神經網路可以作為控制器或系統模型，能夠將任何輸入以線性化的方式描述[4]。

基於上述趨勢，一架 UGV 要於嚴峻的環境中執行任務，例如在田間自動駕駛，甚至是進行考慮到時間的軌跡追蹤(trajectory tracking)，儘管在感測器精度可控且誤差小的情況下，都必須透過準確的系統模型來描述當前非線性且時變系統與環境狀態。這時 deep RL 就非常適合解決非線性控制問題[5]，它可以直接做為控制器，不用替系統與環境進行建模就能對難以數學化的系統進行最佳化控制；也可以作為系統與環境模型將控制器輸出通過神經網路提取特徵後，經過激活函數近似成非線性函數來估計出系統狀態，再由其他控制器做為決策機制，選出適當的動作空間。但在學習中，必須讓智能體操控著機器人在環境中不論成功或失敗地不斷嘗試，這種類似於取樣的過程，使得智能體能夠累積足夠多「在某狀態時，採取某個動作後所得到的獎懲」的經驗來當作訓練資料，可以建立起控制器，或者是環境與系統模型。而在 DDPG 中擁有四個神經網路，兩個用來穩定學習，另外兩個分別是善於處理連續動作空間的決策網路 actor，以及分析當前狀態空間後，針對決策網路做出決策後予以評分的評論網路 critic。在學習的過程中

actor 的決策會被 critic 的評價所影響，彼此間相互影響直到 actor 能在任務中穩定得到足夠高的獎懲值後即完成學習。在狀態空間的設置考量上，可同時考慮多種受控狀態問題。如本論文中，將全局路徑規劃與避障(obstacle avoidance)的局部路徑規劃中的狀態一併考慮，可以避免控制器在分批設計上可能會產生的衝突，但也可能會造成整體效率不彰。當然 DDPG 主要著重在動態決策新的避障局部路徑，讓機器人能最佳化避障後快速回到全局路徑。

1-3 論文架構

第一章介紹 RL 於機器人控制中的優勢，以及本研究的動機和目的；第二章介紹本研究中所使用的履帶機器人硬體部件、主要的通訊框架、YOLO 的障礙物辨識以及 vSLAM 的感測定位演算法架構；第三章著重於介紹 deep RL 的理論背景及算法間推導與關聯，並說明我們所開發的程序如何與 deep RL 融合運用；第四章為分別討論無障礙物與加入障礙物的實驗結果；第五章即為結論與未來方向。

第二章 田間履帶機器人及物件辨識定位

本章節介紹在田間履帶機器人上，從硬體設備到軟體端的應用。首先說明機器人的機電系統，接著介紹提供機器人通訊的機器人作業系統(robot operating system, ROS)框架，及本系統的通訊流程，最後會概述使用深度相機作為主要感測器，應用於 vSLAM 的即時定位以及 YOLO 實時物件偵測算法[6]的障礙物辨識及資訊提取，並詳細說明如何使用 ROS 框架將 YOLO 與深度相機之間相結合。

2-1 ROS 架構下履帶機器人

履帶機器人的重量約 200 kg，尺寸為長 1 m 和寬 0.7 m，高度考慮從地板至深度相機約為 0.75 m，如圖 2-1 所示。透過履帶間的差速做轉彎，履帶的特性可以使機器人克服大部分地形，我們將場域設定在田間，足夠的抓地力比起輪型機器人更能在田間靈活地行動。動力來源為兩組功率為 750 W 的無刷直流馬達(BLDC motor)，電力系統方面為兩具 48 V、20 Ah 之磷酸鐵鋰電池(lithium iron phosphate，LiFePO₄)，為所有系統提供電源供應。



圖 2-1 履帶機器人。

2-1-1 ROS

ROS 為主要運用在機器人上的通訊框架，其函式庫支援多種編程語言，也提供許多方便的工具，如 rosbag、rviz 以及 rqt 等。rosbag 能夠將 ROS 訊息記錄下來，以利日後回顧；而 rviz 是 ROS 提供的一種將訊息可視化的工具，如將像素和特徵點以圖像和點雲方式呈現；rqt 則是 ROS 的圖像使用者介面，提供節點和數據的監控及節點的變數調整等。目前 ROS 為最通用的機器人開發框架，其中機器人各部件之間的運行模式以節點(node)為單位，透過 P2P(peer-to-peer)的方式進行通訊，具有分散計算的優點，也可以避免在傳統序列程序中，會因為一個部件失效而整個程序被終止的情形，且只需要確定好節點的收發頻率，也具有提高整體運算效率的優點。

在節點中，部件之間的訊息收發透過話題(topic)和服務器(service)的 ROS 函式來進行。而節點間的通訊可以是一對多，亦可以是多對一。話題是屬於異步通訊，被分為發布者(publisher)以及訂閱者(subscriber)，發布者可以通過設定好特定頻率，不斷地發布訊息；而訂閱者亦同，可以不斷地訂閱來自於發布者的訊息。服務器則是屬於同步通訊，必須收到來自客戶端(client)的請求(request)訊息後，伺服器(server)才會發送一個回應(response)訊息，回應訊息可以被訂閱者或是其他伺服器所接收，客戶端的請求訊息則可以透過某個發布者亦或是程序中一般的條件式來發送。

2-1-2 履帶機器人 ROS 架構

文中的 ROS 架構初步平行使用了 6 個節點(這邊的節點以及話題名稱與實際名稱不同，僅供示意)來貫穿整體系統，包括了提供感測資訊的深度相機節點(Camera_node)；提供物件偵測資訊的 YOLO 節點(YOLO_node)；提供機器人位置(position)、姿態(orientation)以及速度(twist)的 vSLAM 節點(vSLAM_node)；處理障礙物資訊的 APL 節點(APL_node)等，再彙整成狀態空間提供給 DDPG 節點(DDPG_node)，最後將動作空間的速度值提供給 Arduino 節點(Arduino_node)後轉為電壓值，再經由 I²C 傳輸到 DAC，最終傳送至控制驅動器。最後我們會將這些節點進一步依時序序列式組合成單一節點來進行，以符合序列控制邏輯要求以及節省不必要的計算進而提升計算效率。

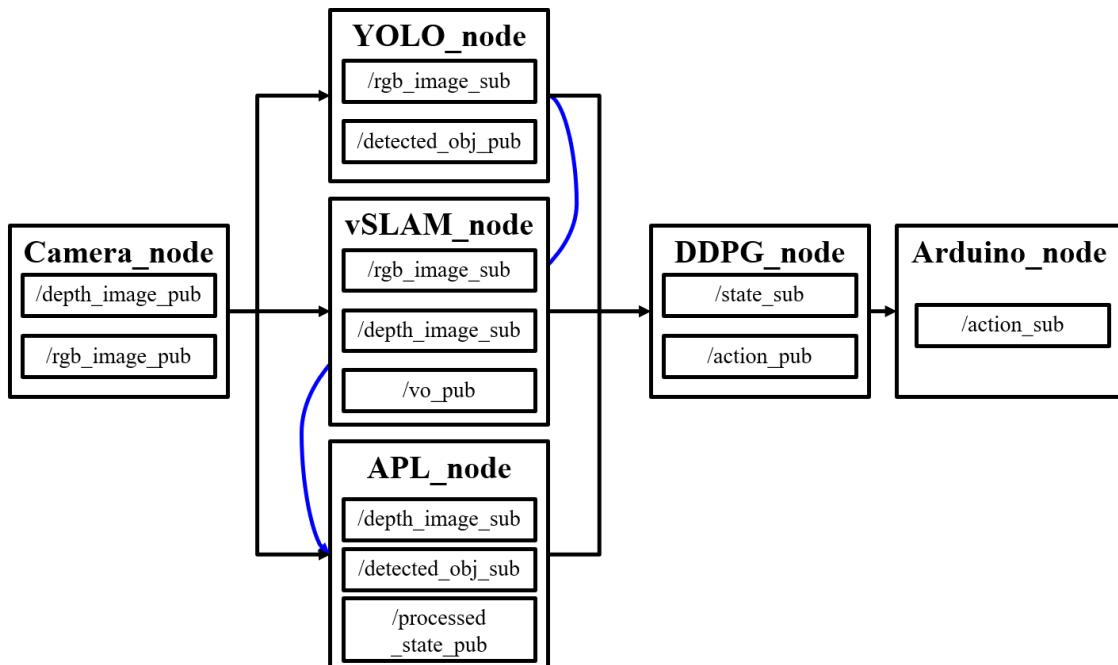


圖 2-2 履帶機器人 ROS 架構。

2-2 SLAM

在機器人對於當下環境一無所知的情況下，SLAM 能夠透過感測器所收到的資訊，將其分析成機器人的運動過程以實時地估計自身的位置以及姿態，並同步建立周圍環境的地圖模型，於感測器的不同也分為光達 SLAM 以及視覺 vSLAM。在本文中我們不會深入探討其原理，此次使用 vSLAM 作為實作實驗使用的定位工具，結合深度相機替機器人進行定位。

2-2-1 深度相機

在模擬中因為有模擬機器人模組的限制，所以只能使用 Intel Real sense R200 的深度相機模組；而在履帶機器人上，我們配置 Intel Real sense D435i 的深度相機模組，規格如表 2-1。

表 2-1 深度相機規格。

	R200	D435i
Depth Field of View (V/H/D, degrees)	43±2 / 70±2 / 77±4	58±1 / 87±3 / 95±3
Working Distance (m)	0.5 - 3.5	0.2 - 10
Frame Rate (fps)	30 @ 1920×1080 pixels	30 @ 1280×720 pixels
Resolution	0.1 mm	1 mm

深度圖的格式為 16UC1 (unsigned int 16 bit, 1 color)，且因解析度(一個數值代表實際單位)於 R200 為 0.1 mm，因此 16 bits (65535)可表示的最遠距離為 6.5535 m (65535×0.1 mm)；D435i 則為 65.535 m (65535×1 mm)。

2-2-2 vSLAM

本文使用 RTAB-Map 作為本次實作實驗中主要的 vSLAM 算法，架構如圖 2-3 所示。在已經有全局路徑的情況下，避障的過程即為局部路徑規劃，無須再進行環境地圖的建立，故在本實驗中省略了建圖的過程，僅利用前端視覺里程計(visual odometry, VO) [8]來進行定位，經由影像中特徵點的變化來估測機器人的位置及姿態。

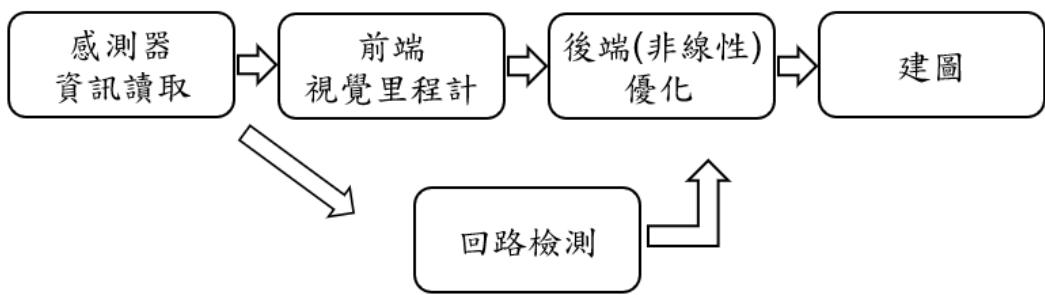


圖 2-3 vSLAM 架構[7]。

2-3 YOLO 辨識

本文主要的視覺感測來自於深度相機，在本文中使用 YOLO 中的 yolo-v3-tiny 作為偵測障礙物的物件偵測算法。

2-3-1 YOLO

YOLO 作為現行最普及的物件偵測算法，會使用 yolo-v3-tiny 是因為它的網路架構只有 24 層，比之 yolo-v3 的 107 層輕量了許多[9]，而輕量化的網路提供了幀速接近每秒 200 幀的實時偵測速度於輸入影像 320×320 像素，

其網路架構具有 13 層卷積層(convolution layer)提取圖像的特徵，6 層池化層(max pooling layer)用以強化特徵，2 層連接層(route layer)將層與層的特徵進行串接，1 層上採樣層(up sampling layer)將壓縮後的特徵解壓縮，和最後的 2 層 YOLO 層，計算完梯度下降的結果後輸出給 softmax 進行機率分類，並輸出偵測結果。

```
apl@apl-Sabre-15WVB:~/darknet$ ./darknet detector train /home/apl/darknet/build/darknet_yolov3-tiny-turtlebot3_obi
layer    filters   size      input           output
  0 conv    16  3 x 3 / 1   416 x 416 x 3    ->  416 x 416 x 16  0.150 BFLOPs
  1 max     2 x 2 / 2   416 x 416 x 16    ->  208 x 208 x 16
  2 conv    32  3 x 3 / 1   208 x 208 x 16    ->  208 x 208 x 32  0.399 BFLOPs
  3 max     2 x 2 / 2   208 x 208 x 32    ->  104 x 104 x 32
  4 conv    64  3 x 3 / 1   104 x 104 x 32    ->  104 x 104 x 64  0.399 BFLOPs
  5 max     2 x 2 / 2   104 x 104 x 64    ->  52 x 52 x 64
  6 conv   128  3 x 3 / 1   52 x 52 x 64    ->  52 x 52 x 128  0.399 BFLOPs
  7 max     2 x 2 / 2   52 x 52 x 128   ->  26 x 26 x 128
  8 conv   256  3 x 3 / 1   26 x 26 x 128   ->  26 x 26 x 256  0.399 BFLOPs
  9 max     2 x 2 / 2   26 x 26 x 256   ->  13 x 13 x 256
 10 conv   512  3 x 3 / 1   13 x 13 x 256   ->  13 x 13 x 512  0.399 BFLOPs
 11 max     2 x 2 / 1   13 x 13 x 512   ->  13 x 13 x 512
 12 conv  1024  3 x 3 / 1   13 x 13 x 512   ->  13 x 13 x 1024  1.595 BFLOPs
 13 conv   256  1 x 1 / 1   13 x 13 x 1024  ->  13 x 13 x 256  0.089 BFLOPs
 14 conv   512  3 x 3 / 1   13 x 13 x 256   ->  13 x 13 x 512  0.399 BFLOPs
 15 conv   21  1 x 1 / 1   13 x 13 x 512   ->  13 x 13 x 21   0.064 BFLOPs
 16 yolo
 17 route  13
 18 conv   128  1 x 1 / 1   13 x 13 x 256   ->  13 x 13 x 128  0.011 BFLOPs
 19 upsample          2x   13 x 13 x 128   ->  26 x 26 x 128
 20 route  19 8
 21 conv   256  3 x 3 / 1   26 x 26 x 384   ->  26 x 26 x 256  1.196 BFLOPs
 22 conv   18  1 x 1 / 1   26 x 26 x 256   ->  26 x 26 x 18   0.006 BFLOPs
 23 yolo
Loading weights from darknet53.conv.74...Done!
Learning Rate: 0.001, Momentum: 0.9, Decay: 0.0005
Resizing
```

圖 2-4 YOLO-v3-tiny 網路架構於 ubuntu 16.04。

2-3-2 ROS 感測結合流程

ROS 中的 YOLO 提供了 `darknet_ros` 節點來發布物件數量/`found_object` 以及物件位置/`bounding_boxes` 訊息；而在深度相機中，提供了 `camera` 節點發布 RGB 圖像/`rgb/image_raw` 及灰階深度圖像/`depth/image_raw` 訊息。故我們可以透過/`rgb/image_raw` 讓 YOLO 得以辨識物件後，利用/`bounding_boxes`

發布物件在圖像中像素位置，最後透過/depth/image_raw，得到被偵測物件與相機的直線距離。其中上述之結合過程由我們編寫的節點/dynamixel_node 進行處理。

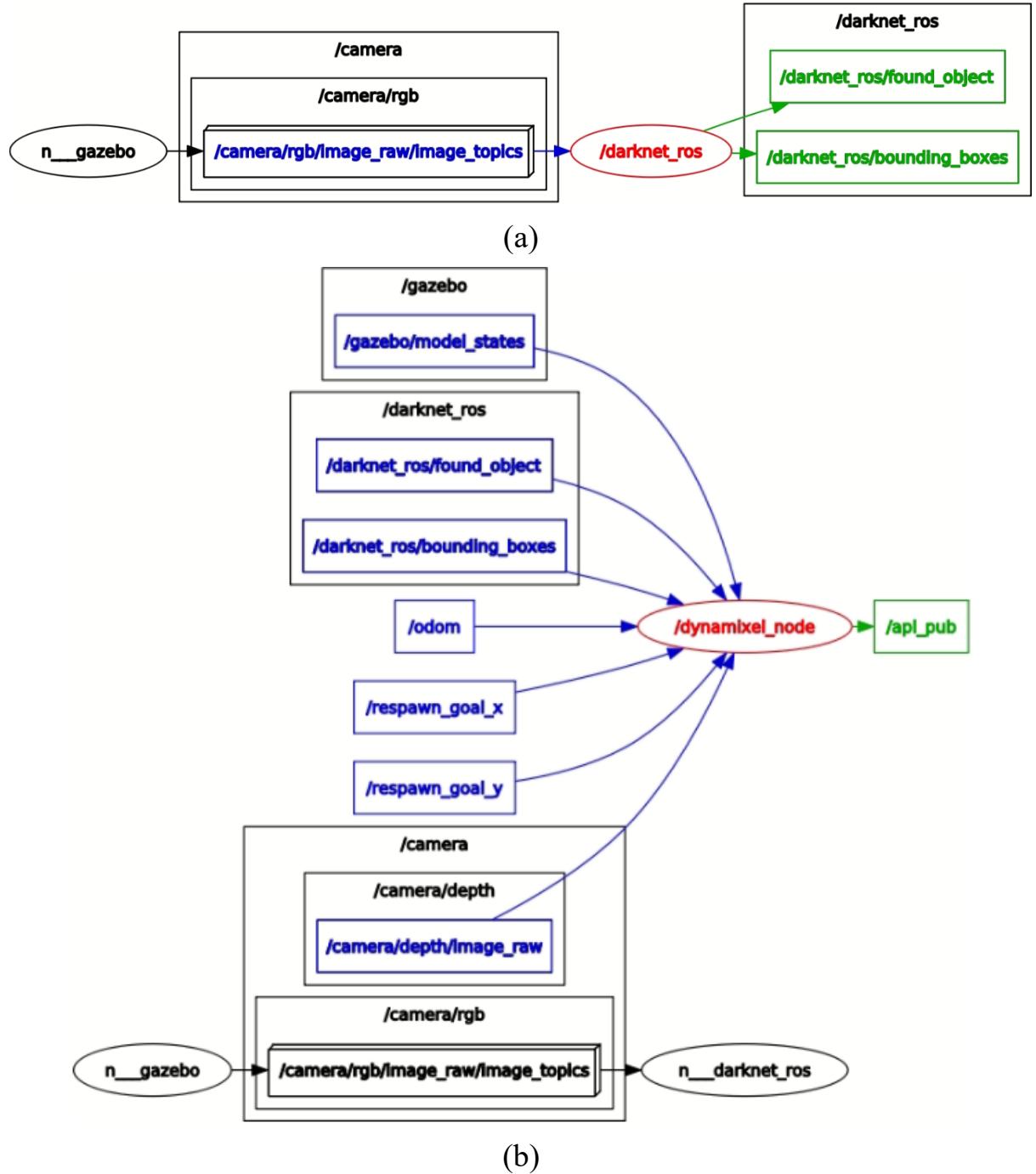


圖 2-5 rqt 節點監控 UI：(a) YOLO 物件偵測收發流程；(b)感測結合收發流程。

第三章 深度確定性策略梯度之理論與模擬

本章節描述 deep RL 的概念，從最初的馬可夫決策過程(Markov decision processes, MDPs)開始，介紹基於價值函數(value function)與策略梯度(policy gradient)的 deep RL 架構，再進一步探討 DDPG 的算法流程，並且說明本文中在模擬實驗上所應用的學習架構設計、控制模型和模擬環境搭建。

3-1 深度強化學習

DL 是以馬可夫決策過程為核心開發出來的一套有別於監督與非監督式學習的機器學習架構，旨在讓智能體能夠在獨立自主的情況下完成指派的任務。監督與非監督式學習是以尋找資料之間的結構相似度為主，差異在於監督式學習是訓練資料在擁有對應結構的條件下進行訓練，也就是所謂的資料標的；非監督式學習則是在訓練資料沒有對應的結構下，自行摸索訓練資料的結構。而強化學習是讓智能體不斷地探索環境後，將每時刻所獲得的過程(transition)來當作訓練資料，透過包含在過程中的狀態和獎懲來學習決策的能力，並無所謂的資料相似度，前面提到的過程指的是馬可夫鏈(Markov chain)或稱馬可夫決策樹，可被表示為 (s_t, a_t, r_t, s_{t+1}) ，以下會進一步說明。

3-1-1 馬可夫決策過程概述

作為 DL 的理論基礎，MDPs [10]是一種描述智能體與可觀測環境互動

過程的數學模型，以最簡單的形式包含感測(sensation)、動作(action)和目標(goal)。其中感測的部分是從環境(就好像控制的 plant)中觀測到的狀態空間集合 S (亦即 output vector)，動作則是指智能體(亦即具有controller)的動作空間集合為 A (亦即 input vector)，而目標就是獎懲值集合 R (亦即 cost function)。當智能體處在時刻 t 的環境中觀察到狀態 s_t ，被當作智能體的輸入後，透過決策機制的動作 a_t ，同時自己設計在環境中的獎懲函數會根據狀態或狀態-動作對(state-action pair)回傳獎懲 r_t ，進而影響智能體的決策機制，再得到下一時刻的狀態空間 s_{t+1} 。在學習中重複步驟以累積 transition 作為訓練資料。

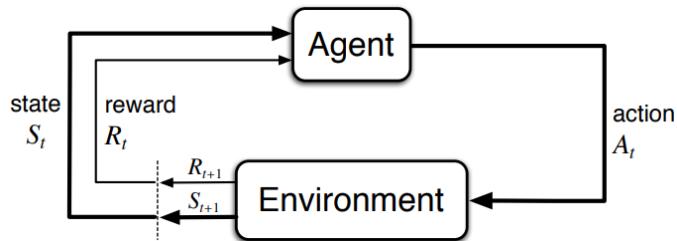


圖 3-1 MDPs 中智能體與環境交互過程圖[10]。

3-1-2 基於價值函數算法

對於智能體在特定狀態中做出的動作給予評價，亦或是給予狀態評價的數值函數，即稱為價值函數或評論器(value function/critic)，在這邊以 Q 學習(Q -learning)算法舉例[11]。在 Q 學習中價值函數以 $Q(s_t, a_t)$ 表示，此處的價值函數亦稱狀態動作價值函數(state-action value function)，意即在狀態 s_t 時採取動作 a_t 後，直到過程結束所預估的累積價值，而動作的選擇是來自於

Q 表格(Q -table)，利用貪婪策略(greedy policy)算法所得到：

$$A = \arg \max_A Q(S, A) \quad (3-1)$$

代表所選的動作 A 將在特定狀態 S 下得到最大的預估價值，其可以輕易地處理低維度且離散的動作空間問題，但當動作空間的維度較高且動作連續的時候，例如 $-\pi$ 到 π ，會產生維度災難(curse of dimensionality)，為了解決高維度動作空間的問題，因此導入深度學習形成了深度 Q 網路學習(deep Q network，DQN)演算法，如圖 3-2，但仍然只能處理離散動作空間的問題。

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

 With probability ε select a random action a_t

 otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

 Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

 Every C steps reset $\hat{Q} = Q$

End For

End For

圖 3-2 DQN 演算流程[12]。

DQN 學習算法[12]擁有兩個網路，網路 Q 和目標網路 \hat{Q} ，他們的初始內部參數 θ 會是相同的， Q 是實際與環境互動的網路，結合貪婪策略替學習過程收集 MDPs transition 並存取至訓練資料集，這裡稱為經驗回放區

(experience replay)或稱回放緩衝區(replay buffer) D ；而 \hat{Q} 存在的意義是提供目標 y_i 來牽制學習震盪幅度大的 Q 。經驗回放區 D 是為了讓訓練的過程從線上訓練方法(on-policy methods)轉換成離線訓練方法(off-policy methods)，使取樣的資料關聯性降低，避免訓練過擬合(overfitting)的情況發生。

在 DQN 學習算法的學習更新中，適合使用動態規劃(dynamic programming) [13]的方式來求解最佳問題，因為在前面提到價值函數 $Q(s_t, a_t)$ 是狀態 S_t 時採取動作 A_t 後，直到過程結束所預估的累積價值，所以可以藉由 MDPs 特性，將 $Q(s_t, a_t)$ 記作：

$$Q(s_t, a_t) = r_t + Q(s_{t+1}, a_{t+1}) \quad (3-2)$$

此 $Q(s_t, a_t)$ 依賴 $Q(s_{t+1}, a_{t+1})$ 的特性，已符合動態規劃的重疊子問題特性(property of overlapping sub-problems)，再透過貝爾曼方程式(Bellman equation)，把尋找最優狀態動作價值函數的過程看作是利用當前獎懲值 r_t 加上下一時刻最佳預估價值 $\max Q'(s_{t+1}, a_{t+1})$ 的方式來求得：

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha[r_t + \gamma \max Q'(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (3-3)$$

其中 α 為學習率， γ 為折扣因子(discount factor)，求得最優狀態動作價值函數的方式使用 MDPs 劃分為子問題來求解，已符合動態規劃中的最優子結構特性(property of optimal substructure)，可以在每次動態規劃求優的過程中，透過實際所得到的獎勵來尋找最優狀態動作價值函數 $Q(s_t, a_t)$ 以更新網路。

3-1-3 基於策略函數算法

策略函數或稱執行器(policy function/actor)是接收狀態並輸出動作的函數，用 π_θ 表示，其中 θ 代表當前策略函數的內部參數，為神經網路中隱藏層(hidden layers)的濾波器權重值的集合代稱，而濾波器是用於提取輸入資料的特徵，在這裡是用於提取狀態的特徵。在基於策略函數演算法(policy-based function algorithms)中如何選擇動作是根據 transition 所獲得的獎懲總和大小來決定。以策略梯度(policy gradient, PG)學習算法為例[14]，為了確保智能體的探索能力，此學習算法的動作選擇是一個機率問題，在學習過程中使用梯度上升的方式去找尋能讓獎懲總和期望值最大的動作，以用來更新策略網路的內部參數(亦即 self-tuning controller)，此目的是希望讓智能體的動作選擇可以被影響，以朝向獎懲總和大的情況靠攏，進而提高樣本的使用效率。首先智能體會累積 N 筆 transition τ 作為一次的訓練資料， $\tau \in N$ 。每筆 τ 的獎懲總和表示為：

$$R(\tau) = \sum_{t=1}^T r_t \quad (3-4)$$

而取樣到 τ 的機率表示為：

$$P_\theta(\tau) = P(s_1) \prod_{t=1}^T P_\theta(a_t | s_t) P(s_{t+1} | s_t, a_t) \quad (3-5)$$

由式(3-4)與式(3-5)可得 π_θ 的獎懲期望值 $\overline{R_\theta}$ ，同時也是我們欲執行梯度上升的目標函數，表示為：

$$\overline{R_\theta} = \sum_{\tau} R(\tau) P_\theta(\tau) = E_{\tau \sim P_\theta(\tau)} [R(\tau)] \approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \quad (3-6)$$

則得到：

$$\theta^* = \arg \max_{\theta} \overline{R_\theta} = \nabla \overline{R_\theta} = \sum_{\tau} R(\tau) \nabla P_\theta(\tau) = \sum_{\tau} R(\tau) P_\theta(\tau) \frac{\nabla P_\theta(\tau)}{P_\theta(\tau)} \quad (3-7)$$

$$\approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log P_\theta(\tau^n) = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T R(\tau^n) \nabla \log P_\theta(a_t^n | s_t^n) \quad (3-8)$$

由式(3-8)的結果來看，可以直觀地認為獎懲總和 $R(\tau^n)$ 是 $P_\theta(a_t^n | s_t^n)$ 的權重，所以當 $R(\tau^n)$ 越大，會同時提高在 s_t^n 的狀態下選擇 a_t^n 的機率，就會進一步影響該 τ^n 出現的機率，使其在未來可以被更容易地取樣。網路內部參數更新如式(3-9)，其中 η 表學習率：

$$\theta^{new} \leftarrow \theta^{old} + \eta \nabla \overline{R_{\theta^{old}}} \quad (3-9)$$

3-2 深度確定性策略梯度之演算法

為了解決 DQN 不擅長處理連續動作空間的問題，2016 Google DeepMind 發表 DDPG 學習算法[15]，此結合了確定性策略梯度演算法 (deterministic policy gradient, DPG) [16] 以及 DQN，而演算流程以 DQN 為主，保留了目標網路、回放緩衝區以及動態規劃求優等經典部件。處理連續動作空間是因為將 DPG 融於 DQN 貪婪策略算法中，策略函數用 μ 表示：

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_A Q(S, A | \theta^Q) |_{S=s_i, A=\mu(s_i)} \nabla_{\theta^\mu} \mu(S | \theta^\mu) |_{s_i} \quad (3-10)$$

有別於 PG 的隨機決策函數 $\pi_\theta(a_t^n | s_t^n) = P_\theta(a_t^n | s_t^n)$ ，DPG 能夠在特定狀態直接選擇特定動作 $\mu_\theta(s_t^n)$ ，也因為 DPG 是與 DQN 結合，並不會喪失探索的能力，最後，在 μ 網路最後的輸出層被設定為 \tanh 層來限制連續動作。

在 DDPG 中共有四個網路，critic 網路 Q 、目標 critic 網路 Q' 、actor 網路 μ 以及目標 actor 網路 μ' ，以及回放緩衝區 R (在 DQN 以 D 表示，意義為 dataset)。圖 3-4 是根據圖 3-3 所敘述的演算流程所製作的架構圖。以下將詳

細介紹 DDPG 的演算流程[15]。

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
 Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
 Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process \mathcal{N} for action exploration
 Receive initial observation state s_1
for t = 1, T **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:
 $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$
 $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$

end for
end for

圖 3-3 DDPG 演算流程[15]。

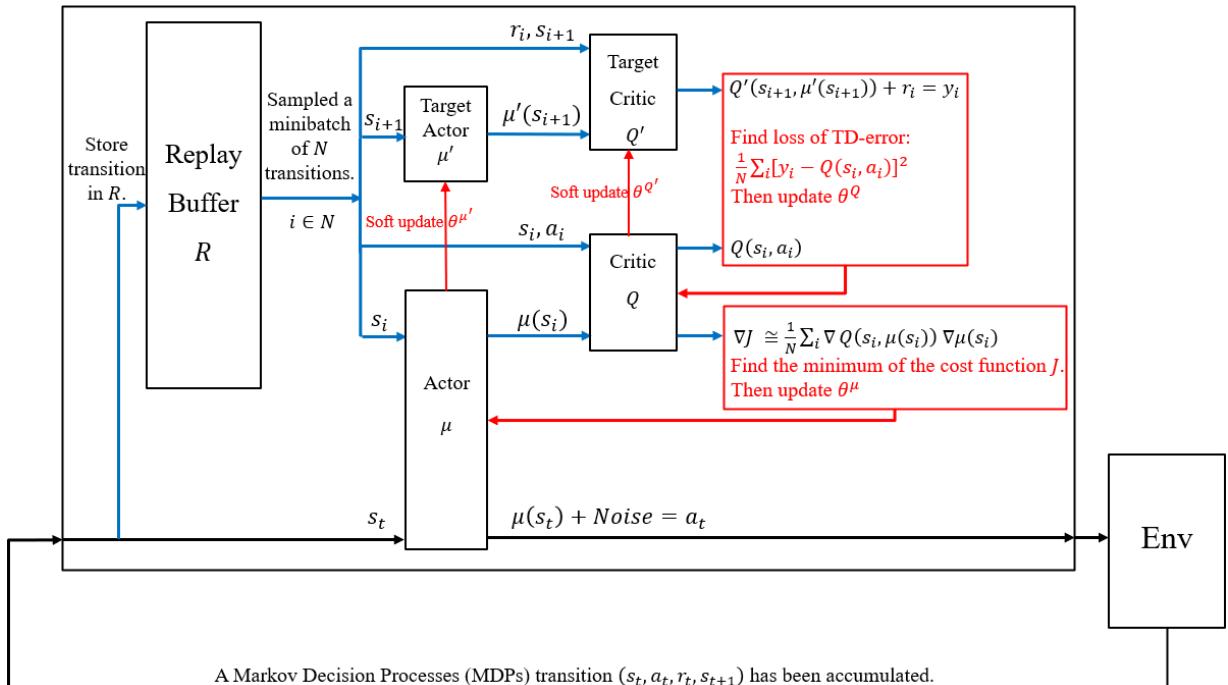


圖 3-4 DDPG 架構圖。

1. 首先在 t 時刻從環境中觀測到狀態 s_t ，輸入給 actor 網路 μ 並輸出 $\mu(s_t)$ ，再加上隨機雜訊後，形成動作 a_t 與環境互動。
2. 累積一段 transition (s_t, a_t, r_t, s_{t+1}) 後存進 R ，並從 R 中取樣 N 筆 transition (s_i, a_i, r_i, s_{i+1}) ，其中 $i \in N$ ，當作訓練資料後各自輸入至四個網路中進行訓練。
3. 在更新 actor 網路的學習中， s_i 輸入給 actor 網路 μ 並輸出 $\mu(s_i)$ 後，將其輸入給 critic 網路 Q 並輸出 $Q(s_i, \mu(s_i))$ 。再用貪婪算法結合 $\mu(s_i)$ 和 $Q(s_i, \mu(s_i))$ 兩者：

$$\nabla J \approx \frac{1}{N} \sum_i \nabla Q(s_i, \mu(s_i)) \nabla \mu(s_i) \quad (3-11)$$

在實作上會給 $Q(s_i, \mu(s_i))$ 負號，讓它成為負值，用損失函數 $L(Q(s_i, \mu(s_i)))$ 方式以尋找擁有最小價值的動作 $\mu(s_i)$ ：

$$Q^*(s_i, \mu(s_i)) = \arg \min_{Q(s_i, \mu(s_i))} L(Q(s_i, \mu(s_i))) \quad (3-12)$$

更新 actor 網路內部參數 θ^μ 求得最優策略函數，其中 α 為學習率：

$$\theta^\mu = \theta^\mu + \alpha \mu(s_i) \frac{dL}{dQ(s_i, \mu(s_i))} |_{Q(s_i, \mu(s_i))=Q^*(s_i, \mu(s_i))} \quad (3-13)$$

再用軟更新(soft update)的方式同步更新目標 actor 網路 μ' 的內部參數 $\theta^{\mu'}$ ，

且 τ 為軟更新的學習率：

$$\theta^{\mu'} = \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \quad (3-14)$$

4. 在更新 critic 網路的學習中， s_{i+1} 輸入給目標 actor 網路 μ' 並輸出 $\mu'(s_{i+1})$

後，再將其輸入目標 critic 網路 Q' 並輸出 $Q'(s_{i+1}, \mu'(s_{i+1}))$ ，乘上 γ 後再加上 R_i 即得到最優目標價值(optimal target value)：

$$\gamma Q'(s_{i+1}, \mu'(s_{i+1})) + r_i = y_i \quad (3-15)$$

再與 critic 網路 Q 輸出的 $Q(s_i, a_i)$ 組成目標函數：

$$J = \frac{1}{N} \sum_i [y_i - Q(s_i, a_i)]^2 = Q_{MSE} \quad (3-16)$$

用損失函數 $L(Q_{MSE})$ 方式，尋找最小平方差的最小值：

$$Q_{MSE}^* = \arg \min_{Q_{MSE}} L(Q_{MSE}) \quad (3-17)$$

更新 critic 網路內部參數 θ^Q 求得最優動作價值函數：

$$\theta^Q = \theta^Q + \alpha \frac{dL}{dQ_{MSE}} |_{Q_{MSE}=Q_{MSE}^*} \quad (3-18)$$

再用軟更新的方式同步更新目標 critic 網路 Q' 的內部參數 $\theta^{Q'}$ ：

$$\theta^{Q'} = \tau \theta^Q + (1 - \tau) \theta^{Q'} \quad (3-19)$$

3-3 Simulink 控制與學習架構

文中機器人的 DDPG 學習與控制架構皆在 MATLAB Simulink 上開發，MATLAB 提供 ROS 的工具包，可以編寫 ROS 節點，透過基於 HTTP (HyperText Transfer Protocol) 通訊協議的 ROS 網路，串連起在 Simulink 中的 DDPG 智能體和機器人之間的節點來進行 ROS 通訊，如圖 3-5 所示。

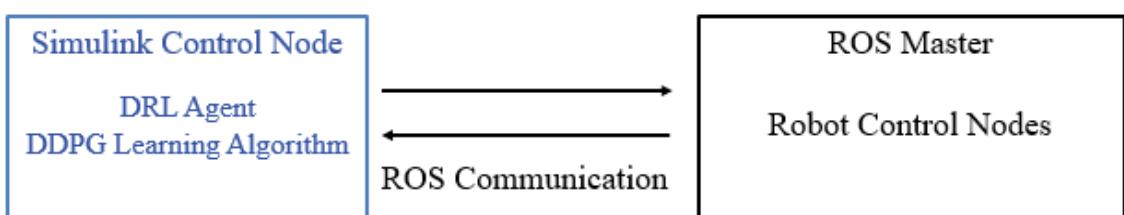


圖 3-5 MATLAB 與機器人 ROS 通訊方法。

3-3-1 Simulink 控制架構

於圖 3-6 中，智能體即為我們的控制模型，包含控制器，方塊名稱為 RL Agent，詳細演算流程已在圖 3-4 討論完畢，其取樣頻率為 10 Hz。

1. Sensors 方塊接收到來自感測器的 ROS 訊息後，提供給 Calculate Observation 方塊、Calculate Reward 方塊以及 Check if Done 方塊進行運算及邏輯判斷。
2. Calculate Observation 方塊提供當前時刻的狀態空間給 RL Agent 方塊，經過 actor 網路後發布 2 維度的線速度和角速度動作空間給 APL Robot 方塊將數據轉換為 ROS 訊息格式後，經由/cmd_vel 發布給機器人與環境互動，後再由 sensors 方塊訂閱新的狀態空間。
3. 而 Calculate Reward 方塊包含著獎懲函數，處理著當前時刻所得到的獎懲值。
4. Check if Done 方塊則為判斷在當前時刻，任務是否完成或失敗，以利進行任務重置以及 MDPs transition 儲存。

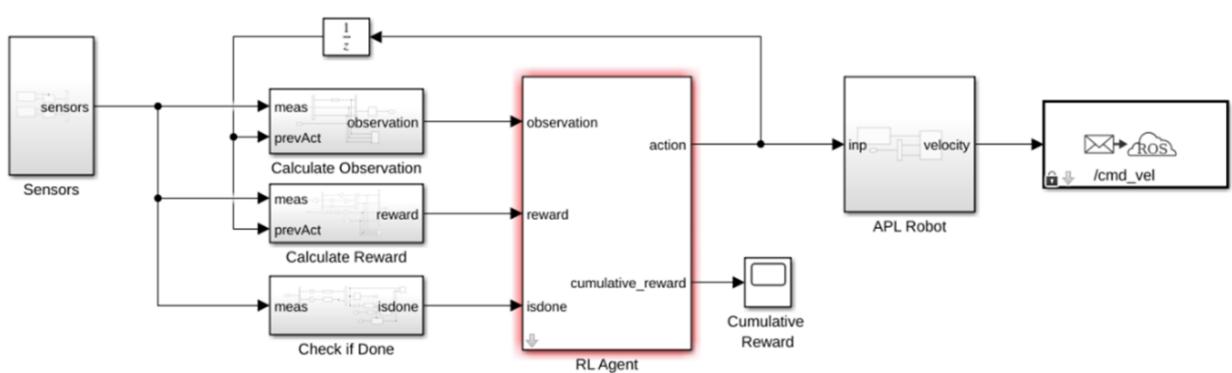


圖 3-6 Simulink 控制架構。

控制模型的節點為 APLRL2D_oneAct，於圖 3-7 淡藍色框表示。而 sensors 除了直接訂閱機器人(X,Y)全局位置資訊 /gazebo/odom 以及 /darknet_ros/found_object 物件數量外，也必須訂閱我們所負責處理視覺訊息的 dynamixel_node 節點，為圖 3-7 中紅圈表示。

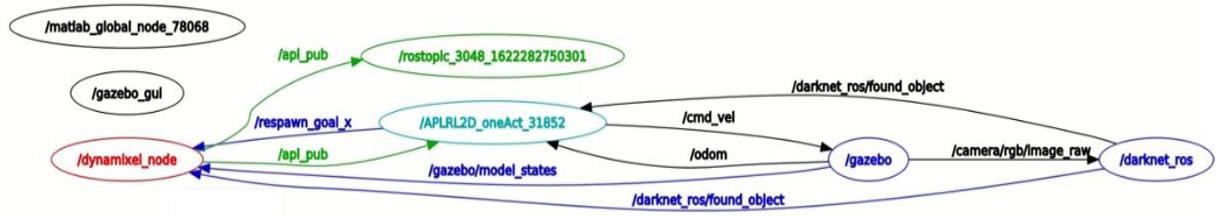


圖 3-7 rqt 節點監控 UI。

3-3-2 MATLAB DDPG 架構

使用 critic 網路架構為 9 層且學習率 10^{-3} ，actor 網路架構為 7 層且學習率 10^{-4} ，優化器皆使用 Adam。訓練參數設置上，回放緩衝區長度為 10^5 ，取樣頻率 10 Hz，單次訓練資料為 64 筆 MDPs transitions，折扣因子 γ 為 0.99，以及軟更新(soft update)學習率 10^{-3} ，圖 3-8 為在 MATLAB Deep Network Designer 中顯示的網路架構。

3-4 TurtleBot3 於 Gazebo 環境模擬

在成本的考量上，我們以深度相機作為主要的機器視覺，所以在模擬上我們使用與實際履帶車上的 Real Sense D435i 最相近的配備 Real sense R200 深度相機模組的 TurtleBot3 Waffle 做為模擬機器人，其尺寸為長 28 cm，寬

31 cm 和高 14 cm，重量為 1 kg，實體圖如圖 3-9 所示。

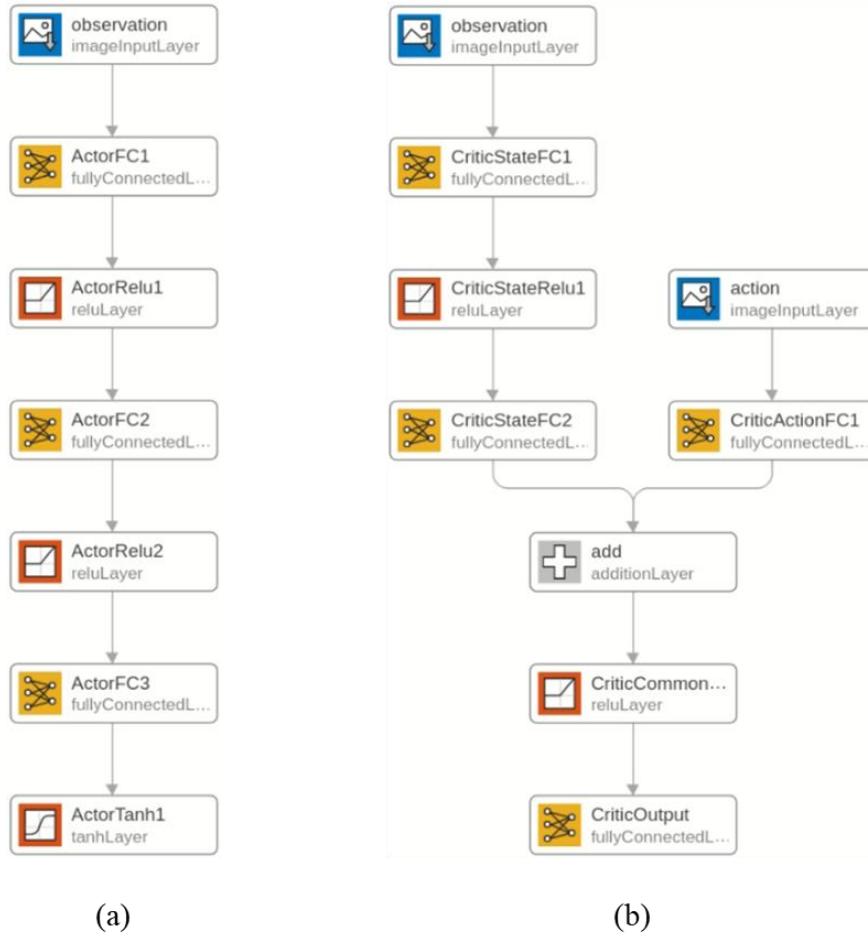


圖 3-8 DDPG 網路架構：(a) actor 網路架構；(b) critic 網路架構[17]。

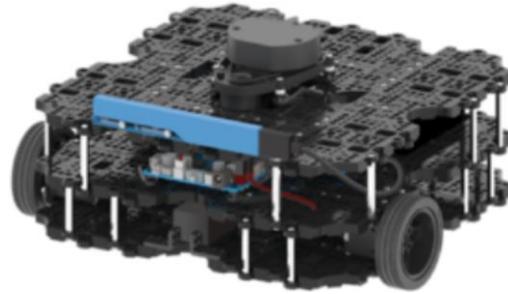


圖 3-9 TurtleBot3 Waffle [18]。

模擬環境為 Gazebo，它是一款支援 ROS 以及 GPU 加速的 3D 模擬器，提供多種物理引擎以及虛擬模組，如光達(LiDar)等，能夠擁有高真實度的模擬環境。如圖 3-10 即為實際於 Gazebo 環境運行 YOLO 物件偵測的模擬畫

面。

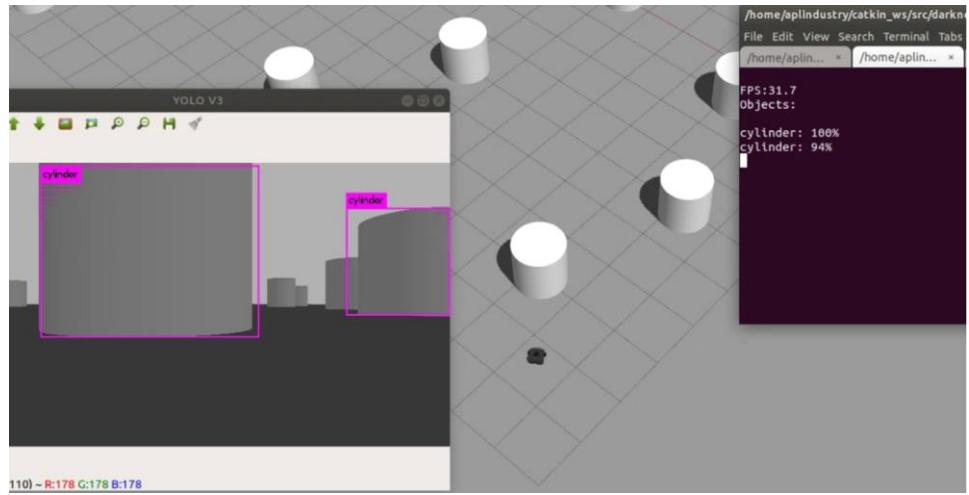


圖 3-10 Gazebo 環境運行 YOLO 物件偵測。

第四章 實驗討論與結果

實驗分為兩部分，在無障礙物環境中的追點控制，及在有障礙物環境的追點控制結合避障控制，都是使用無模型方式進行學習。而所謂無模型方式是指，在訓練的一開始，智能體與環境互動的過程中，無從知曉任務的規則，且無任何線索得以遵循，必須經過智能體長時間地探索和利用後，才能學習出來的模型。反之，基於模型方式是在訓練的一開始即已知曉任務的規則，能夠有規則地預測出下一步的動作為何，例如，我們可以針對模型預測控制(mode predictive control, MPC) [19] 中的動力學模型利用基於模型方式先行學習，將神經網路模型作為動力學模型(dynamics model)訓練出可以準確估計狀態的模型，再由無模型方式繼續進行優化學習[20]。首先在 Gazebo 模擬環境中進行訓練，待模擬部分實驗驗證完後再於履帶機器人上進行實作測試，且不論在模擬環境中還是現實履帶機器人中所使用的座標都如圖 4-1 所示[21]。在本章節中，會討論學習中所考慮的狀態空間以及詳述獎懲函數的設置過程。

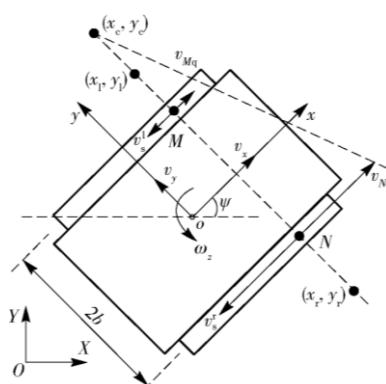


圖 4-1 履帶機器人運動學模型(kinematic model) [21]。

4-1 履帶機器人追點控制

為了塑造單純的訓練環境以進行純追點控制的實驗，將不考慮任何障礙物干擾的情況，為求訓練出一個智能體能夠控制機器人完成從起點走到終點的任務，環境如圖 4-2，其中機器人的初始點設置在全局座標系 (X, Y) 中的 $(-9.5, -9.5)$ ，及終點位於 $(-5.5, -5.5)$ 。判斷任務成功或失敗的條件有三個，當機器人距離終點小於 0.5 m 或大於 8.15 m 時，及模擬時間超過取樣區間時，就會重置機器人於初始位置並且將動作空間值歸零。

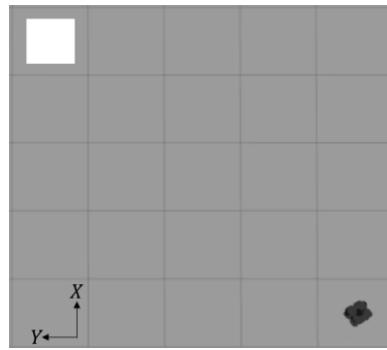


圖 4-2 追點控制 Gazebo 訓練環境。

在訓練的第一個版本中，一個 episode 的取樣區間 $T_f = 40$ 、取樣週期 $T_s = 0.1$ ，動作空間為線速度 v_x 以及角速度 ω_z ，約束設置皆為 -0.5 至 0.5 m/s 及 rad/s，狀態空間為：

$$s_t = \begin{bmatrix} d_{goal}(t) \\ t \\ v_x(t-1) \\ \omega_z(t-1) \end{bmatrix} \quad (4-1)$$

其中 $d_{goal}(t)$ 表示機器人與終點的直線距離。而獎懲函數為：

$$r_t = 100v_x(t) + \frac{200}{1+d_{goal}(t)^2} - 10t \quad (4-2)$$

其中第一項 $100v_x(t)$ 是希望讓機器人速度為正且值越大越好；第二項位能項： $\frac{200}{1+d_{goal}(t)^2}$ ，是讓機器人距離終點越近，值就越大的位能函數；第三項是時間的懲罰，目的是不希望機器人花費太久的時間達成目標。圖 4-3 是其訓練兩天後的統計結果，黃線代表在當前 episode 一開始時所預估出的 Q 值，藍線為當前 episode 實際獲得的獎懲值總和，紅線代表當前 episode 到前 100 筆間的獎懲值平均值。

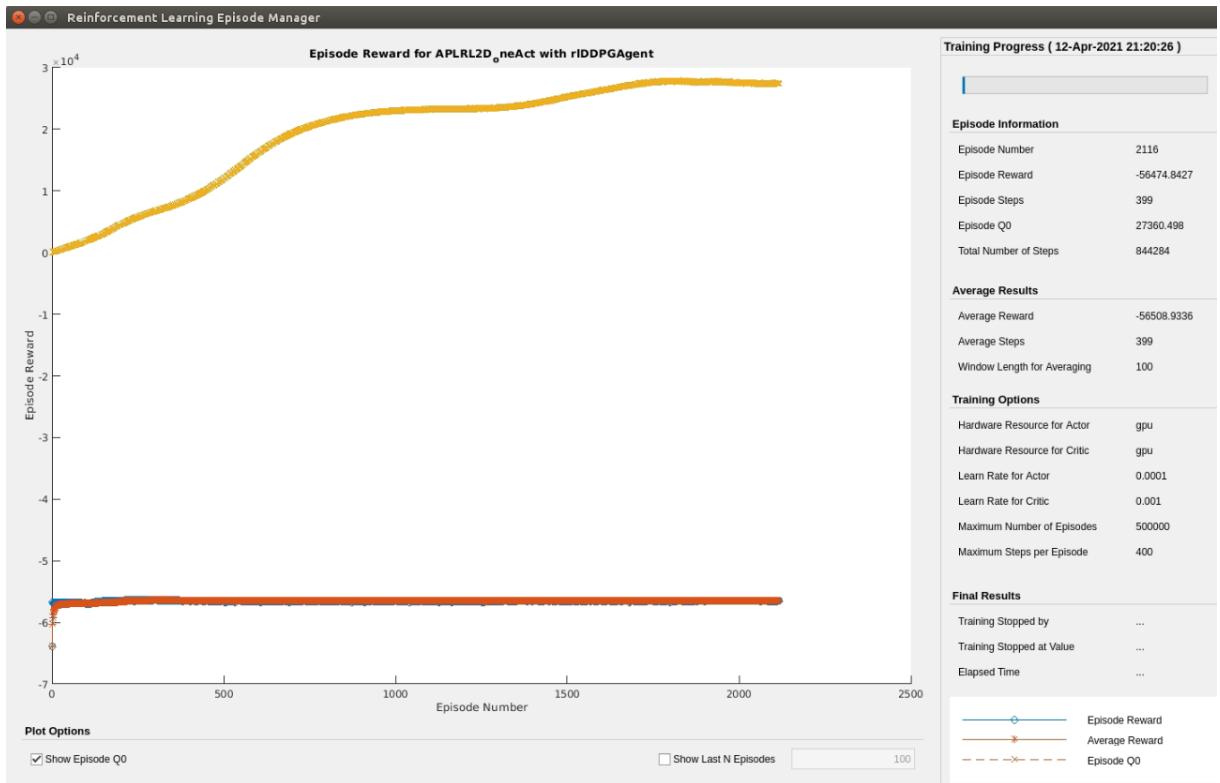


圖 4-3 追點控制第一版訓練統計。

第一版訓練是失敗的，初步研判是第一項速度獎勵權重太大的關係，且第二項位能獎勵在 episode 最一開始時的獎勵太小，以至於無法有效引導機器人，也基於賽局理論(game theory)，在一開始對於怎麼下決策一無所知的情況下，只會選擇當前最容易得到最大利益的決策，所以最後學習出一個會

在一開始就把 v_x 提到最大，在原地繞圓圈的智能體。

在第二版中，我們只改變了獎勵函數，降低第一項速度獎勵的權重，並提高第二項位能獎勵的權重，同時也用指數的方式取代了原本線性的時間懲罰，如圖 4-4，目的是為了讓智能體有更多的時間探索，不至於在前期就承受很大的懲罰。圖 4-5 是第二版訓練後的統計結果。

$$r_t = 20v_x(t) + \frac{600}{1+d_{goal}(t)^2} - 6e^{0.2t-e^1} \quad (4-3)$$

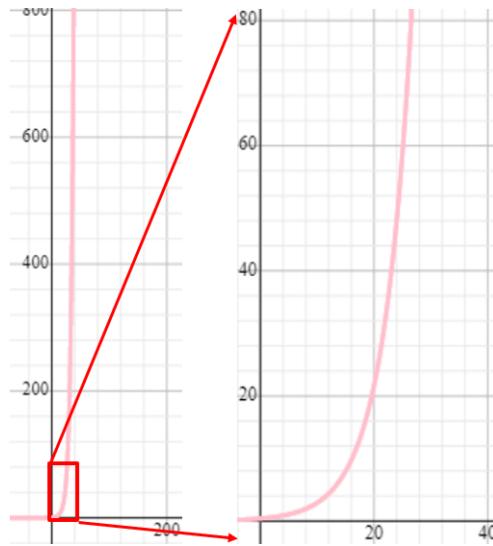


圖 4-4 時間懲罰函數。



圖 4-5 追點控制第二版訓練統計。

第二版訓練相較第一版的智能體有探索能力，但行為上仍然無法符合預期，仍有繞圓圈的情況。在第三版中，針對狀態空間和獎懲函數做出修正，新狀態空間為：

$$s_t = \begin{bmatrix} d_{goal}(t) \\ \Psi_{goal}(t) \\ t \\ v_x(t-1) \\ \omega_z(t-1) \end{bmatrix} \quad (4-4)$$

新增了 $\Psi_{goal}(t)$ 為機器人正前方與終點之間的偏航角(yaw)角度，範圍為 $-\pi$ 至 π ，新獎懲函數為：

$$r_t = 20v_x(t) + \frac{600}{1 + d_{goal}(t)^2} - 6e^{0.2t-e^1} + 10e^{10(\Delta d_{goal})^2} | if \Delta d_{goal} > 0$$

$$20v_x(t) + \frac{600}{1 + d_{goal}(t)^2} - 6e^{0.2t-e^1} - 10e^{10(\Delta d_{goal})^2} | if \Delta d_{goal} < 0 \quad (4-5)$$

相較第二版，多考慮了 d_{goal} 變化量 $\Delta d_{goal} = d_{goal}(t - 1) - d_{goal}(t)$ ，作為第四項新獎懲，目的是引導機器人在每個時刻都能朝向目標前進，試圖消除繞圓圈的行為。圖 4-6 為第三版訓練完的統計結果，綠線代表在當前 episode 一開始時所預估出的 Q 值，藍線為當前 episode 實際獲得的獎懲值總和，紅線代表當前 episode 到前 100 筆間的獎懲值平均值。

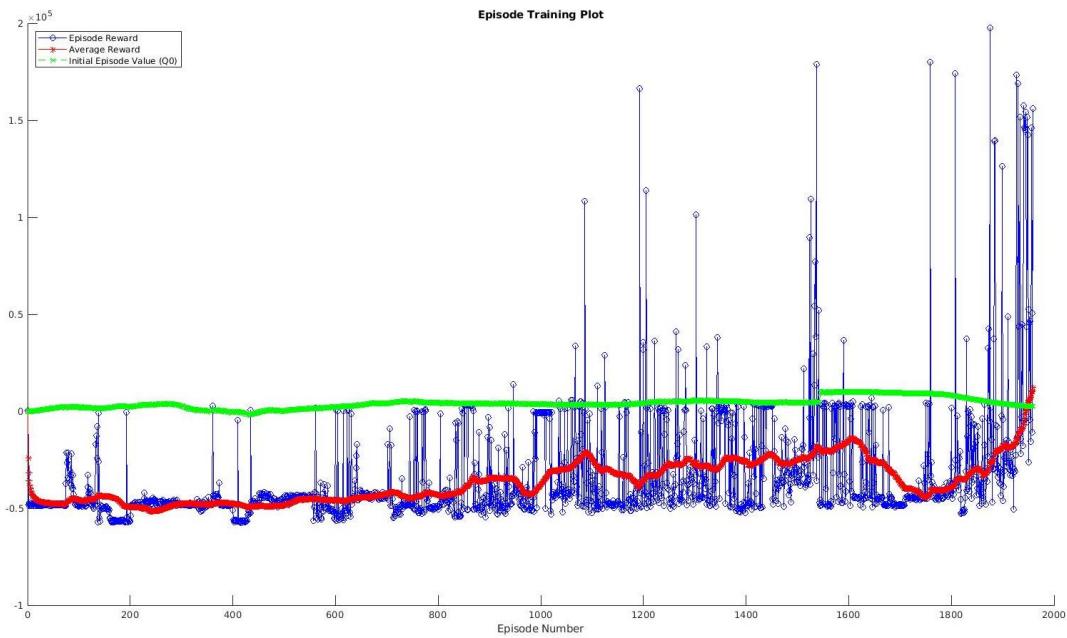


圖 4-6 追點控制第三版訓練統計。

第三版在行為上，有向終點前進的結果，但還是有繞圓圈的情況發生，也會在還沒抵達終點的情況下，出現走完 400 個時間步後總獎勵卻很高的情況，這種行為在有設置時間懲罰的條件下，並不符合期望。所以在第四版繼續針對獎懲函數予以修正，新的獎勵函數為：

$$r_t = \begin{cases} 10^3 v_x(t) - 10^4 |\omega_z(t)| \\ 200 \cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} + 200 |if_{d_{goal}(t) < d_{goal}(0)} \\ -200 \cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} - 200 |if_{d_{goal}(t) > d_{goal}(0)} \\ -6 \times e^{0.3t - e^1} \\ 100e^{10(\Delta d_{goal})^2} |if_{\Delta d_{goal} > 0} \\ -100e^{10(\Delta d_{goal})^2} |if_{\Delta d_{goal} < 0} \\ 10^5 |if_{d_{goal}(t) < 0.6m} \end{cases} \quad (4-6)$$

在第一項速度獎勵方面多考慮了對角速度絕對值的懲罰，試圖使角速度的值減小，並修改第二項位能獎懲，讓太過於遠離終點會有懲罰，也提高了第三項時間懲罰和第四項距離變化量獎懲的權重，最後增加第五項達到終點的大獎勵，同時將 T_f 的長度從 40 秒降至 30 秒。圖 4-7 為第四版訓練完的統計結果。

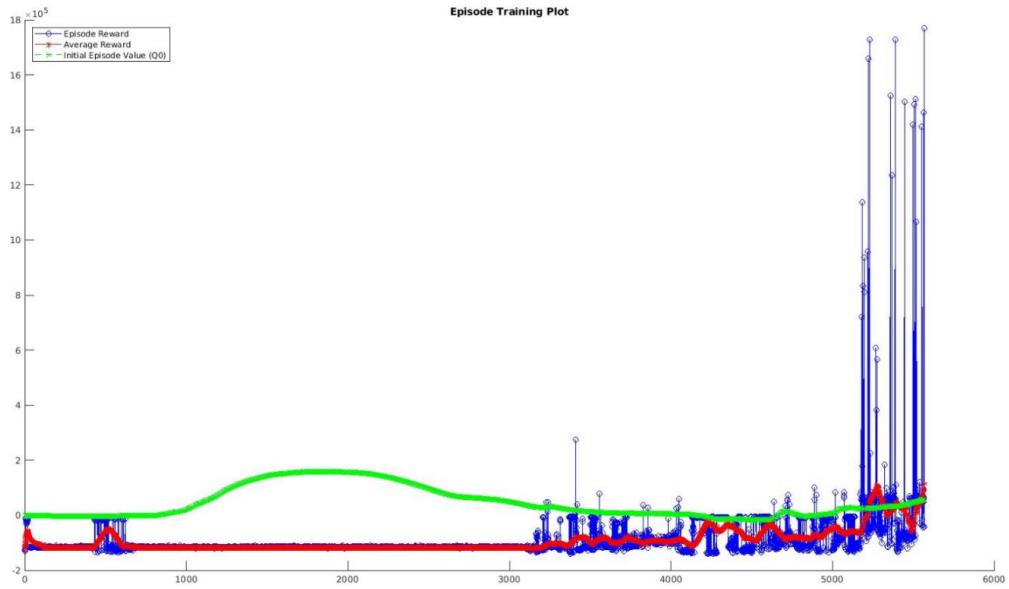


圖 4-7 追點控制第四版訓練統計。

直至第四版才最終學習到符合預期的行為，但有一些瑕疵，因為智能體會在距離終點小於 0.6 m 但大於 0.5 m 的周圍繞圓圈，以收取最大化累積獎

勵，如圖 4-8 所示。

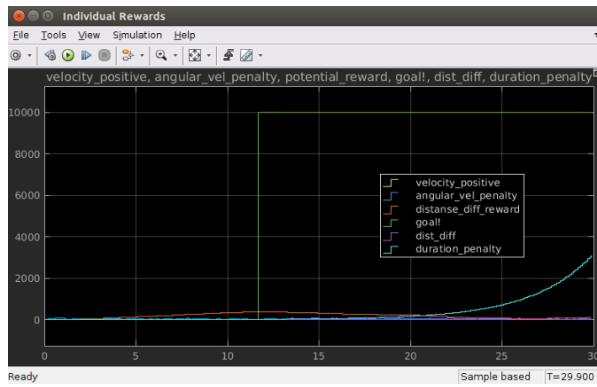


圖 4-8 追點控制第四版獎懲值累積情形。

4-1-1 模擬測試

驗證的結果中，歷時 132 個取樣時間抵達終點，但機器人會傾向在終點周圍繞圓圈。雖然機器人的行為有瑕疵，但確實是符合獎懲函數的理念，在取樣區間內獲取最大獎勵走到終點，結果如圖 4-9。圖 4-10 至 4-12 中的橫軸為時間。

圖 4-10 黃線表示狀態空間中的 $d_{goal}(t)$ ，圖 4-11 綠線顯示最終得到到達終點的獎勵，圖 4-12 黃線顯示最終於條件為任務完成。

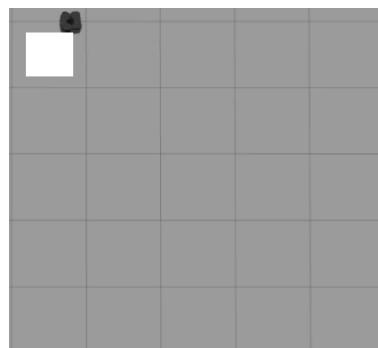


圖 4-9 純追點模擬驗證中機器人從起點走向終點。

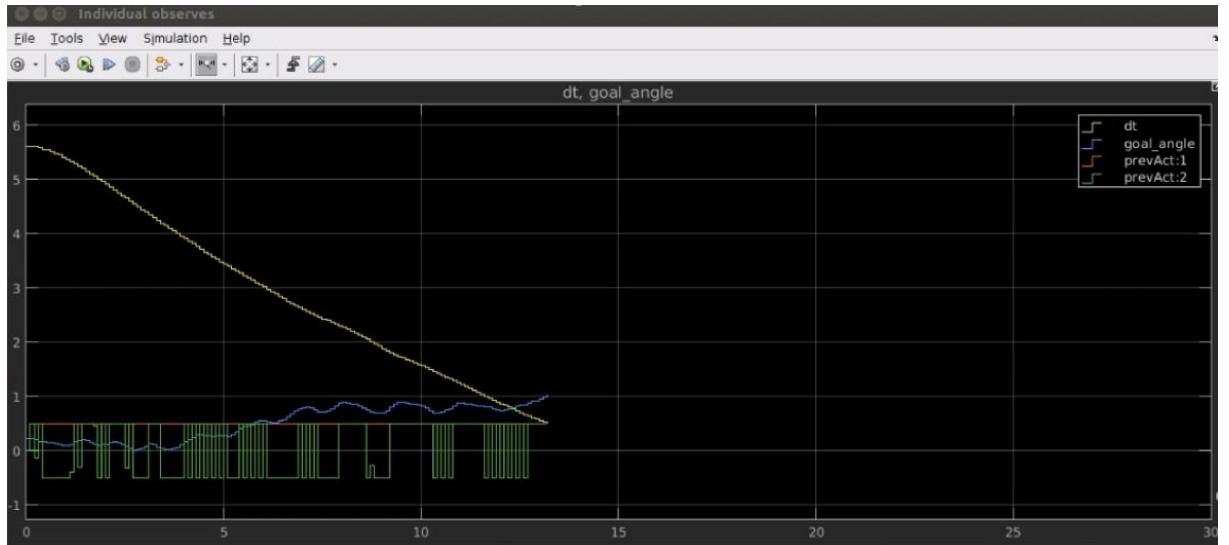


圖 4-10 純追點模擬驗證中機器人每時刻的狀態。

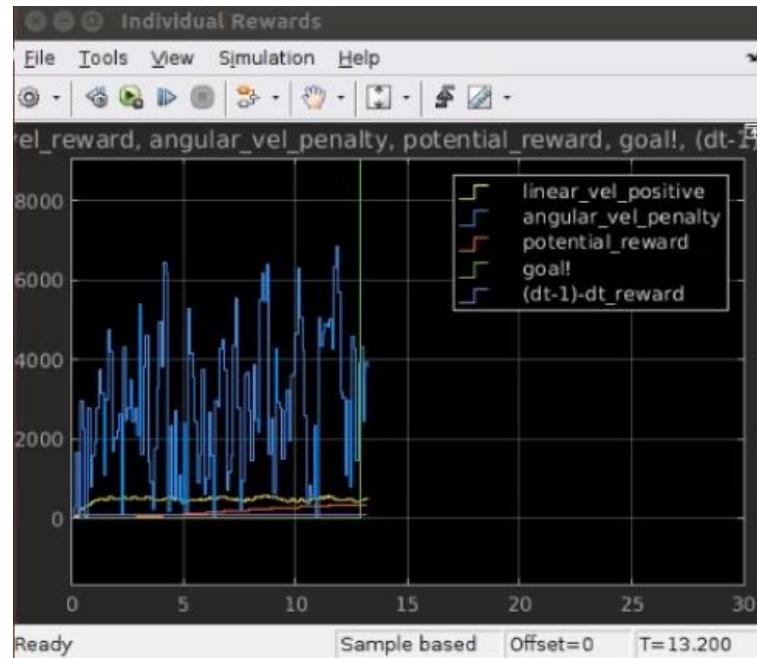


圖 4-11 純追點模擬驗證中機器人每時刻的獎懲值。

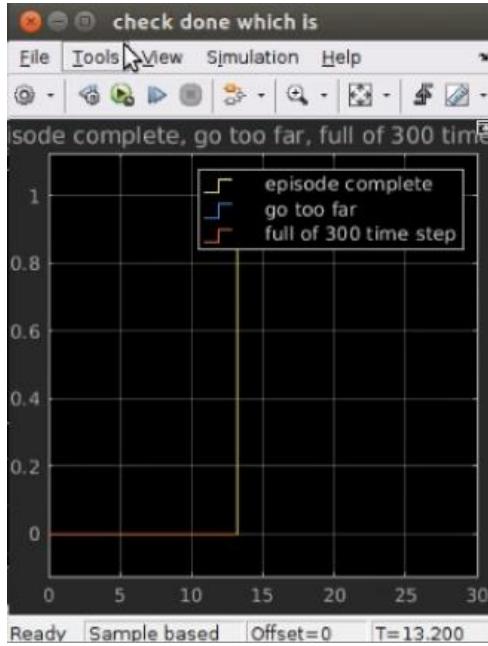


圖 4-12 純追點模擬驗證中機器人每時刻的終止條件狀態。

4-1-2 實作結果

我們先將線速度 v_x 與角速度 ω_z 利用履帶機器人運動學轉向關係，如圖4-1，轉換成左履帶速度 v_x^l 與右履帶速度 v_x^r ，在本文只考慮理想的環境，故利用：

$$a = \frac{v_x}{\omega_z} \quad (4-7)$$

即可得到履帶機器人的旋轉半徑 a ，則再利用：

$$v_x^l = (a - b)\omega_z \quad (4-8)$$

$$v_x^r = (a + b)\omega_z \quad (4-9)$$

即可得到在無打滑環境的轉換關係。

在純追點控制的智能體用於履帶機器人上測試的實驗中，分為室內和室外測試兩部分，在此次實驗中使用的 vSLAM 為 ORB-SLAM [22]。

在室內環境的實驗中，當程序一啟動時，視覺里程計隨即丟失，雖在後續的取樣時間中仍斷斷續續地提供不穩定的位置資訊，但這情況對於智能體來說是致命的，因為在模擬環境的學習中並沒有遇到過接收到在當前時刻位置有值，但在下時刻位置為零的情況，在輸入不穩定且不準確的狀態空間給智能體，會使得智能體喪失其決策能力，最終約在 12 秒時強制終止程序。在圖 4-13 (a) 中黃、藍和橘線分別代表欲輸入至獎懲函數中的，計算出的狀態 $\omega_z(t)$ 、 $v_x(t)$ 與 $v(t)$ 的 y 分量，在動作空間的約束條件皆為 -0.5 至 0.5 m/s 及 rad/s 的情況下卻計算出大於 0.5 的數值，明顯與圖 4-13 (b) 不符，且在圖 4-13 (c) 中所得到的 $d_{goal}(t)$ 及 $\Psi_{goal}(t)$ 都為不連續的狀態。

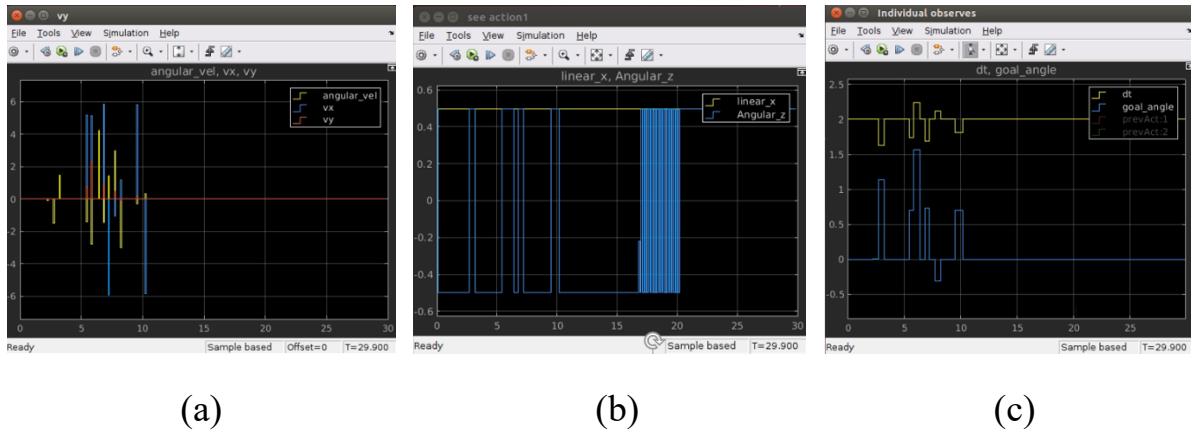


圖 4-13 室內純追點實作驗證中每時刻：(a)計算出的線速度及角速度；(b)智能體輸出的動作空間；(c)智能體觀察到的狀態空間。

室內實驗的失敗，原因初步歸咎於 vSLAM 程序因特徵點不夠密集所造成的運算失敗，導致 VO 的資訊不穩定。

在室外的驗證測試中，我們將地點選在特徵點較多的草地進行，但在程

序啟動後 VO 也是隨即丟失，故判斷失敗的原因可能是因為機器人在此動作空間約束下的速度過快所導致，特徵點追蹤的運算速度不夠快造成的 VO 丟失，最終程序在 19.1 秒時終止。情況與與室內實驗相同，在動作空間的約束條件皆為 -0.5 至 0.5 m/s 及 rad/s 的情況下 4-14 (a)卻計算出大於 0.5 的數值，明顯與圖 4-14 (b)不符，且在圖 4-14 (c)中所得到的 $d_{goal}(t)$ 及 $\Psi_{goal}(t)$ 都為不連續的狀態。

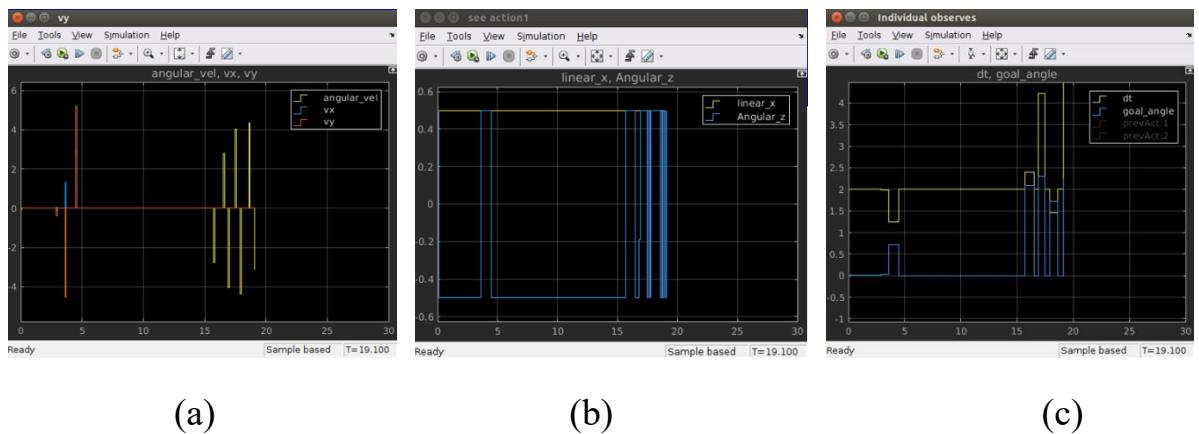


圖 4-14 室外純追點實作驗證中每時刻：(a)計算出的線速度及角速度；(b)智能體輸出的動作空間；(c)智能體觀察到的狀態空間。

4-2 履帶機器人追跡避障

在前一節我們成功地完成了純追點控制的模擬訓練，故保留了其狀態空間以及獎懲函數的設置，用追加的方式延續成功的經驗。在訓練避障控制的部分，我們希望訓練出一個智能體能夠控制機器人完成從起點走到終點，並且完成規避障礙物的行為。在此之前，我們需要先替 YOLO 建立模擬環境中障礙物的資料庫，在這邊我們使用直徑 0.62 m，高度為 2.4 m 的圓柱體

作為障礙物，且固定在地圖中不動，訓練環境如圖 4-15，其中機器人的初始點設置在全局座標系 (X, Y) 中的 $(-9.5, -9.5)$ ，及終點位於 $(-5.5, -5.5)$ 。判斷任務成功或失敗的條件有四個，當機器人距離終點小於 0.5 m 或大於 8.15 m 時，機器人距離障礙物小於 0.1 m 時，及當模擬時間超過取樣區間時，就會重置機器人至初始位置並且將動作空間的值歸零。

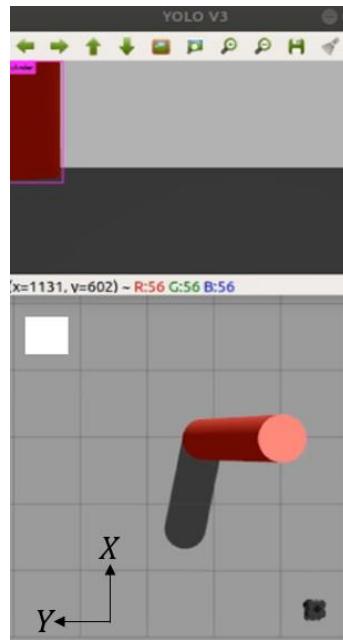


圖 4-15 追點及避障控制 Gazebo 訓練環境。

在訓練規避障礙物控制的第一個版本中，一個 episode 的取樣區間 $T_f = 30$ 、取樣週期 $T_s = 0.1$ ，動作空間為線速度 v_x 以及角速度 ω_z ，約束設置 v_x 為 0.1 至 0.3 (m/s)，而 ω_z 為 -0.2 至 0.2 (rad/s)，將 v_x 的輸出設為恆正的原因是因為對障礙物的感測是由相機來處理，對於機器人的後方完全沒有視野，狀態空間為：

$$s_t = \begin{bmatrix} d_{goal}(t) \\ \Psi_{goal}(t) \\ d_{obstacle}(t) \\ \Psi_{obstacle}(t) \\ t \\ v_x(t-1) \\ \omega_z(t-1) \end{bmatrix} \quad (4-10)$$

其中 $d_{obstacle}(t)$ 為機器人距離障礙物的深度， $\Psi_{obstacle}(t)$ 為機器人正前方與障礙物之間的偏航角角度，範圍因 FOV 的關係，為 -0.194π 至 0.194π 。獎懲函數為：

$$r_t = \begin{cases} 10^3 v_x(t) - 10^4 |\omega_z(t)| \\ 200 \cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} + 200 |if_{d_{goal}(t) < d_{goal}(0)} \\ -200 \cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} - 200 |if_{d_{goal}(t) > d_{goal}(0)} \\ -6 \times e^{0.3t-e^1} \\ 100e^{10(\Delta d_{goal})^2} |if_{\Delta d_{goal} > 0} \\ -100e^{10(\Delta d_{goal})^2} |if_{\Delta d_{goal} < 0} \\ 2 \times 10^5 |if_{d_{goal}(t) < 0.5m} \\ \frac{-10^3}{1+10^{-4}d_{obstacle}^2(t)} |if_{d_{obstacle}(t) < 1m} \end{cases} \quad (4-11)$$

增加第六項障礙物位能懲罰，讓機器人越接近障礙物時，會累積到越大的懲罰值。圖 4-16 為第一版訓練結果。

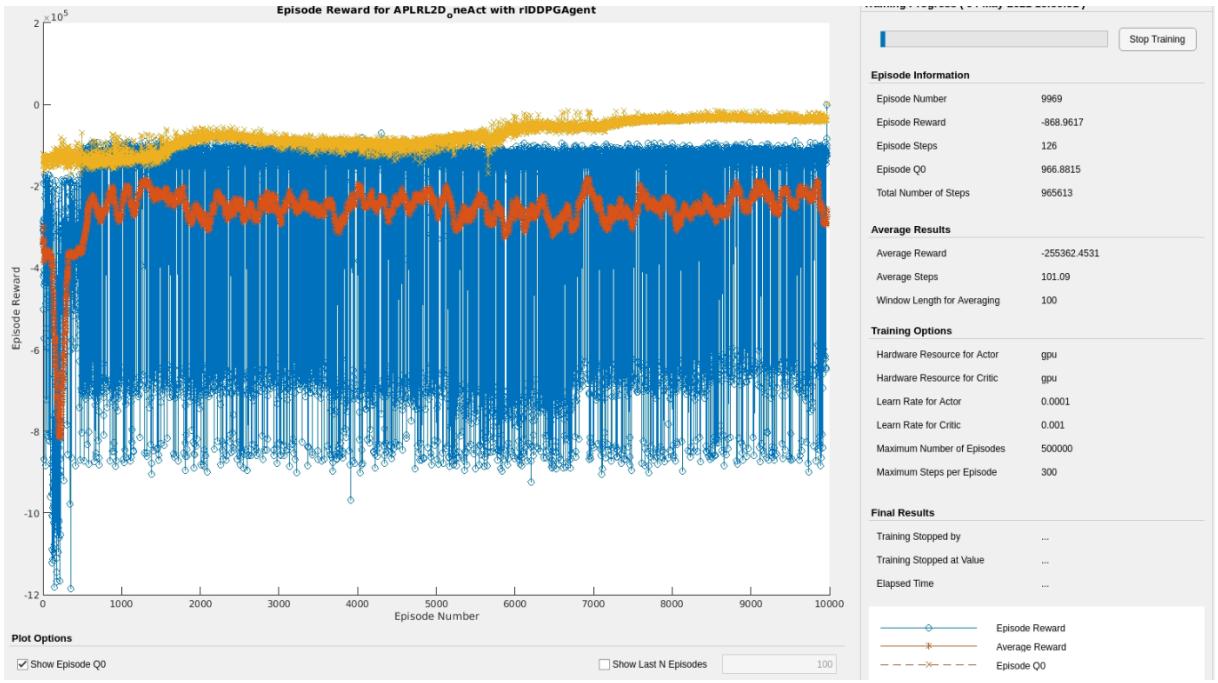


圖 4-16 追點及避障控制第一版訓練統計。

在此一版中的學習是失敗的，在行為上雖有探索行為，但就算訓練了 10^4 個 episode，卻也始終沒有獲得走到終點的獎勵，故猜測智能體並沒有足夠的探索能力。所以在規避障礙物控制的第二版訓練中，為了輔助智能體在探索能力上的不足，在程式上做了修正，首先將固定障礙物的設置改動為在 episode 即將開始時令障礙物的位置在 $5 \times 5 \text{ m}^2$ 的環境中隨機生成，同樣地，終點的位置也改動成在 episode 即將開始時在 $5 \times 5 \text{ m}^2$ 的環境中隨機生成，且 episode 結束後，並不會初始化機器人地位置，只留下智能體輸出一個線速度及角速度為零的動作空間給機器人。圖 4-17 為追點及避障控制第二版的訓練結果。

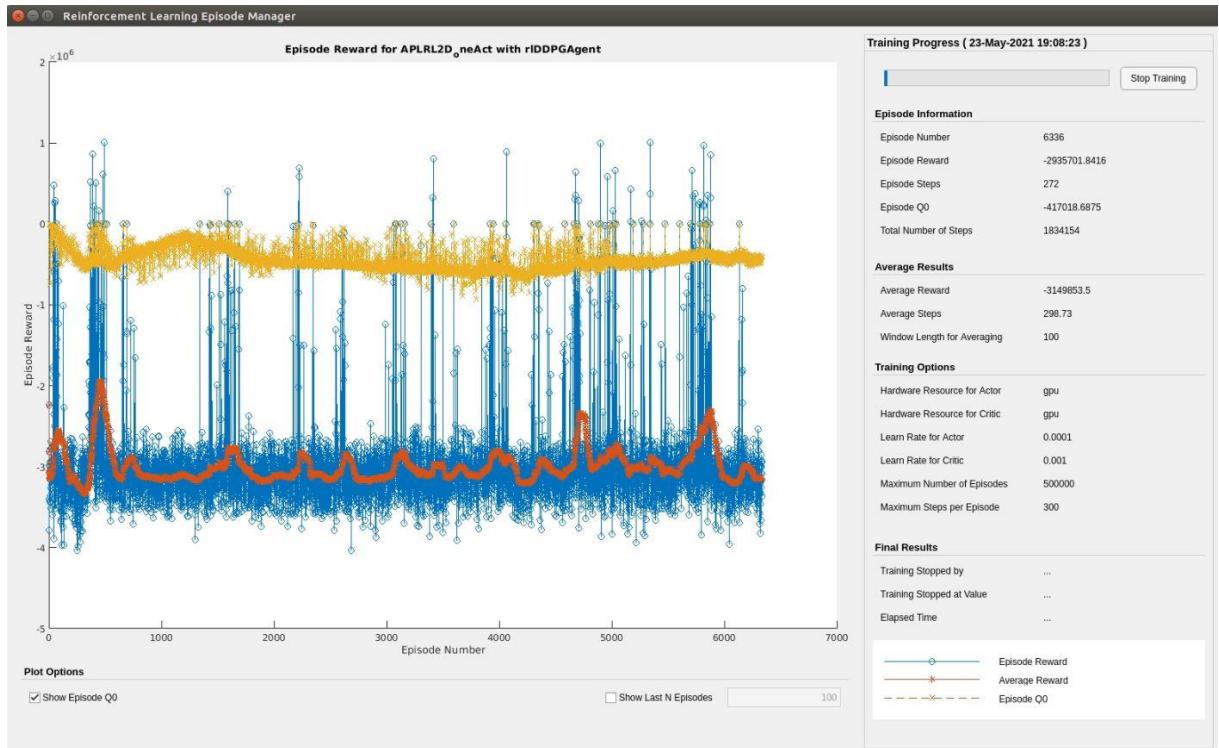


圖 4-17 追點及避障控制第二版訓練統計。

在追點及避障控制第二版中發現，在 6,000 多個 episode 中智能體在線速度約束為 0.1 至 0.3 (m/s)的情況下，在行為上會一直輸出 0.1 m/s 的速度來進行學習，這在 $d_{goal}(0)$ 很大的情況下很不利於探索，所以針對第一項速度獎懲，將線速度 $v_x(t)$ 的權重提高至 10^5 ，並延續使用第二版訓練出來的智能體進行第三版的訓練。圖 4-18 為第三版訓練結果。

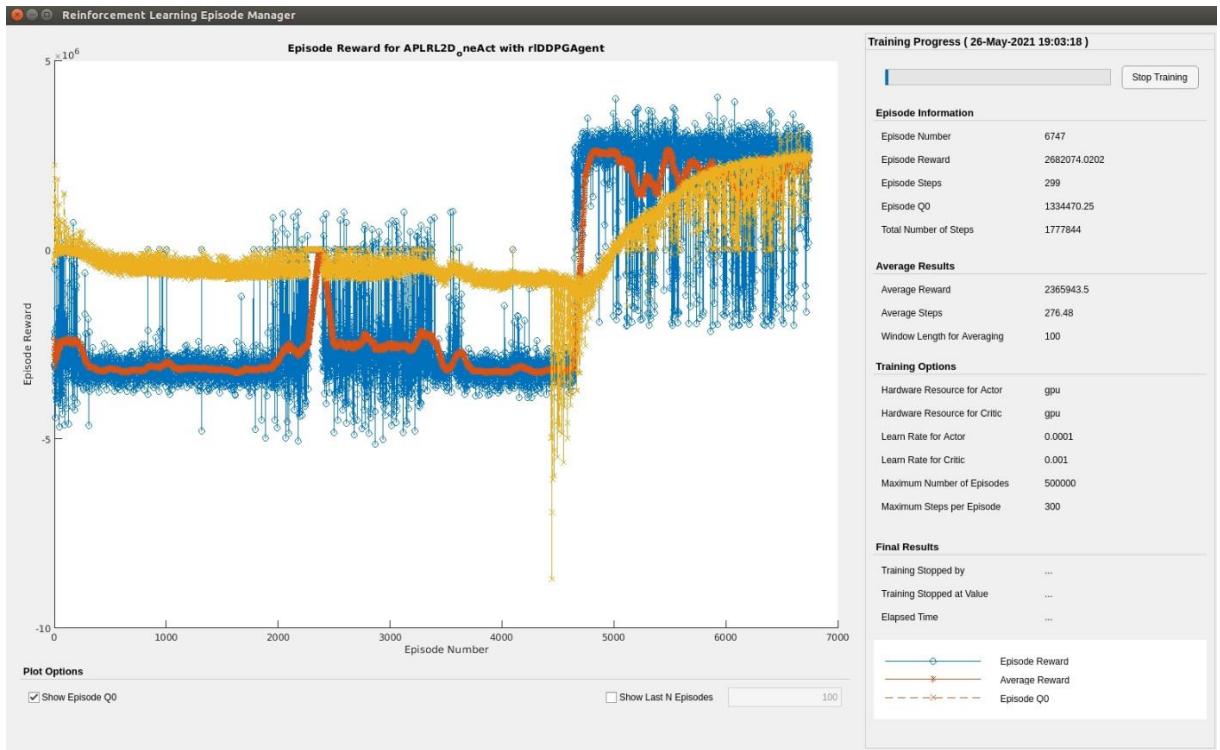


圖 4-18 追點及避障控制第三版訓練統計。

第三版訓練中，在四千多 episode 後成功使線速度不再處於 0.1 m/s，並且提高探索效率。在第四版中，我們針對獎懲函數繼續修正，降低第一項速度獎懲的線速度權重值，再提高第二項位能獎懲、第四項變化量獎懲與第五項抵達終點獎勵的權重，並且增加第七項碰撞發生懲罰，並延續使用第三版訓練出來的智能體進行第四版的訓練。圖 4-19 為第四版訓練結果。

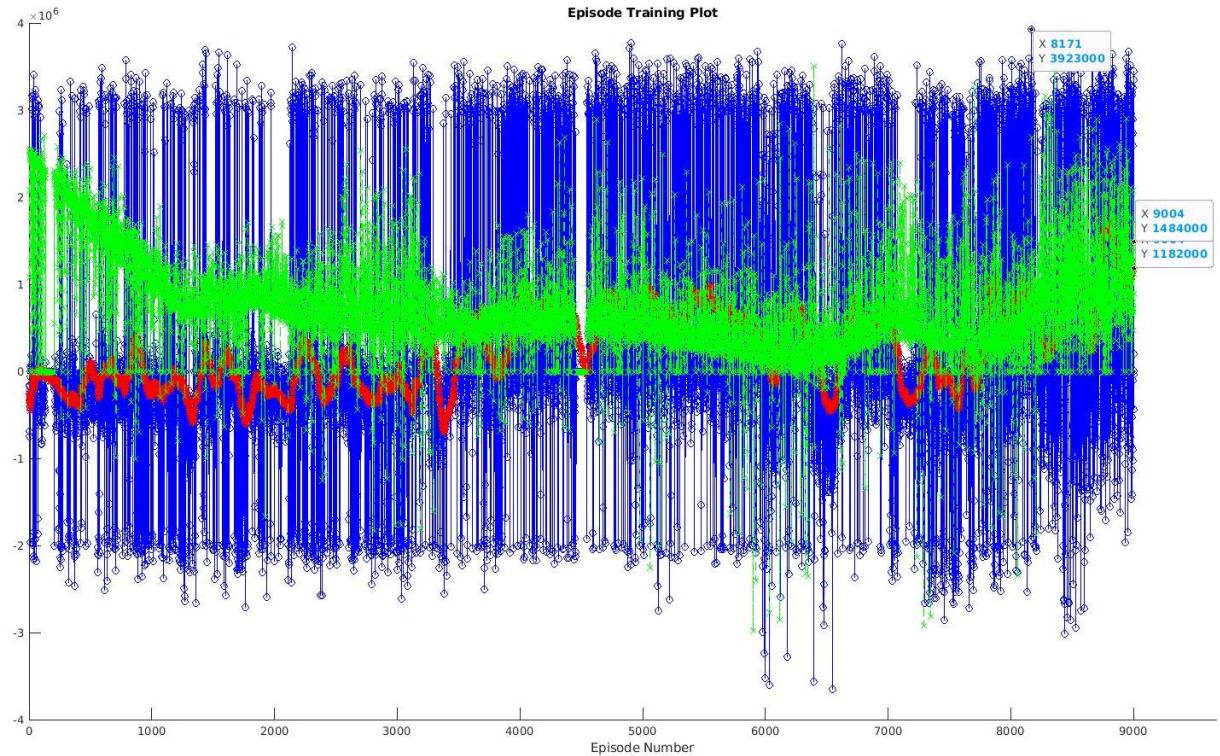


圖 4-19 追點及避障控制第四版訓練統計。

最終在第四版中，完成了此次追點及避障控制訓練，也因為第二版至第四版的環境和訓練機制都是相同的，只針對獎懲函數作微調，用獎勵引導的方式引導智能體朝向我們所期望的行為前進，且狀態空間使用一致，所以我們將智能體從第二版開始，用遷移式學習的方式，最終經過三版的訓練，總共歷時 10 天，訓練了 22,087 個 episode 後完成訓練。第四版的獎懲函數如下：

$$r_t = \begin{cases} 66666v_x(t) - 10^4|\omega_z(t)| \\ 800\cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} + 800|if_{d_{goal}(t) < d_{goal}(0)} \\ -800\cos\{0.5[d_{goal}(t) - d_{goal}(0)] - \pi\} - 800|if_{d_{goal}(t) > d_{goal}(0)} \\ -6 \times e^{0.3t-e^1} \\ 1000e^{10(\Delta d_{goal})^2}|if_{\Delta d_{goal} > 0} \\ -1000e^{10(\Delta d_{goal})^2}|if_{\Delta d_{goal} < 0} \\ 3 \times 10^6|if_{d_{goal}(t) < 0.5m} \\ \frac{-10^3}{1+10^{-4}d_{obstacle}^2(t)}|if_{d_{obstacle}(t) < 1m} \\ -1 \times 10^6|if_{d_{obstacle}(t) < 0.1m} \end{cases} \quad (4-12)$$

4-2-1 模擬測試

在模擬驗證中，我們將起點設置在全局座標系(X, Y)中的 $(-9.5, -9.5)$ ，終點位於 $(-5.5, -5.5)$ ，障礙物位於 $(-7.0, -8.0)$ ，圖 4-21 顯示經歷 221 個取樣時間後抵達終點，環境如圖 4-20 所示。

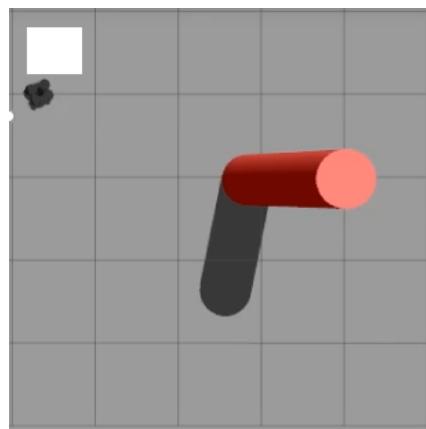


圖 4-20 追點及避障模擬驗證中機器人避開障礙物從起點走向終點。

在圖 4-21 (a)黃線表示狀態空間中的 $d_{goal}(t)$ ，圖 4-21 (b)綠線顯示最終得到到達終點的獎勵，圖 4-21 (c)黃線顯示最終終止條件為任務完成。

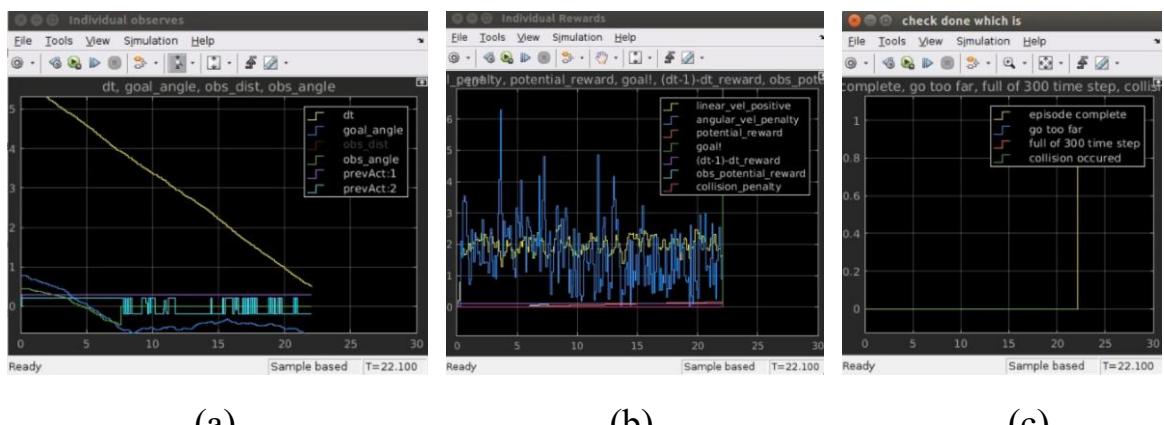


圖 4-21 成功的追點及避障模擬驗證中每時刻：(a)智能體觀察到的狀態；(b)智能體獲得的獎懲值；(c)終止條件狀態。

但並不是每一次的模擬測試都會成功，智能體有約 20% 的機率會使機器人與障礙物發生碰撞，如圖 4-22 所示。

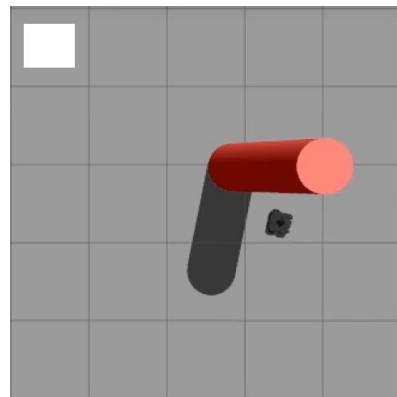


圖 4-22 追點及避障模擬驗證中機器人避障失敗。

在圖 4-23 (b)紅線顯示最終發生碰撞的懲罰值，圖 4-23 (c)綠線顯示最終終止條件為發生碰撞。

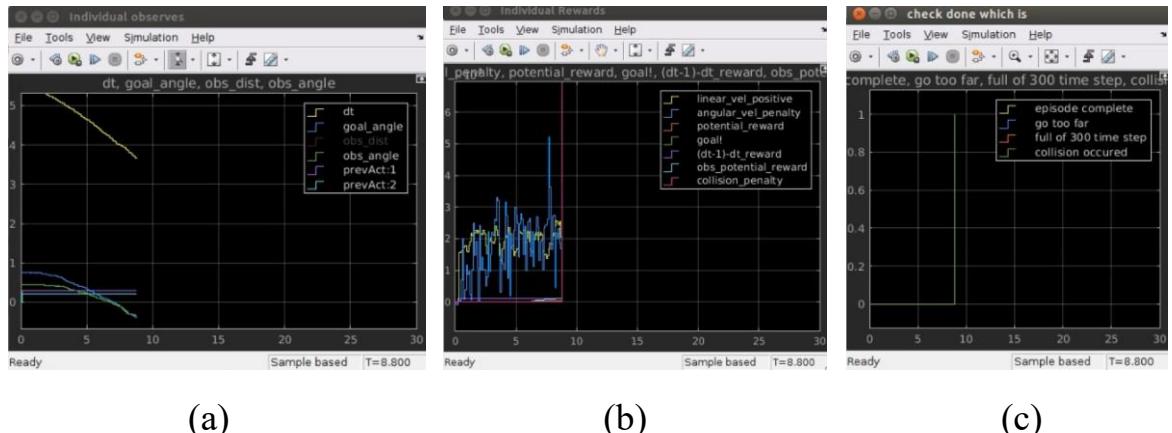


圖 4-23 失敗的追點及避障模擬驗證中每時刻：(a)智能體觀察到的狀態；(b)智能體獲得的獎懲值；(c)終止條件狀態。

4-2-2 實作結果

前面 4-1 節中提到不穩定的 VO 資訊會造成智能體失控，在追點及避障控制的智能體用於履帶機器人上測試的實驗中，使用兩種不一樣的 vSLAM，ORB-SLAM 以及 RTAB-MAP 分成兩部分進行測試。現實中的場域設定在田間，如圖 4-24，我們將起點設置在全局座標系 (X, Y) 中的 $(0.0, 0.0)$ ，終點用紅色椅子表示，位於 $(5.0, 5.0)$ ，障礙物為白色檣版，位於 $(2.5, 1.5)$ ，障礙物辨識情況如圖 4-25。



圖 4-24 田間場域驗證布置。

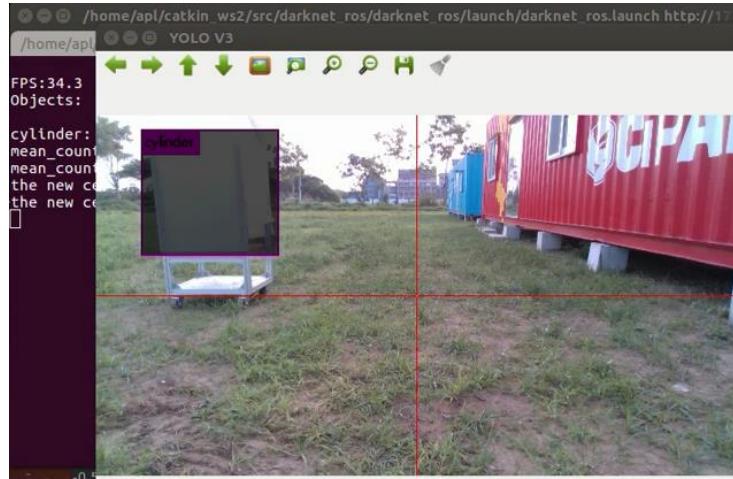


圖 4-25 田間障礙物。

在 ORB-SLAM 的實驗中實驗是失敗的，原因也是如 4-1-2 節的實驗一般丟失 VO 資訊，在運行過程中，總共約有 200 個取樣時間沒有收到來自 VO 的資訊，如圖 4-26(a)中約第 5 秒至約第 20 秒，以及第 25 秒至第 30 秒，這 20 秒中的黃線 $d_{goal}(t)$ 正常來說就算靜止不動也應該要有浮動值，所以判斷這範圍內的位置資訊為丟失的情況。

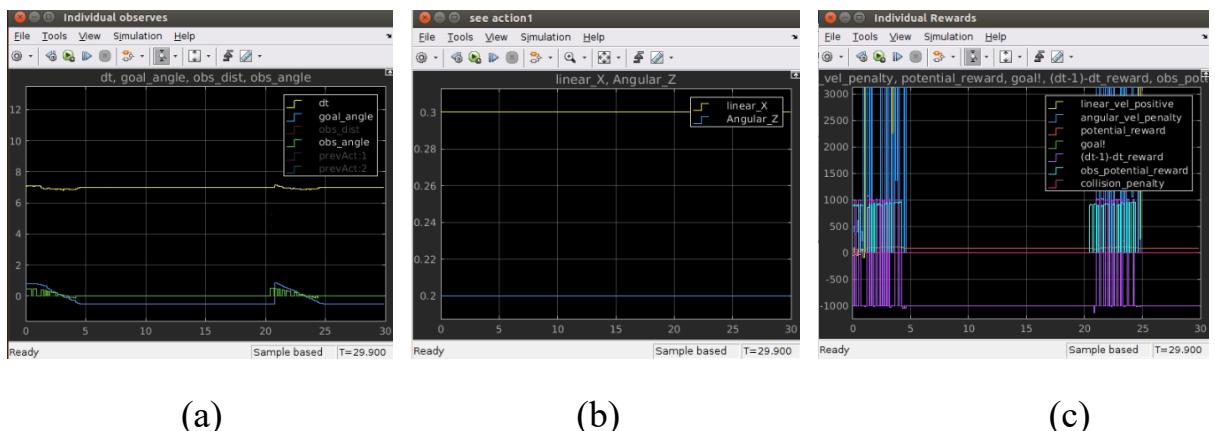


圖 4-26 失敗的 ORB-SLAM 追點及避障實作中智能體每時刻：(a)觀察到的狀態空間；(b)輸出的動作空間；(c)獲得的獎懲值。

在 RTAB-MAP 實驗中實驗是成功的，如圖 4-27 顯示經歷 245 個取樣

時間後成功抵達終點，在圖 4-27 (a)黃線表示狀態空間中的 $d_{goal}(t)$ 的距離成功小於 0.5 m，圖 4-27 (b)黃線顯示最終終止條件為任務完成，圖 4-27 (c)綠線顯示最終得到到達終點的獎勵。從資料的連續性可以看出其 VO 的穩定性較 ORB-SLAM 好很多，智能體接收的狀態空間及獎懲值，如圖 4-27 (a)與圖 4-27 (c)，資料走勢也比較貼近於如圖 4-21 所示之模擬驗證的情形。

而其中值得討論的是圖 4-27 (c)，在程序一開始至結束的過程中是機器人往終點靠近的趨勢，按照當初的變化量獎懲設計，獲取的獎懲值應該為正，而不應該是正負相間的，這顯示 RTAB-MAP 的 VO 精度不足以支撐我所設計的約束條件下，每個取樣時間之間的最小位置變化量，但獎懲值不穩定的問題並不會對訓練完的智能體造成太大的影響，真正影響的是正在訓練中的智能體，原因是訓練完的智能體是以單純的 actor 網路 μ 作為控制器與系統模型，作用如圖 3-4 的 DDPG 架構中下方的黑線流程。

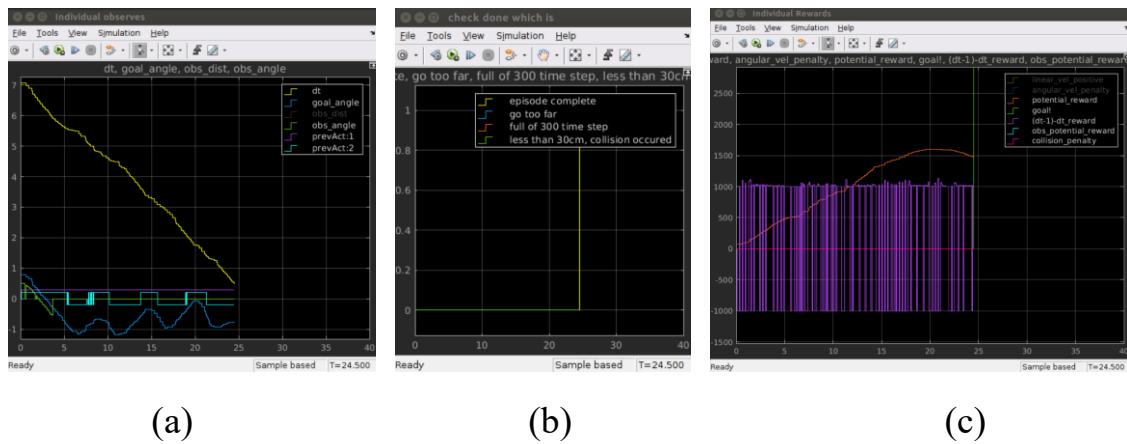


圖 4-27 成功的 RTAB-MAP 追點及避障實作中智能體每時刻：(a)觀察到的狀態空間；(b)終止條件狀態；(c)獲得的獎懲值(zoom in)。

但並不是每一次的實驗測試都會成功，智能體有約 40% 的機率會失敗，如圖 4-28 所示。圖 4-28 (b)的橘線表示在走完 30 秒的取樣區間後，雖然沒有發生機器人與障礙物之間的碰撞，但也沒有成功走到終點完成任務。

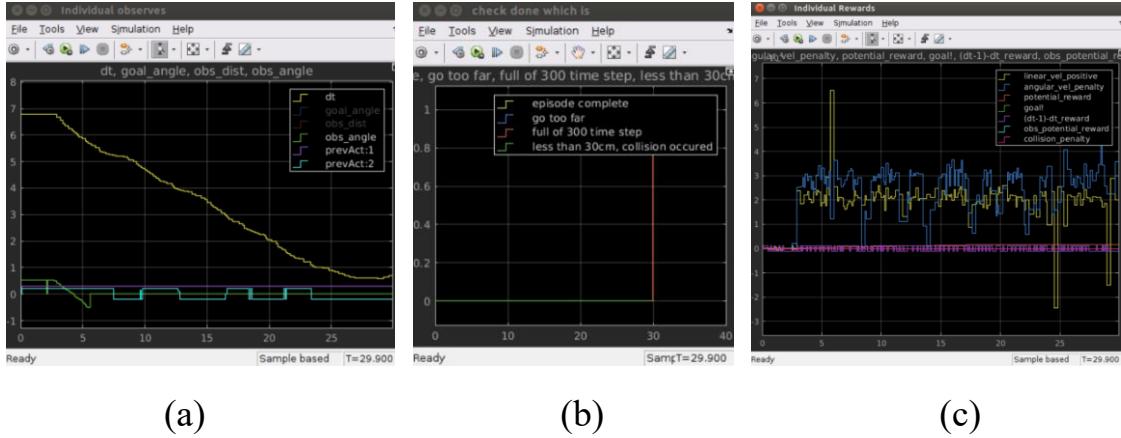


圖 4-28 失敗的 RTAB-MAP 追點及避障實作中智能體每時刻：(a)觀察到的狀態空間；(b)終止條件狀態；(c)獲得的獎懲值。

兩部分的實驗僅有使用 RTAB-Map 作為 vSLAM 的實驗才能讓智能體操控著履帶機器人在田間達成追跡避障的控制行為。主要原因來自於 ORB-SLAM 的 VO 無法提供有效資訊且與模擬環境中穩定的數據提供落差太大。

第五章 結論與未來工作

DDPG 演算法在避障問題上，不用先行對系統環境(plant)進行建模，透過強化學習的方式就能夠對難以數學化的複雜系統進行控制，主要是利用動態規劃的方式於每一次的 time-step 中尋找最優狀態動作價值函數 $Q(s_i, a_i)$ 來逐漸影響 actor 的決策，但因為神經網路裡的參數在一開始都是隨機的，在學習智能體的過程中只能透過不斷地使用試誤法(trial and error)與環境互動後所得到的狀態空間以及獎懲，來學習出神經網路裡的權重值，就算獎懲函數設計得很好，也需要累積足夠的經驗，學習才會慢慢地朝獎懲函數設計的理念前進，且為了讓學習的過程能夠穩定地收斂，會在新權重乘上一個折扣因子後才加上原本的權重。

在 Gazebo 驗證的結果上，純追點控制、追點及規避障礙物控制都是成功的，且起點與終點直線距離皆為 5.656 m，純追點任務在動作空間約束為線速度 v_x 在 -0.5 至 0.5 m/s 以及角速度 ω_z 在 -0.5 至 0.5 rad/s 下，約花費 13.2 秒達成任務，成功率為 100%；追點及規避障礙物任務在動作空間約束為線速度 v_x 在 0.1 至 0.3 m/s 以及角速度 ω_z 在 -0.2 至 0.2 rad/s 下，約花費 22.1 秒達成任務，成功率約為 80%。

在田間中的履帶機器人驗證結果上，純追點控制、追點及規避障礙物控制在 ORB-SLAM 上，都沒有如模擬結果般成功地執行任務，但在 RTAB-Map 上的追點及規避障礙物控制是成功的，起點與終點直線距離為 7.07 m，

在動作空間約束為線速度 v_x 在 0.1 至 0.3 m/s 以及角速度 ω_z 在 -0.2 至 0.2 rad/s 下，約花費 24.5 秒達成任務，成功率約為 60%。主要是因為模擬環境與現實環境的機器人與環境差異太大，且 vSLAM 在 VO 的測量精度與穩定性也不如模擬環境。

在未來會於 Gazebo 中導入更符合現實情況的環境進行學習，如凹凸不平的地面、按照實驗室的田間履帶機器人所建構出的模擬機器人，以及量測到的履帶機器人系統雜訊。在現實中 VO 資訊的測量精度與穩定性上，並嘗試使用卡爾曼濾波器(Kalman filter) [23]進行多感測器的融合，加入如慣性里程計 IO (inertial odometry) 以及 RTK-GPS (real-time kinematic) [24]等，來輔助量測。也會嘗試讓 deep RL 結合至不同的任務，例如藉由其能夠對系統建模的能力，與其他控制器融合 [20,25]；在機器人導航上基於參考路徑來生成輔助性質的局部路徑[26]；將 DDPG 應用於 MPC 控制系統中，經由兩個控制器進行平行優化控制的方式來引導智能體學習，且 actor 作為額外的補償控制器，可以用於彌補 MPC 樣本效率差的問題[27]。

參考文獻

- [1] M. Labb   and F. Michaud, “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics* 36, 416- 446 (2018).
- [2] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural Networks* 6, 861-867 (1993).
- [3] L. Baird, “Residual algorithms: Reinforcement learning with function approximation,” *Machine Learning: Proceedings of the Twelfth International Conference*, pages 30-37 (1995).
- [4] E. Hernandaz and Y. Arkun, “Neural network modeling and an extended DMC algorithm to control nonlinear systems,” *American Control Conference*, 2454-2459 (1990).
- [5] I.-M. Chen and C.-Y. Chan, “Deep reinforcement learning based path tracking controller for autonomous vehicle,” *Proc. IMechE, Part D: Journal of Automobile Engineering*. (2020). DOI: 10.1177/0954407020954591.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 779-788 (2016).
- [7] X. Gao, T. Zhang, and Y. Liu, *14 Lessons of Visual SLAM: From Theory to Practice*[M]. Publishing House of Electronics Industry, Beijing (2017).
- [8] D. Nist  , O. Naroditsky, and J. R. Bergen, “Visual odometry,” *Intl. Conference on Computer Vision and Pattern Recognition*, 652-659 (2005).
- [9] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” arXiv

preprint arXiv:1804.02767 (2018).

- [10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (2nd Edition), MIT Press (2017).
- [11] C. J. C. H. Watkins, *Learning from Delayed Rewards*, PhD thesis, University of Cambridge England (1989).
- [12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature* 518, 529-533 (2015).
- [13] M. L. Puterman, *Markov Decision Processes Discrete Stochastic Dynamic Programming*, John Wiley & Sons, Inc., New York (1994).
- [14] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks* 21, 682-697 (2008).
- [15] T. Lillicrap, J. Hunt, *et al.*, “Continuous control with deep reinforcement learning,” *International Conference on Learning*, (2016).
- [16] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” *International Conference on Machine Learning (ICML)*, (2014).
- [17] MathWorks Student Competitions Team (2021). MATLAB and Simulink Robotics Arena: Walking Robot (<https://github.com/mathworks-robotics/msra-walking-robot>), GitHub. Retrieved June 23, 2021.
- [18] <https://github.com/ROBOTIS-GIT/turtlebot3>
- [19] E. F. Camacho and C. B. Alba, *Model Predictive Control (2nd Edition)*. Springer Science & Business Media, (2013).
- [20] A. Nagabandi, *et al.*, “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning,” *International*

Conference on Robotics and Automation, (2018).

- [21] H. Lu, G. Xiong, and K. Guo, “Motion predicting of autonomous tracked vehicles with online slip model identification,” *Mathematical Problems in Engineering*, 1-13 (2016).
- [22] R. Mur-Artal, J. Montiel, and J. Tardos, “ORB-SLAM: A versatile and accurate monocular SLAM system,” *IEEE Transactions on Robotics* 31, 1147-1163 (2015).
- [23] K. Reif, S. Gunther, E. Yaz, and R. Unbehauen, “Stochastic stability of the discrete-time extended Kalman filter,” *IEEE Transactions on Automatic Control* 44, 714-728 (1999).
- [24] Y. Feng and J. Wang, “GPS RTK performance characteristics and analysis,” *Journal of Global Positioning Systems* 7, 1-8 (2008).
- [25] K. Chua, R. Calandra, *et al.*, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” arXiv:1805.12114, (2018).
- [26] B. Evans, H. W. Jordaan, and H. A. Engelbrecht, “Autonomous obstacle avoidance by learning policies for reference modification,” arXiv:2102.11042, (2021).
- [27] H. Xie, X. Xu, Y. Li, W. Hong, and J. Shi, “Model predictive control guided reinforcement learning control scheme,” *2020 International Joint Conference on Neural Networks*, 1-8 (2020).