中華民國第 59 屆中小學科學展覽會

作品說明書

高級中等學校組　工程學(一)科

第三名

052311

看得到，拿得到 ── 機械手臂與電腦視覺的結合

學校名稱：臺中市立臺中第一高級中等學校

作者：	指導老師：
高二 鄭宇皓 高二 賴繼堯 高二 林冠曄	林奇鋒

關鍵詞：機械手臂、電腦視覺、ROS

摘要

本研究使用自製的機械手臂並透過 ROS 與 MoveIt 實現機械手臂的路徑規劃，且給予機械手臂自身的視覺系統，並透過深度學習進行物體辨識並使用深度攝影機獲取物體的距離，使機械手臂能夠自行移動到目標物的位置並抓取物體。將電腦視覺、深度學習與機械手臂結合，機械手臂便可以達到自動化，而無須人為操作。本研究所開發的機械手臂系統為完全開源、價格低廉且軟體方面可套用於其他安裝 ROS 的系統中，十分適合與其他領域結合。而日後在更多自動化的系統上，如物流系統的自動揀貨、行動不方便的長者的輔助系統或是自動化垃圾分類等，使用本系統的電腦視覺、機器人系統，可以讓生活中一些需要人力的事情變得更快更方便。

壹、研究動機

在這個科技快速發展，機器逐漸取代人工的時代，機械手臂在工業中被大幅應用，而上次在大學的實驗室參觀計畫中，電機工程系實驗室的教授及研究生像我們示範如何操控機械手臂及機器人運動學的相關理論，在他們的操作下，機械手臂可以流暢地移動並抓取物品，靈活的轉動、控制使我們看得目不轉睛。

回到學校之後，我們也想要自己製作一個機械手臂，在給定目標物座標的情況下順利抓取物體，再結合電腦視覺，達到自動化的效果。

貳、研究目的

根據我們的動機，經過討論後訂定出幾個研究目的：

一、設計並組裝出多自由度且具開發空間的機械手臂。

二、設計並編寫出機械手臂計算及運動的軟體。

三、編寫深度攝影機成像及物件辨識的軟體。

四、結合電腦視覺讓實體機械手臂達到完全自動化的效果。

參、研究設備及器材

一、硬體

(一) 電腦(AERO 15)

1. CPU：Intel Core I7-8750H
2. GPU：Nvidia Geforce GTX1070Max-q
3. Ram：16GB DDR4

(二) 自組機械手臂

其中包括伺服馬達(原為 MG996R 伺服馬達，後來更換為 DSSERVO 的 DS3218、DS3225 和 DS3230)、底座、馬達支撐架、法蘭桿、U 型馬達支架、L 型支架、U 型支架、法蘭杯士、軸承、舵盤。

(三) 單片機 KSB026(為 Arduino Micro 和伺服馬達控制器透過 I2C 通訊的組合)

把許多訊號發收器和微處理器集合到一個晶片上，相當於一種微型控制器。

(四) 深度攝影機 Kinect V1(後來更換為 Intel® RealSense™ D435)

可以測量每個物體和攝影機的 Z 軸距離，因此儲存的是三維的空間資訊。

(五) 電源供應器(5V12A)

將 110V 交流電轉換成 5V、12A 的直流電。

二、軟體

(一) Ubuntu 作業系統

為 Linux 發行的桌面環境，主要功能為提供用戶穩定且多自由軟體的開發環境。

(二) Solidworks

1. Solidworks 採用參數化的特徵為基礎建立模型，「特徵」是一個能被獨立修改的最小建造方塊，「參數式」則是「能夠使用標準的屬性或參數去定義一個幾何物件的形狀和尺寸」。其中有三種模式，分別是零件(Part mode)、組合件(Assembly mode)和工程圖(Drawing mode)，其中“零件”模式中又包括草圖設計、零件設計等，應用於本研究之建模過程。

   （1） 零件

   主要是用實體建模可以用拉伸、旋轉、鏡像等方式對特徵和草圖的修改

   （2） 組合件

可以動態地觀察整個組合件中的所有運動，並且可以對運動的零件進行動態的檢查及檢測。

2. 在建立一個模型時，通常開始於一個 2D 草圖，草圖由幾何，如點，線，弧等等組成，再添加尺寸到來定義的大小和位置，完成草圖以後，再延伸成 3D 圖形。

(三) MoveIt

針對機械手臂操作的一個十分便利的開發平台，結合了運動規劃及運動學，其功能有效的取代了大部分繁雜的程式。

(四) Arduino IDE

控制 Arduino 板的軟件，主要支援 C 跟 C++語言。

(五) ROS(Robot Operating System)

本系統的開發環境，主要支援 C++ 及 Python 兩種程式語言，其執行的程式之間可以很容易地互相溝通及傳輸資料。而現今網路上已有很多其他人發展出現成的 package，因此我們並不需要為研究中每一部分重新開發程式，只需依據素材修改參數。

(六) Roboware

ROS 專用開發工具，給予模型和程式連接運行的環境，簡單強大的功能適合初學者快速創建資源。

肆、研究過程及方法

一、開發環境及工具

(一) ROS (Robot Operating System)

在本研究中，機械手臂使用 ROS 系統來進行開發，此系統的特點在於其由節點(node)、發布者(Publisher)和訂閱者(Subscriber)之間的傳訊(Topic)和 Package (程式包)所構成的結構，這樣 P2P 的通信方式不僅大量提高了單個程式碼的使用率，當機器人需要進行大量運算如電腦視覺處理時也能透過其點對點的通信方式分散運算壓力給其他伺服器。且其相容多種語言(Python, C++, Java, etc)的特性也讓我們組內每個人能使用他們最熟悉的語言進行編寫。

(二) Rviz

Rviz 為 ROS 系統中一款相當強大的圖形化工具，提供我們做為模擬機械手臂和監看規劃場景的主要工具。

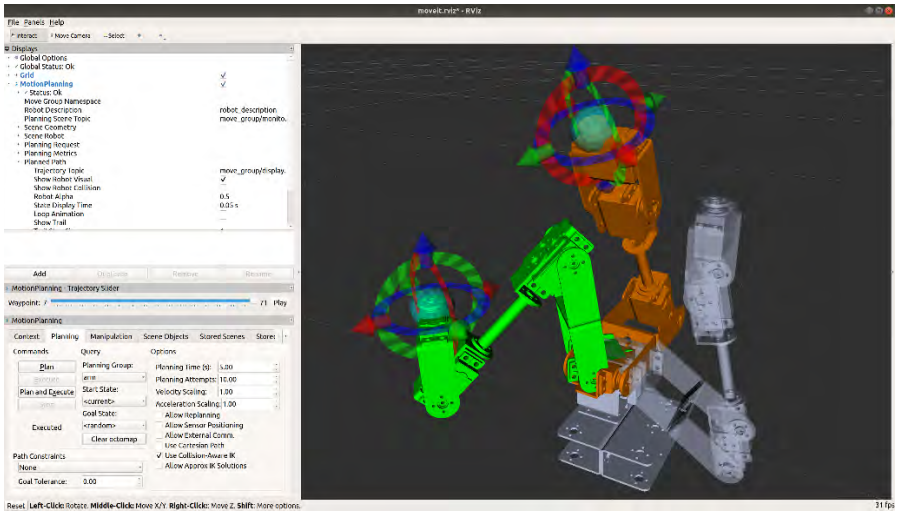


圖 1(Rviz 模擬機械手臂場景)

(三) Roboware Studio

Roboware Studio 是 ROS 的一款 IDE(編譯器)，也可以說是專門為 ROS 設計 Visual studio core，其介面簡潔、功能強大，且支援中文。

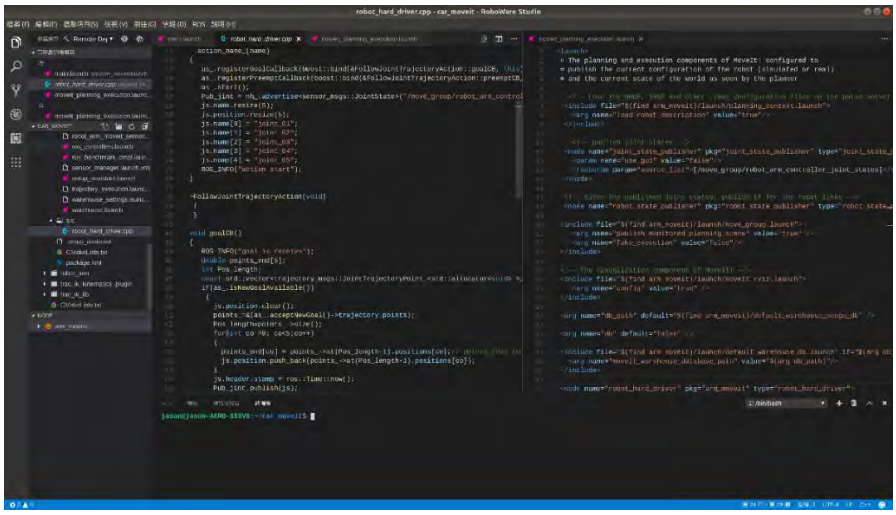


圖 2(Roboware 作業系統)

(四) MoveIt

為了實現機械手臂的抓取和路徑規劃，而當中包含了正逆向運動學、碰撞檢測、環境感知和手臂動作規劃。因此本研究使用 ROS 系統當中的 MoveIt 運動規劃庫完成機械手臂系統層的運動規劃。

MoveIt 是 ROS 系統中整合許多與移動和路徑規劃相關組合包的運動規劃庫，以 move_group 為核心，它提供了大部分機器人路徑規劃所需的功能，並提供相當友好的開發環境和介面便於機器人的初始化和配置。

以下為 MoveIt 的主架構圖。

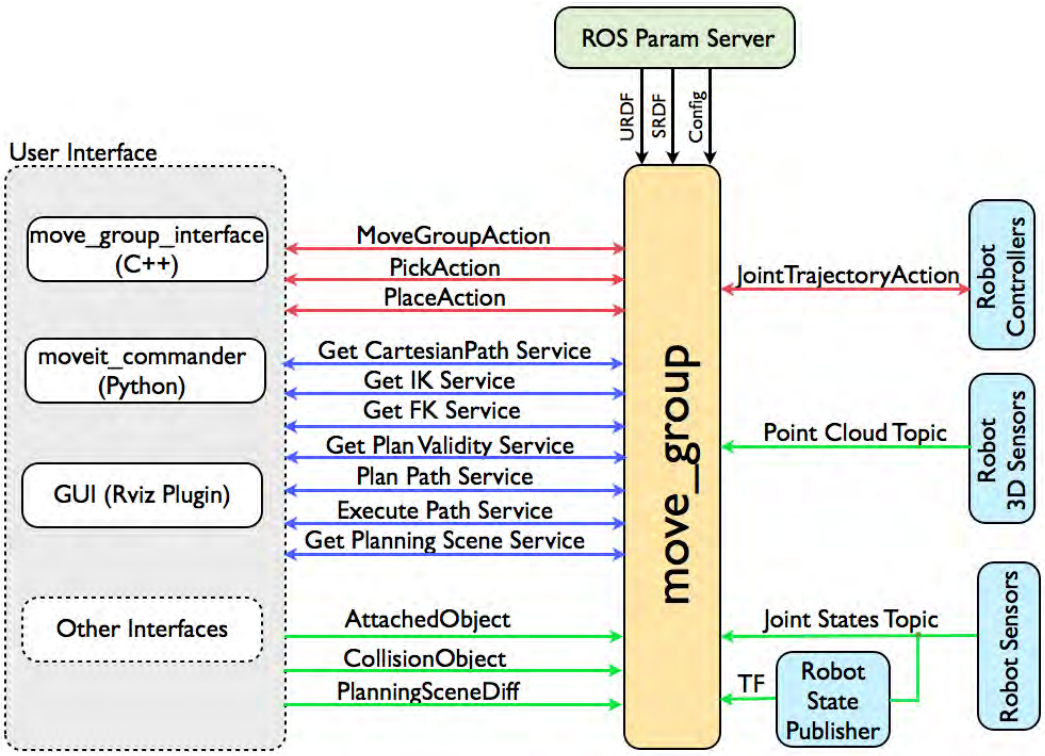


圖 3(MoveIt 運動規劃庫圖示)

二、理論基礎

(一) 正向運動學與逆向運動學(Forward Kinematics(FK) and Inverse Kinematics(IK))

Kinematics 定義：討論機器人所有的動作可能性，包括其臂長、關節的極限、方向的變動等等。

我們先把機械手臂數據化，以利計算：定義出 D-H model 做為各個構件之間的數學關係(a, d, θ, α)。

a 沿 X 軸之位移。

d 沿 Z 軸之位移

θ 繞 Z 軸之角度

α 繞 X 軸之角度

而六軸機械手臂總共會有六組數據，舉例如下：

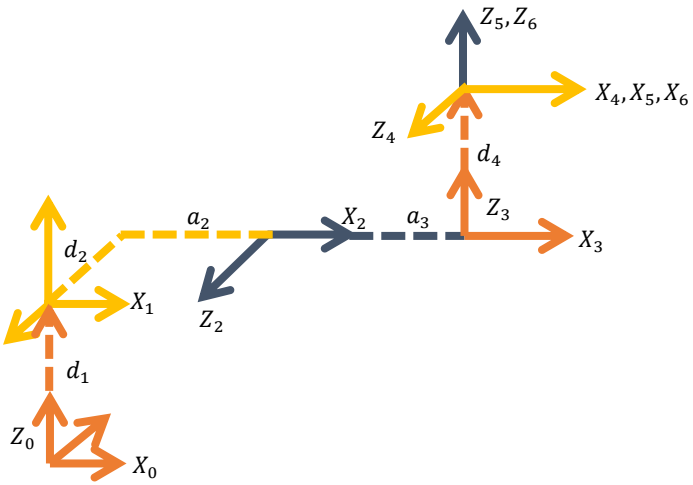


圖 4(PUMA 560 手臂檢視圖)

joint	θ	d	a	α
1	θ1	D1	0	90
2	θ2	D2	A2	0
3	θ3	0	A3	-90
4	θ4	D4	0	90
5	θ5	0	0	-90
6	θ6	0	0	0

表 1(PUMA 560 自由度數據)

而我們將每個構件寫成一組 4x4 矩陣，舉例如下：

$$A_1=\begin{pmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_2=\begin{pmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3=\begin{pmatrix} C_3 & 0 & -S_3 & a_3C_3 \\ S_3 & 0 & C_3 & a_3S_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_4=\begin{pmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5=\begin{pmatrix} C_5 & 0 & -S_5 & 0 \\ S_5 & 0 & C_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_6=\begin{pmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

圖 5(6 構件之矩陣)

此六個矩陣即為機械手臂數據化的基礎要件，現在來說明 FK 跟 IK。

FK(Forward Kinematics)：當前一個節點旋轉或運動時，下一個節點會受到影響。

即將基準點 $\begin{matrix} x \\ y \\ z \\ w \end{matrix}$ *A1*A2*A3*A4*A5*A6 = $\begin{matrix} x' \\ y' \\ z' \\ w' \end{matrix}$ = T

而 T 矩陣即是所求的末端點。

IK(Inverse Kinematics)：互相連結的物件中，末端點移動時，整體的節點會被牽動的行為。

IK 通常會有多組解，因此在考慮關節極限的狀況下，剔除不可行的選項，剩下的解即為關節的可設定值。利用反矩陣和反三角函數進行一系列運算後可得出答案，但在網路之下發現有許多更加有效率且穩定的方法。

KDL IK：目前最常見的 IK solver，同時為 MoveIt 中的默認 IK solver。能夠在封閉情況下得到數值解，但是速度慢且容易發生找不到解的情況。

IK Fast：OpenRave 所提供的逆運動學求解器，與大多數逆運動學求解器不同，IK Fast 可以解析任何複雜運動鏈的運動學方程，將旋轉及位移分別計算，並生成特定於語言的文件（如 C ++）供以後使用。穩定且速度快，且每次解具有一次性。

Trac IK：與 KDL 比較，Trac IK 更加地快速且精確。在求解時，同時運行

兩個 IK 方法： simple extension to KDL's Newton-based convergence algorithm(牛頓收斂算法的擴張型態)以及 SQP(順序二次規劃)，前者利用隨機檢測減輕因為關節限制引起的局部最小值，後者係利用非線性優化來處理關節限制。當這些算法中的任何一個收斂到答案時，IK 搜索會立即返回。還提供了距離和可操縱性的次要約束，以便接收“最佳”IK 解決方案。但每次解不一定相同。

(二) Pinhole Camera(針孔相機座標成像原理)

在使用針孔相機成像模型中，需討論四個座標系：

世界坐標系(3D)：三維現實世界的絕對座標系。$(x_w, y_w, z_w)(m)$

相機坐標系(3D)：以相機焦點為原點的座標系。$(x_c, y_c, z_c)(m)$

圖像坐標系(2D)：以 CCD 圖像平面中心為原點的座標系。$(x, y)(m)$

像素坐標係(2D)：以 CCD 圖像平面左上角頂點為中心的座標系，同時以數值表示每個像素的灰階。$(u, v)(pixel)$

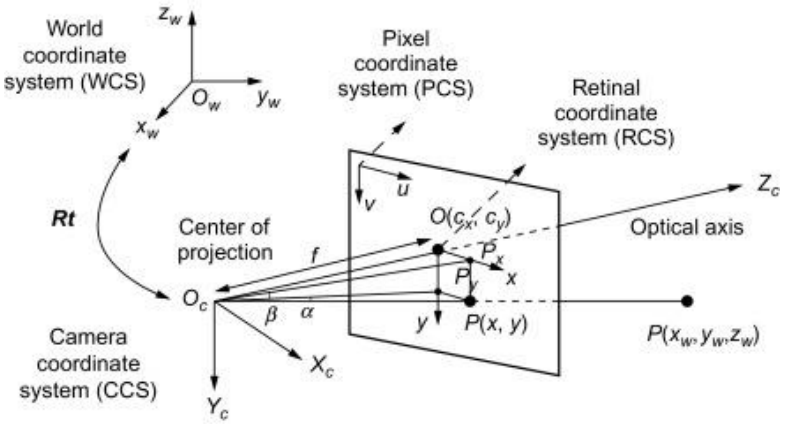


圖 6(四個坐標系示意圖)

將三維世界中的物體座標 $P(x_w, y_w, z_w)$ (世界座標)，轉換成以相機焦點為原點的座標 $P(x_c, y_c, z_c)$ (相機座標)，接著再將此座標 $P(x_c, y_c, z_c)$轉換成以感光元件平面中心為原點的座標 $P(x, y)$(圖像坐標)，最後再轉換成以 CCD 左上為原點的 $P(u, v)$(像素坐標)以及灰階數值，即是針孔成像的過程。而在這過程中，需使用大量旋轉矩陣、位移矩陣以及修正畸變參數，於此不加以論述，後續會介紹本研究使用的方式。

(三) YOLOv3 Real-Time Object Detection

Yolo 系列(You only look once)為物件偵測及影像辨識的類神經網路演算法(深度學習)，優點為快速，能達到即時辨識的效果。其最大的特色為利用整張圖片作為輸入，直接預測 bounding box(物件框架)的位置及類別。而 YoloV3 為目前 Yolo 最新的版本，使用了 resnet 網路(加深網路層級來解決梯度問題)和 FPN 網路(提升小物體辨識能力)，提升了速度跟精度。

而另一大宗的影像辨識的方法為 CNN(捲積神經網路)，CNN 為模仿人腦的認知方式，將每個特徵抽象化並進行堆疊，而在辨識過程中，便是使用局部的比對，逐步綜合比對結果。優點為結果較為精確。目前有許多 CNN 的延伸方法：R-CNN、Fast-CNN、Mask R-CNN 等。

本研究使用 YOLOv3 而非傳統的 CNN，是因為 YOLOv3 運算的即時性恰能符合本研究及時辨識及抓取的需求。

(四) 深度學習優化演算法

優化演算法的目的在於加快神經網路的訓練速度，使得目標函式快速收斂至接近正解。常見的演算法有 RMSprop、SGD、Adam 等，其主要運作方式皆是以給定待優化的模型參數 θ 和目標函式 J(θ)後通過梯度下降更新 θ ，使新的目標函式 J(θ)向正解逼近。

1. 計算目標函數對於參數的梯度

$$g_t = \nabla_\theta J(\theta)$$

2. 根據歷史梯度計算一階、二階動量

$$m_t = \emptyset(g_1, g_2, \dots, g_t)$$

$$v_t = \psi(g_1, g_2, \dots, g_t)$$

3. 更新模型參數(ε 為平滑項)

$$\theta_{t+1} = \theta_t - \frac{1}{\sqrt{v_t + \epsilon}} m_t$$

本研究將使用 Adam 演算法於自我校正和提升精度。

三、總流程圖

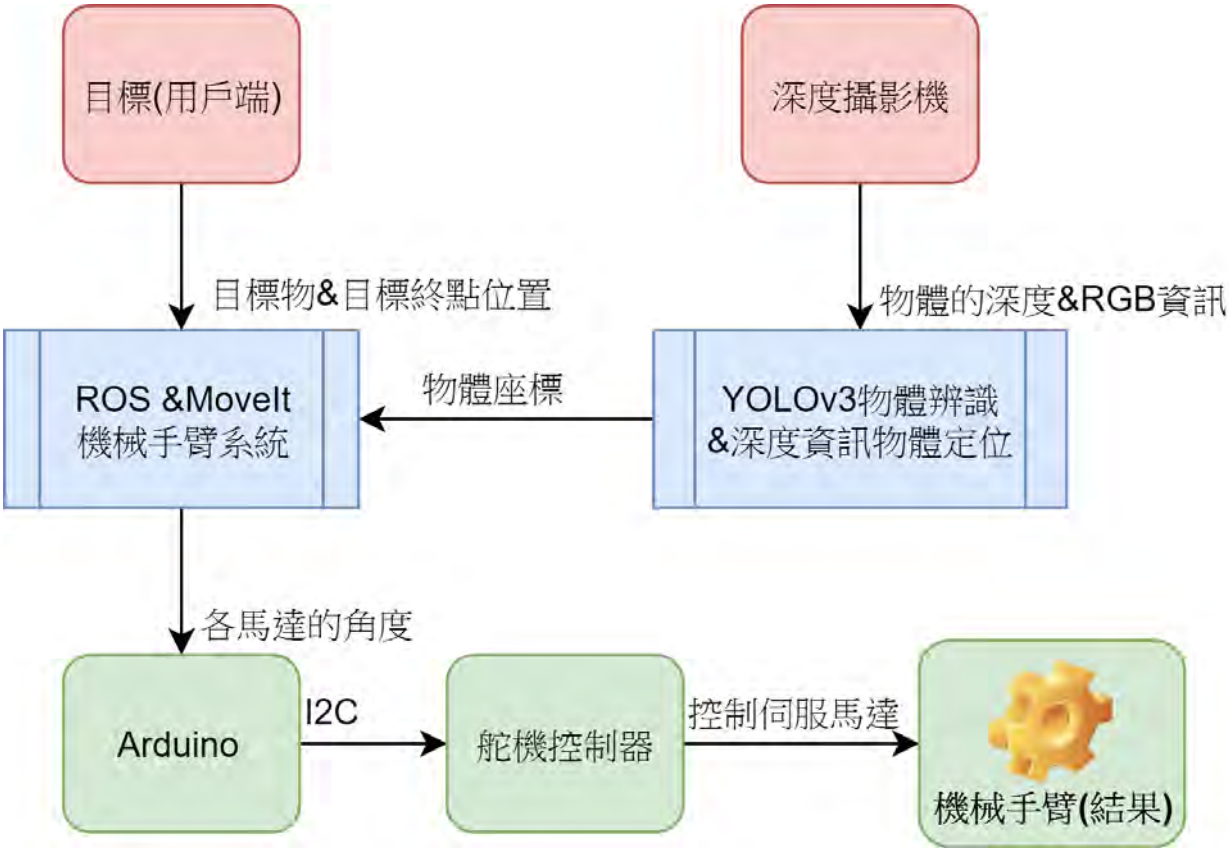


圖 7(流程圖)

四、調整伺服馬達及組裝機械手臂

(一) 使用 Arduino IDE 開啟調整伺服馬達的 Example

我們所使用的 Arduino Micro 附使用手冊及 Demo Code，讓使用者能直接使用範例程式控制伺服馬達至指定角度。

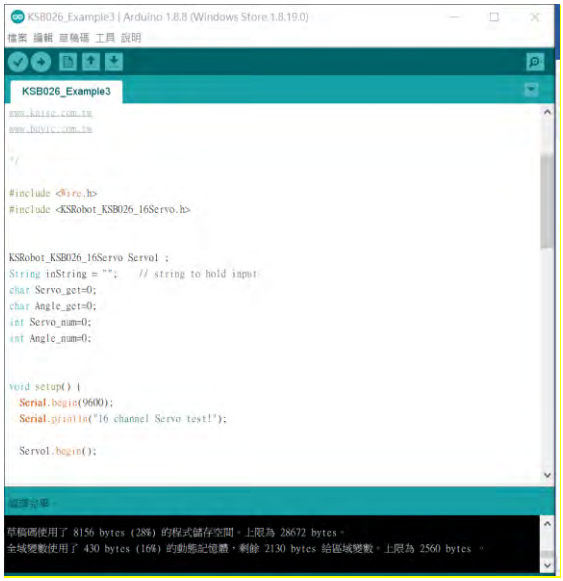


圖 8(Arduino Demo Code Example 3)

(二) 將調整好的伺服馬達依照自訂姿態組裝

由於每個伺服馬達各自的工作區間不同，因此要照一定的方向及角度組裝零件及伺服馬達，以下為每個伺服馬達在 90 度時呈現的手臂姿態。

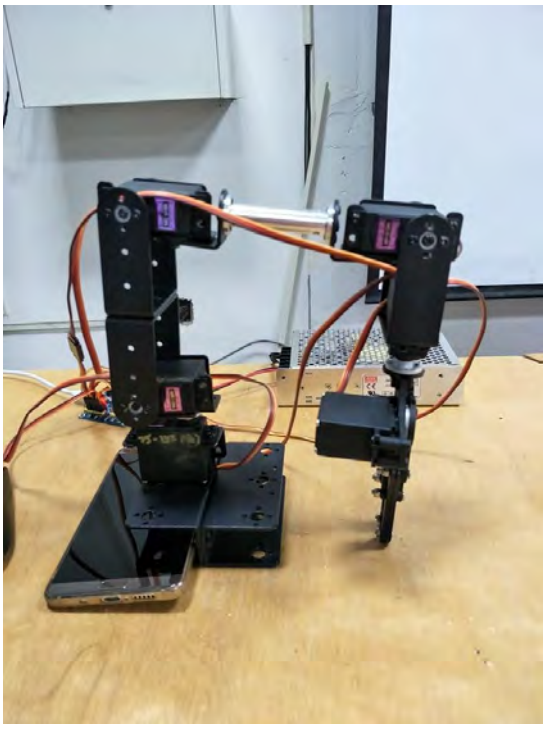


圖 9(機械手臂在全部伺服馬達為 90 度時的姿態)

五、並利用 Solidworks 建模並匯出 URDF 檔

因要在 ROS 系統上計算出機械手臂的運動，ROS 必須有機械手臂的 URDF 檔。而在本研究中，因為我們使用自製的機械手臂，不像市面上的機械手臂有自帶 URDF 檔，所以必須要自行建模並匯出 URDF 檔，我們採用 Solidworks 將機械手臂的模型建出並用 Solidworks 的插件「SolidWorks to URDF Exporter」來我們使用 Solidworks 來製作。

(一) URDF (Unified Robot Description Format) - 統一機器人描述格式

URDF 檔是一種特殊的 XML 格式，係透過 link 及 joint 來表現機器人的主要結構：

1. link(構件)：為設計機器人構件的模型、碰撞、旋轉等之參數。
2. joint(關節)：設定關節的類型(revolute, continuous, fixed, floating, planar)、旋轉速度、角度限制等之參數。

用 joint 連結前後的 link，再設定 joint 的運動方程式，使的每個 link 及 joint 可以以彼此之相對位置建立出視覺模型。

```
<robot name="<name of the robot>">
    <link> . . . . . </link>
    <link> . . . . . </link>
    . . . . . .
    <joint> . . . . . </joint>
    <joint> . . . . . </joint>
</robot>
```

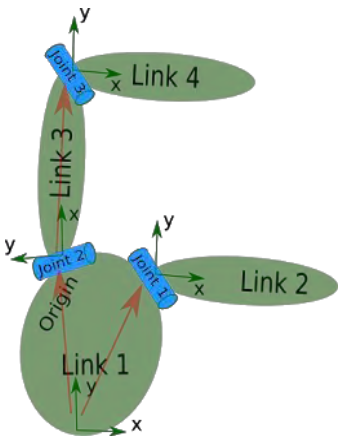
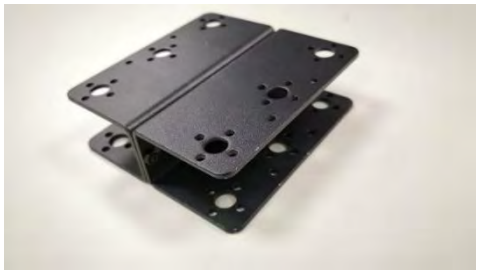
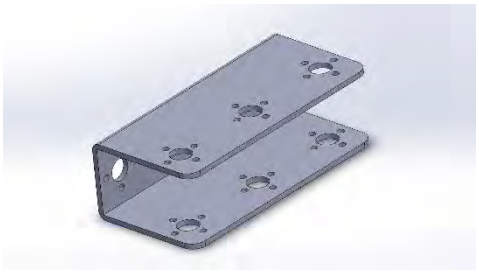
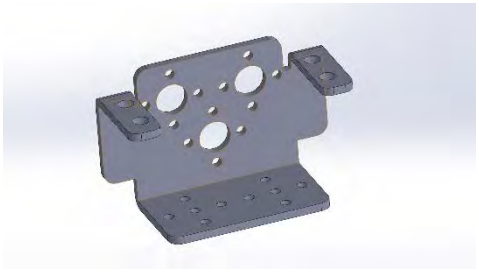
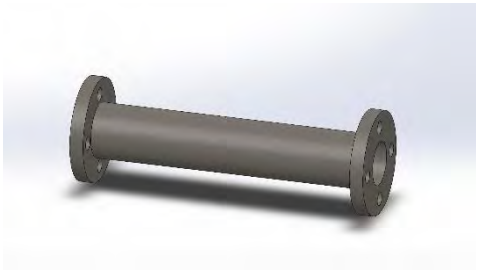


圖 10(URDF 示意圖)

(二) 使用 solidworks 建模機械手臂的各個零件

伺服馬達		
底座		
馬達支撐架		
法蘭桿		

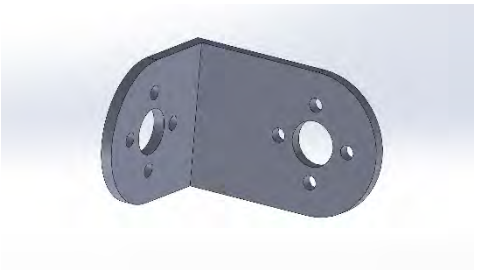
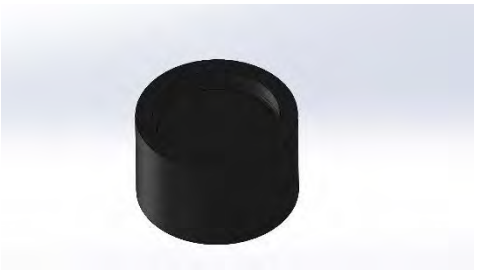
U 型 馬達 支架		
L 型 支架		
U 型 支架		
法蘭 杯士 軸承		
舵盤		

表 2

(三) 使用 Solidworks 的組合件將零件組合為我們的機械手臂並加入 URDF 所需的參考物

這個階段，我們將我們機械手臂組合起來，並加入生成 URDF 檔所需的每個 link 的座標系和每個 joint 的軸線。

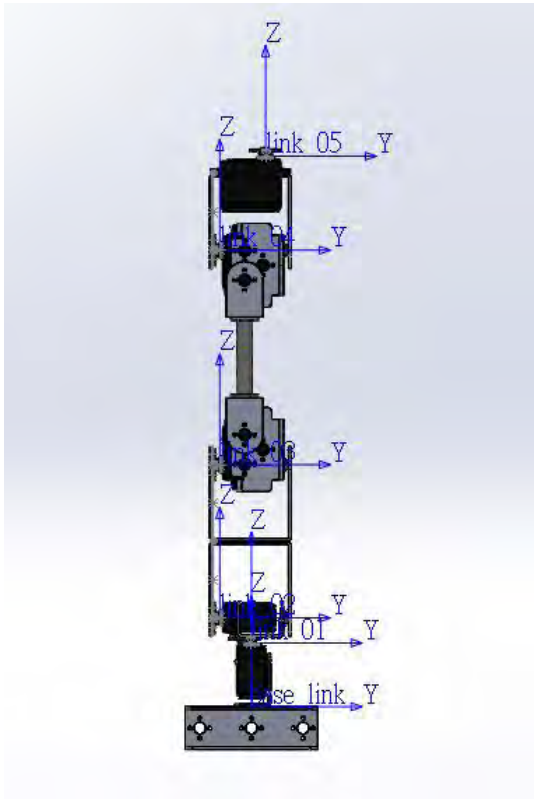
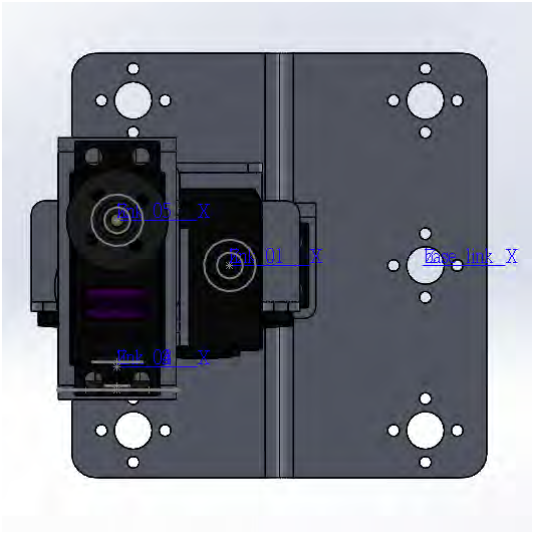
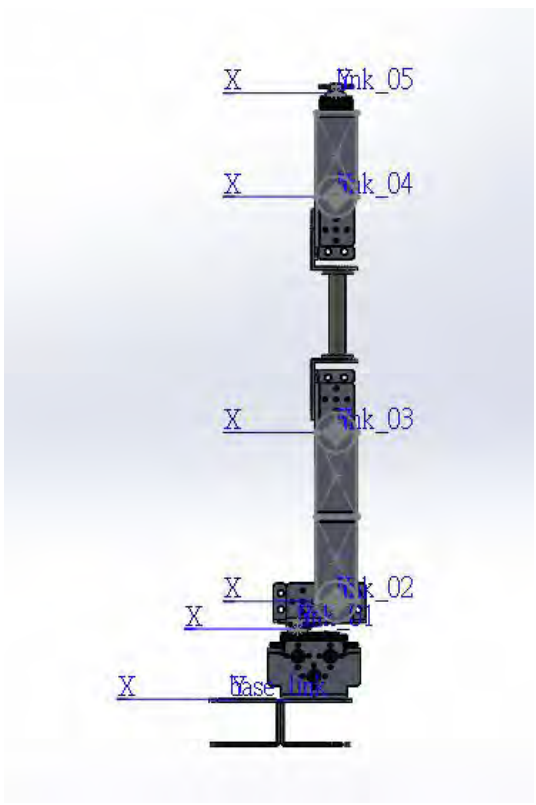
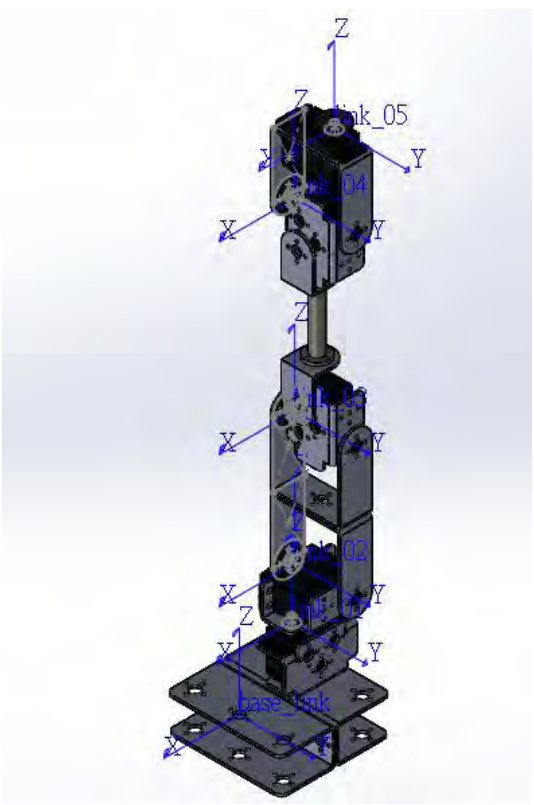
前視圖		上視圖	
右視圖		等角圖	

表 3

(四) 生成 URDF 檔

我們有了我們的機器人模型後，將 SolidWorks to URDF Exporter 插件開啟，這時除了有機器人的模型還需要設定機器人 joint(關節)和 link(構件)的資訊，才能生成機械手臂的 URDF 檔。

以下為需要的設定：

1. 建立一個基底的 base link

   為機械手臂的固定端，同時它的座標系也會做為往後 Moveit 場景規劃的世界坐標系。

2. 建立另外的機械手臂 4 個構件。

3. 設定每個構件的父子關係

   因機械手臂的結構是一節一節的臂由下往上連接，所以每個臂的構件的父構件都是上一個臂的構件，這樣當一個關節的旋轉時，下層的構件都會一起旋轉。

4. 指定每個構件的座標系。

   因在運動學時每個構件都需要自己的坐標系才能將 link 的狀態以 4x4 的矩陣數據化，所以每個構件的都需要屬於自己的坐標系

5. 指定每個構件所繞著旋轉的關節。

   我們需要指定這個構件是以哪個關節旋轉，而它的子 link 也同樣會受到這個關節的影響。

6. 設定構件的物理性質

   每個構件重量、重心位置、初始位置、初始角度。

7. 設定關節的轉動方式和轉動限制

   關節的轉動方式為 revolute(因伺服馬達為以某個軸旋轉並有角度限制)、角度限制、最大角速度、伺服馬達的扭力。

設定完後，SolidWorks to URDF Exporter 就會生成機械手臂的 URDF 檔。

六、在 ROS 和 MoveIt 平台上，編寫機械手臂的軟體

(一) 架構圖

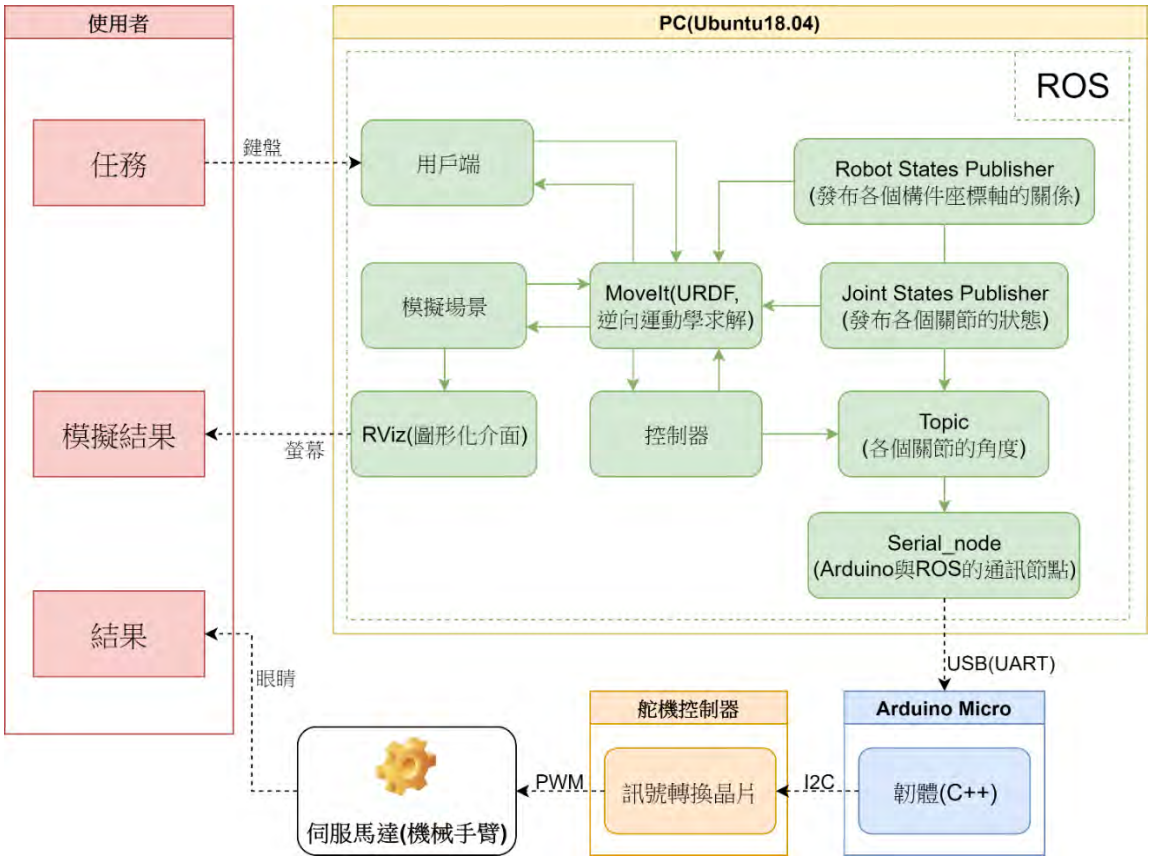


圖 11(架構圖)

(二) 建立 ROS 系統中的工作空間(Workspace)

ROS 系統中的每個專案都是一個工作空間，在這裡我們先創建一個工作空間。

(三) 使用 MoveIt! Setup Assistant 設置並生成包含 Moveit 所必要的檔案的 Package

MoveIt! Setup Assistant 是 Moveit 當中一個設置機器人設定的圖形化工具。

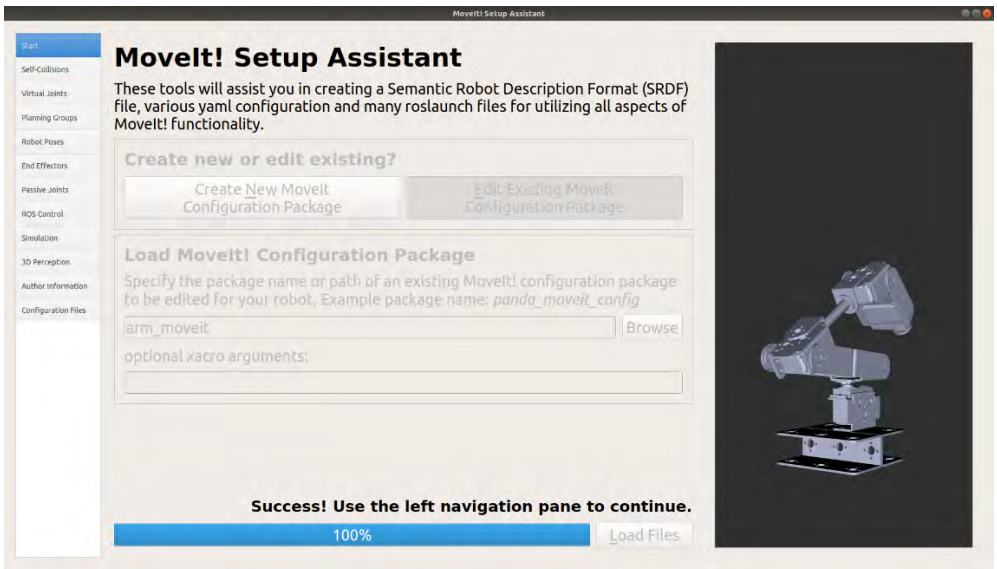


圖 12(MoveIt! Setup Assistant)

在此圖形化工具中，需要配置的機械手臂設定分別為

1. 自身碰撞檢測(Self-Collisions)

   在此設定中，我們要創建碰撞免檢矩陣(Allowed Collision Matrix, ACM)，ACM 會告訴後續運動規劃中的碰撞檢測算法那些肢體的碰撞是可略過的，以提高檢測的效率。

2. 虛擬關節(Virtual Joints)

   這種特殊的關節會連接機械手臂和規劃場景(planning scene)。在本研究中，機械手臂是固定的，所以我們將這個關節設為固定(Fixed)。

3. 規劃組(Planning Groups)

   在 Moveit 的運動規劃是以一組一組的規劃組來進行規劃。因此，我們將機械手臂分為手臂(arm)和夾子(gripper)兩個規劃組分別進行後續的運動規劃，並設置我們所要使用的逆向運動學求解器，此處我們使用的是 Trac-IK。

4. 機械手臂的姿態(Robots Pose)

   在此設定中，我們創建了機械手臂的預設姿態。

5. 末端執行器(End Effector)

   在這裡指的就是機械手臂的夾子，也就是後續所要指定目標的部位。

完成後，程式就會自動生成機械手臂所需的 Package。

(四) 測試機械手臂是否可在模擬環境中運動

在上一個步驟所生成的檔案中，有一個 demo.launch 可以使用 RViz 來模擬機械手臂的運動，並進行基本的規劃，以確認機械手臂的設置沒有錯誤。

(五) 編寫機械手臂控制器的 ROS 節點(C++)

因本研究不只進行機械手臂的模擬，還要使真實的機械手臂工作。而在使用真實機械手臂時，Moveit 並沒有控制器來進行接收和回傳，所以我們需要編寫一個手臂的控制器來使模擬環境中的機械手臂與真實的機械手臂溝通。

此處使用到了 ROS 當中的功能包 actionlib，雖然 ROS 本身已採用發布者和訂閱者之間的通訊方式，但當程式執行時使用者要取消或查看狀態時，這樣的通訊方式就無法滿足。因機械手臂在規劃中，須將狀態實時傳回，故採用此傳訊方法。

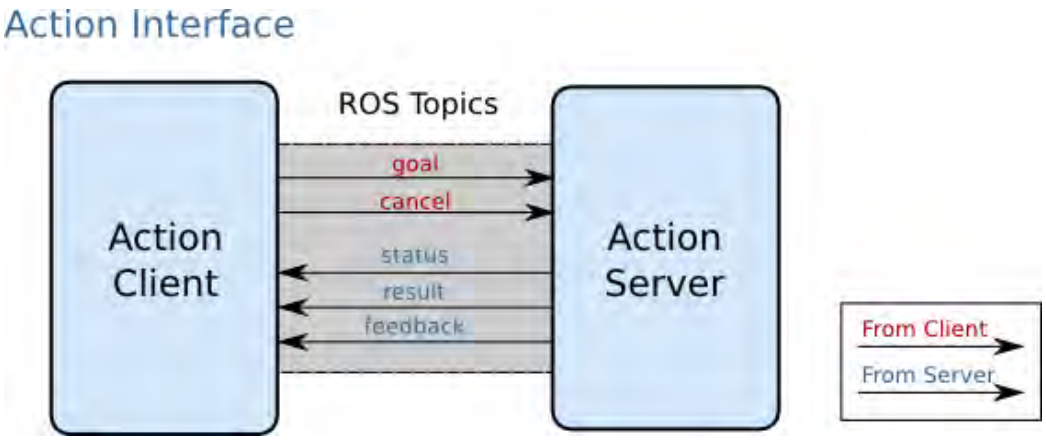


圖 13(Action 傳遞示意圖)

當我們將終點目標發布後，我們所設置的逆向運動學求解器會接收目標，並使用 Moveit 設置出機械手臂的運動規劃軌跡，再透過 ActionClient 將運動規劃軌跡以目標信息(goal)的方式傳送給 ActionServer，也就是我們的控制器。此時，我們控制器會將目標訊息處理為機械手臂各個關節的角度，如果處理成功，控制器會發布一個包含各關節角度的 Topic，而 Joint States Publisher 則會讀取這個 Topic 以更新機械手臂的狀態；如果處理不成功，控制器則會回傳失敗讓用戶得知。以下為此部分的流程圖。

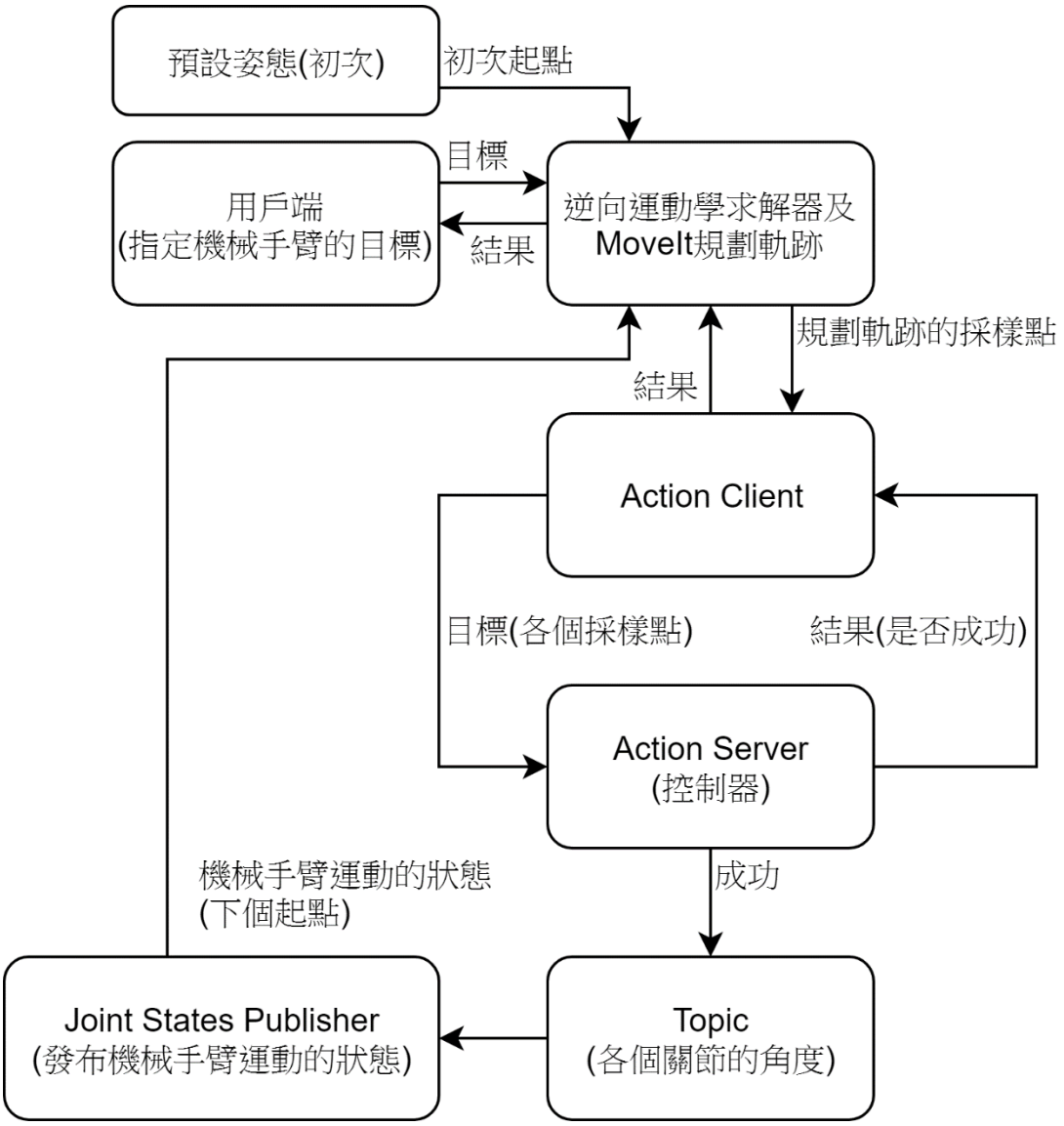


圖 14(機械手臂工作全圖)

(六) 編寫機械手臂運動規劃與場景規劃的用戶端的 ROS 節點(Python)

雖 Moveit 本身有提供路徑規劃的圖形化工具(RViz)，但無法達成如指定目標座標、抓取物體、放下物體等。且我們也需要進行模擬場景中的場景規劃：將真實世界物體加入模擬場景中讓機械手臂能夠避障，因此我們編寫了一個節點來作為機械手臂的路徑規劃和場景規劃的用戶端。

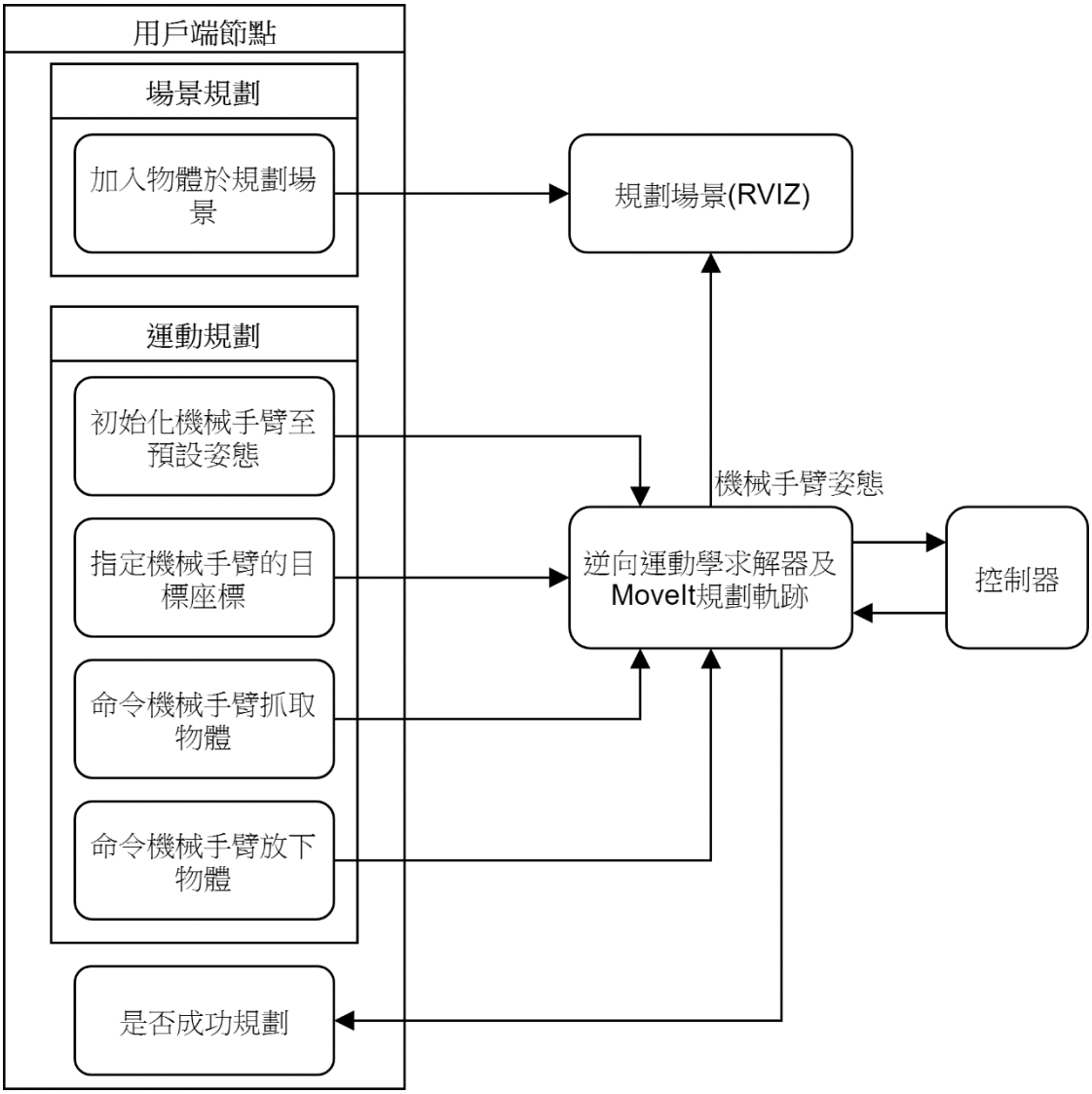


圖 15(加入場景規畫隻機械手臂工作全圖)

(七) 編寫 Arduino 的程式碼(C++)

Arduino 的工作十分簡單，訂閱控制器所發出的各關節的角度，並將角度轉換成 PWM 訊號使伺服馬達轉至指定角度。

但 Arduino 並沒辦法直接訂閱 ROS 的 Topic，因此我們使用了一個插件 rosserial_arduino 來作為 Arduino 和 ROS 系統的溝通橋樑，它提供了一個適用於 Arduino UART 的 ROS 通訊協議，允許我們的 Arduino 發布、訂閱 ROS 的 Topic，讓 Arduino 變成 ROS 的一個節點。當需要互相通訊時，必須啟動 rosserial_arduino 當中的 Serial_node 節點才能溝通。

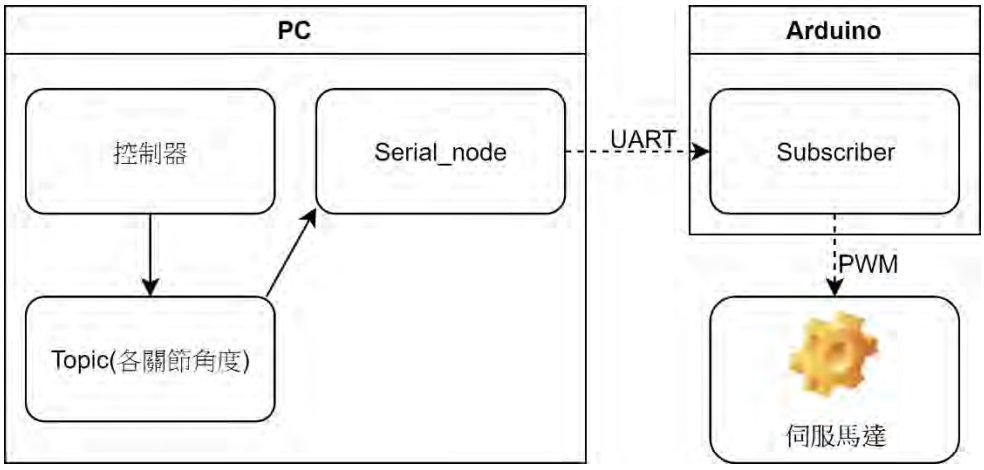


圖 16(ROS 溝通橋樑示意圖)

(八) 編寫 Package 中的 launch 檔

Launch 檔是 ROS 系統中的執行檔，由 YAML 所寫成，他能夠一次開啟多個節點，並能指定節點當中所需的系統參數，能夠增加節點的通用性和重複使用率。

我們將 Moveit 的核心 move_group、我們的控制器、我們的用戶端、模擬場景(Rviz)、Joint States Publisher、Robot States Publisher(從 URDF 發布機械手臂的初始狀態)和 Serial_node(發布關節角度給下位機 Arduino)這些節點和節點所需的參數寫入我們的 launch 檔(real_robot_arm.launch)中。

(九) 執行 real_robot_arm.launch

此時 Serial_node 會顯示錯誤，因為我們還沒將 Arduino 接上電腦，但其餘的路徑規劃、場景規劃、用戶端等都是可以順利執行的。

七、實測機械手臂

(一) 將 Arduino 接上電腦並將程式碼寫入 Arduino 中。

(二) 將機械手臂的伺服馬達接上的伺服馬達控制器並接上電源供應器。

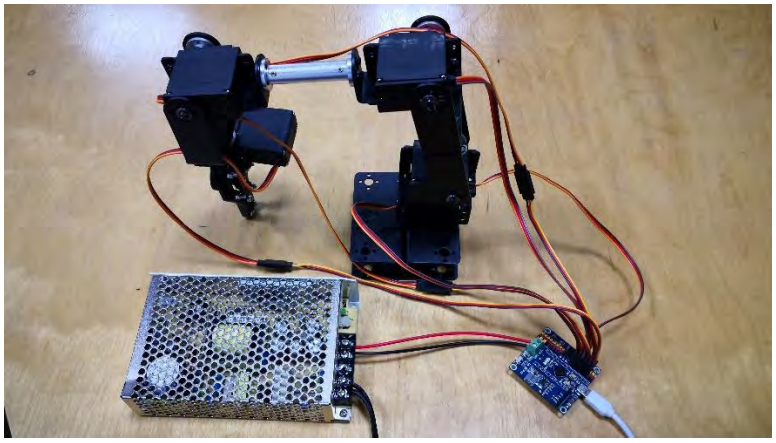


圖 17(自組機械手臂)

(三) 運行 real_robot_arm.launch

(四) 使用用戶端控制機械手臂

八、使用深度攝影機進行物體辨識和測距以取得要抓取用戶指定物體的座標

(一) 使用 OpenNI 作為深度攝影機的驅動程式

OpenNI 為一款針對深度攝影機提供的跨平台、開源開發環境。而我們在 ROS 系統中使用它所提供的 openni_launch 這個 Package 中的 launch 檔，以初始化我們的深度攝影機，並自動發佈深度攝影機的點雲(pointcloud)、深度資訊、RGB 資訊及相機的相機坐標系。

(二) 以 RGB 資訊進行 YOLOv3 物體辨識並發布用戶指定物體的像素座標

這裡我們使用 leggedrobotics 所編寫的一個從 Darknet 架構下的 YOLOv3 移植成為 ROS 的一個 Package (darknet_ros)，並以微軟提供的 COCO 的深度學習資料庫執行 darknet_ros 中的 launch 檔後，darknet_ros 中的節點會訂閱深度攝影機的 RGB 資訊並發佈物體辨識的結果，結果分別為辨識物體的名稱、辨識物體的在像素座標中的 u_{max}, u_{min}, v_{max}, v_{min}。

此時我們編寫一個節點先訂閱辨識的結果，當辨識結果的名稱為用戶的指定物體的名稱時，將指定物體的$(u_{max}+u_{min})/2$ 指定為 u_{mid} ，$(v_{max}, v_{min})/2$ 指定為 v_{mid}，而(u_{mid}, v_{mid})就是指定物體中心的像素座標了，此時我們就可以將(u_{mid}, v_{mid})發佈出去了。

(三) 將物體的向素座標與深度資訊轉換為相機坐標，並轉為物體在機械手臂座標系的座標(x_w, y_w, z_w)

這一個節點將訂閱物體的像素座標(u_{mid}, v_{mid})、深度資訊(depth)和相機的焦距(focal_length)並進行以下的運算得到相機坐標(x_c, y_c, z_c)。

$$x_c = \text{depth}$$

$$y_c = -(v - 480/2) * \text{depth}/\text{focal\_length}$$

$$z_c = (u - 848/2) * \text{depth}/\text{focal\_length}$$

先將相機座標軸的轉換成與世界座標軸相同。然後測量相機與機械手臂的 XYZ 距離，再將轉換後的相機座標減掉 XYZ 距離就變成我們所需的物體世界座標。此時，我們就得到了用戶指定物體的機械手臂座標(x_w, y_w, z_w)，並發布出去。

九、以物體的座標和用戶指定的物體放置位置達成機械手臂的自動化

(一) 編寫一個節點，並訂閱物體的座標(x_w, y_w, z_w)後發布至用戶端，讓此節點代替人為操控，使機械手臂的抓取夾自動抵達目標物後完成抓取並到用戶事先指定的放置位置以放置物體，達成機械手臂的自動化。

伍、研究結果

一、在 ROS 平台模擬後將結果傳至機械手臂進行運動

於 Roboware 中執行編寫好的 launch 檔，並指定座標予模擬手臂進行運動，再將計算出之伺服馬達角度傳至單片板進行實體手臂運動。

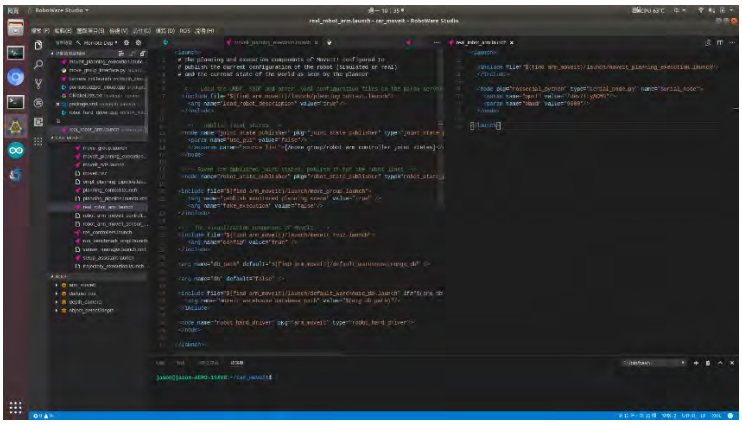


圖 18(Roboware 中的 launch 檔)

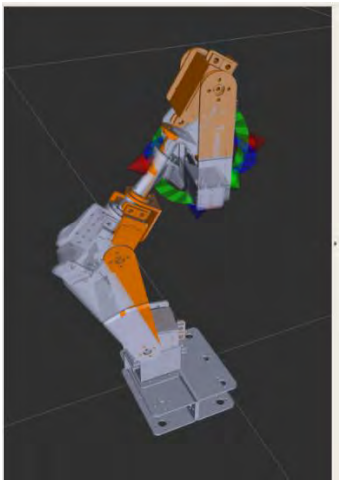


圖 19(虛擬機械臂)

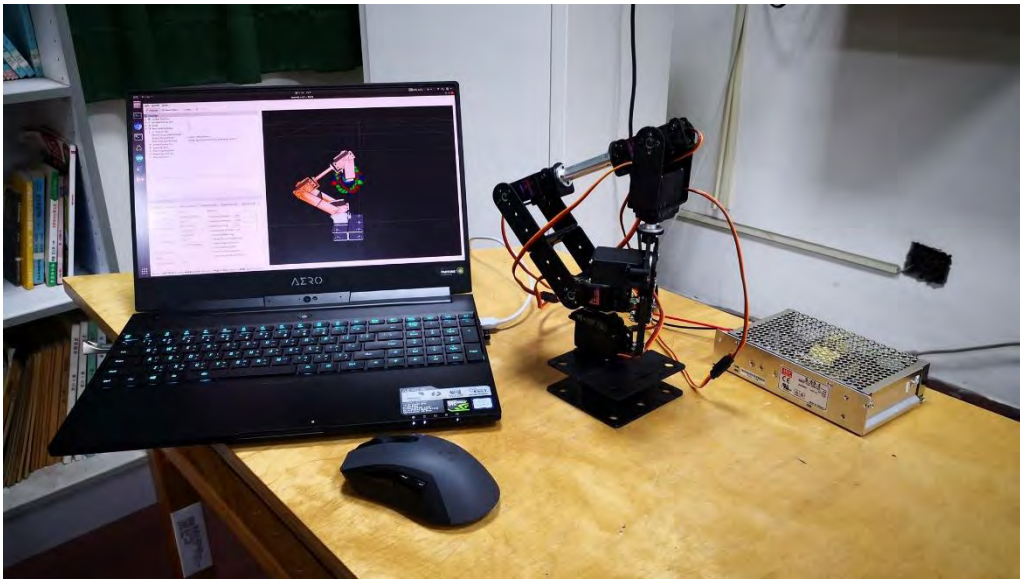


圖 20(電腦將角度傳給單片板以控制手臂)

二、以電腦視覺進行物件辨識和取得世界座標

利用 YoloV3 進行深度學習後，將攝影機對準預設環境進行辨識。將所得出的世界座標傳給電腦，轉換成角度值傳給單片板進行運動。

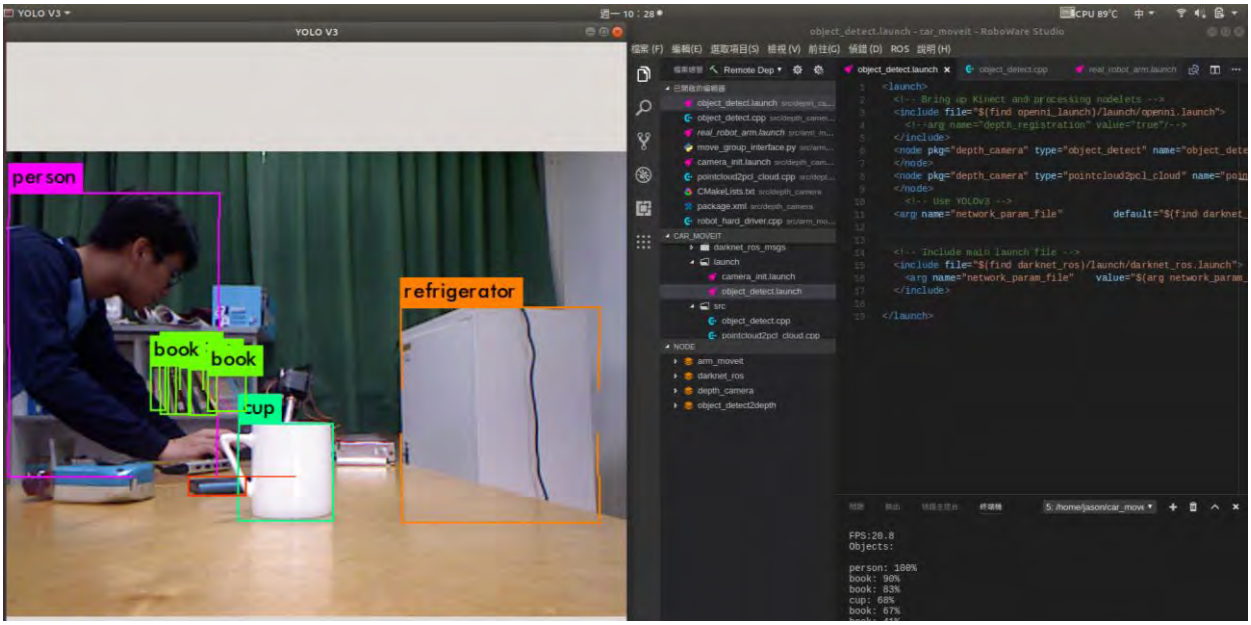


圖 21(物件辨識並加 Bounding Box 進行框架)

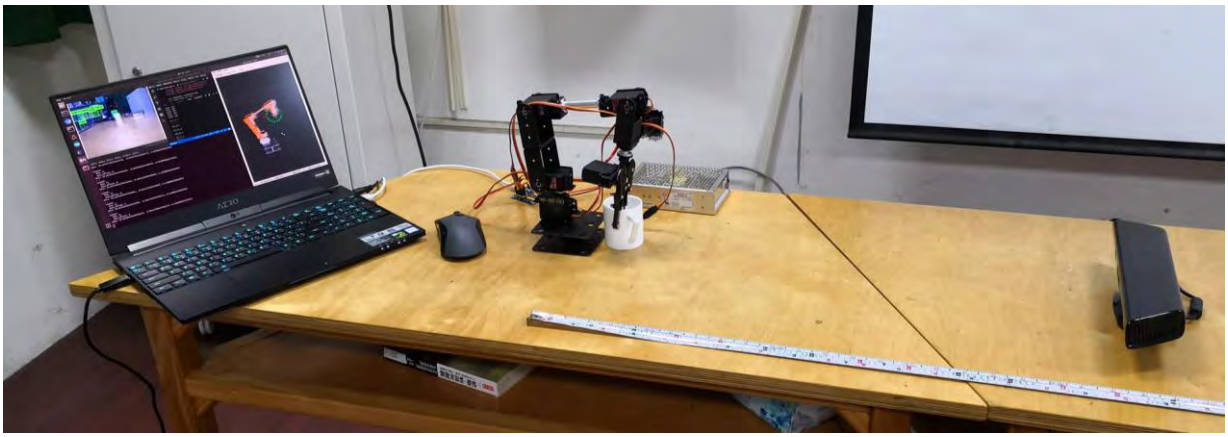


圖 22(統合深度攝影機、機械手臂及電腦)

陸、討論

本研究在進行時有幾項問題

一、機械手臂的不穩定現象

在同時操控多個伺服馬達以完成任時，會出現抖動跟傾斜現象的原因為：

(一) 電源供電不足

伺服馬達所需電壓為 3W 至 15W，而 Arduino 的供電最大為 5W，如果只是操控單一馬達且沒有負載並無問題，但同時操作多個馬達會使供電功率不足，需要外接電源來解決問題。

(二) 伺服馬達扭力不足

市面上非工業的伺服馬達所標示的扭力大多為短時間所能產生的最大扭力，並沒有辦法長時間維持。針對力矩較大的伺服馬達，我們已在不增加過多成本的情況下進行升級。儘管如此，當伺服馬達扭力不足時，還是會出現些微的上下抖動。

(三) 手臂結構不堅固

我們加固手臂結構，並進行整體外觀得整理。

1. 整線：避免手臂運作時被電線纏住，並在整線時預留工作範圍的空間。
2. 螺絲：我們將原本的螺絲換成螺帽較大的螺絲，以增加接觸面積；並增加螺絲數量，加強穩固零件之間的連接。
3. 增加水平旋轉支架：原本水平旋轉的伺服馬達只有少數支撐點，當負重大時，機械手臂容易產生傾斜。因此我們在附近加上支架了，增加水平旋轉面與底座

的接觸面積，減少傾斜的程度。

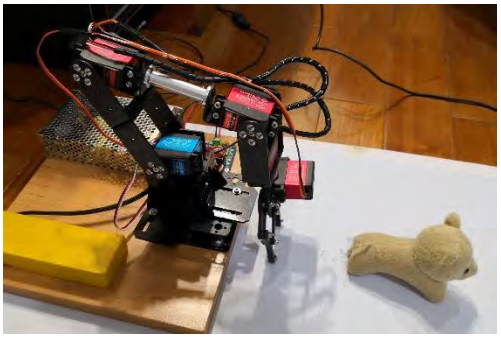


圖 23(加固後的機械手臂)

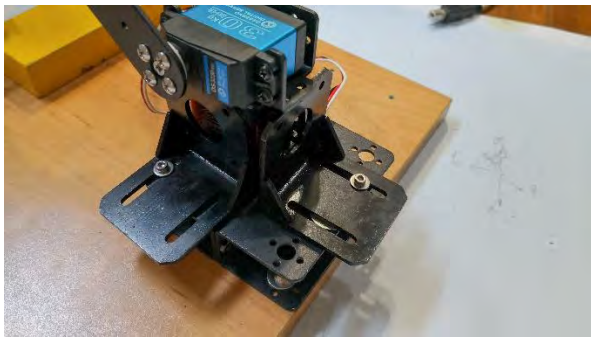


圖 24(水平旋轉支架)

二、伺服馬達角度不精確導致成功率下降

為了保持機械手臂的價格低廉，我們決定從演算法來增加機械手臂的精準度。

(一) 黏貼 ArUco 標籤於機械手臂末端，並透過深度攝影機辨識後取得標籤的位置。將標籤的位置轉換為機械手臂末端的位置，作為真實世界的機械手臂末端位置。

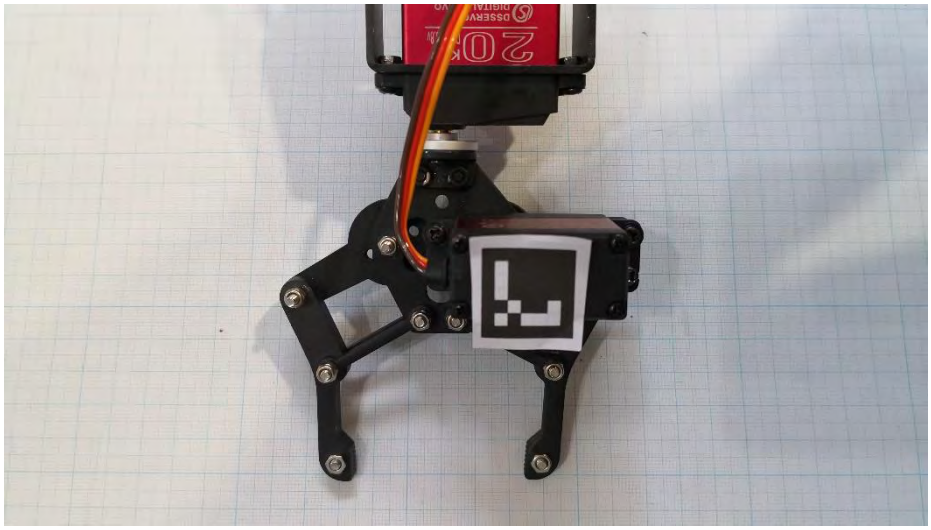


圖 25(ArUco 標籤於機械手臂末端)

(二) 將真實世界的機械手臂末端位置(θ_r)、模擬世界的機械手臂末端位置(θ_s)和目標的機械手臂末端位置(θ_g)使用 Adam 深度學習優化演算法

$$J = (\theta_g - \theta_r)^2$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial J}{\partial \theta_r}$$

$$v_t = \beta_1 v_{t-1} + (1-\beta_2)(\frac{\partial J}{\partial \theta_r})^2$$

$$\widehat{m_t} = \frac{m_t}{1-{\beta_1}^t}$$

$\beta_1 = 0.9$

$\beta_2 = 0.99$

$\gamma = 0.02$

$\epsilon = 10^{-8}$

m_1=0

v_1=0

$$\widehat{v_t} = \frac{v_t}{1 - {\beta_2}^t}$$

$$\theta = \theta - \gamma \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

$$\theta_s = \theta_s + \theta$$

進行迭代逐漸更新模擬世界的機械手臂末端位置(θ_s)以規劃出新的真實世界中的精準路徑，並將誤差$(\theta_g - \theta_r)$逐漸逼近 0。

本研究透過此方法可將原本的誤差 1~3cm 縮減為 0.15~0.25cm。

三、手臂未達六自由度

本研究組裝之機械手臂雖然有六個伺服馬達，但最後一個是控制爪子的開闔，因此自組之機械手臂能控制末端位置的自由度時為五個(三個控制移動，兩個控制旋轉)。由於想要表現出完整的末端姿態最少需要六個自由度，所以在使用逆向運動學時需要限制伺服馬達的運動角度、同時使用求解率更高的 Trac-IK 來減少求解失敗的可能性。

四、物體辨識與測距不精準

原先使用的 Kinect V1 有最短距離過長(0.8m)和解析度不足(深度 320×240、顏色 640×480)的問題，因此為了縮小裝置所需空間同時增加精準度，我們更換了 Intel® RealSense™ D435 深度攝影機和 Intel® RealSense™ SDK，使解析度提升(深度 1280×720、顏色 1280×720)和最短距離縮減(0.25m)，提升精準度。此外，我們添加了 ArUco 標籤作為校準相機位置的目標物，並將測量出的誤差進行補償。

五、避障應對

起初本研究只將最後計算出的值傳回機械手臂，但這會導致機械手臂只能進行起點到終點的直線運動。因此，本研究設計成在 ROS 平台中也模擬障礙，並讓電腦每 100 毫秒傳輸角度值予機械手臂，使其能依照計算的軌跡移動並避障，同時也能增加手臂的平滑度。

柒、結論

一、研究總結

本研究的機械手臂系統藉由機器學習、電腦視覺及機器人運動學結合，將機械手臂拓展出自動抓取物體和自我校正的功能，並在機械手臂本身不精準的狀況下，透過演算法將其誤差補償，雖仍未能達到工業級別的精準度，但在大部分情況已可完成本研究的任務目標。

二、本研究的機械手臂系統，其優勢有：

(一)程式容易更改和開源

因購買之零件並無附帶開發源或程式庫，因此本研究製作之機械手臂軟件全是由本組人員編寫。也因為如此，對於機械手臂軟件的部分，我們能擁有更多的操控權限，從最根本的程式碼進行修正，而不會侷限透過開發公司的操作視窗進行調控。

(二)逆向運動學

利用參考自 MoveIt 的程式碼加以修改，我們能給予虛擬手臂向指定座標運動，再傳輸角度至真實手臂進行操控，而不是一個一個 servo 進行微調，運行至目標。

(三)可自動化

相較於其他機械手臂，本研究所開發之機械手臂能透過電腦視覺達成自動化效果，亦能有效應用在工業用途。

(四)視覺系統

本研究的機械手臂加裝了視覺系統使機械手臂有更多的可能性，如本研究做的精度增加，自動抓取。

(五)自適應校準

透過深度學習方法校正機械手臂的精度，即使機械手臂的精度、物理條件很差，仍可透過自我校正，將機械手臂的精度提升至一定的水準，而不會侷限於機械手臂本身的條件。

三、本研究的機械手臂系統，其彈性有：

(一)零件容易代換

目前本研究使用的伺服馬達，具有精度不佳、扭力不足、有死區的問題，在有較高預算的情況下能夠以更加精確且穩定的伺服馬達進行替換。而進行數據運算的電腦，也可以使用其他運算平台代換。

(二)未來發展空間

若能將電腦模組嵌入機械手臂系統中，便能將運算與電腦分離，使其不依賴於電腦，達到可遠距操控、輕量化、本地化的效果。除此之外，如果將機械手臂和深度攝影機同時加裝於自走車的裝置上，使機械手臂在 XY 平面上的工作範圍拓展，也是一項值得發展的方向。

再者，本研究使用了 YOLOv3 即時物體影像辨識和 Adam 深度學習優化法使機械手臂擁有自主控制和校正的功能，於無人車或太空車探索等自動化且無法人為校準和操控科技有巨大的發展空間。

捌、參考資料及其他

一、SolidWorks 立體製圖基礎課程講義，取自 http：//dragon.ccut.edu.tw/~jlshih/personal/handout/SW_unit_1.pdf?fbclid=IwAR0qYejFnICxWF-oNc3XIqHMGTyHbAKcoAn_wqjxWO-AQFNaDD1RC6HTGzQ。

二、SolidWorks to URDF Exporter，取自 http：//wiki.ros.org/sw_urdf_exporter。

三、ROS，取自 http：//wiki.ros.org/。

四、MoveIt，取自 https：//ros-planning.github.io/moveit_tutorials/index.html。

五、R.Patrick Goebel(民 105 年)，ROS By Example 1。中山大學出版社。

六、R.Patrick Goebel(民 106 年)，ROS By Example 2。中山大學出版社。

七、Stephen Prata，C++ Primer Plus 中文版。碁峰出版社。

八、Kinect 原理分析與系統設計，取自 http：//vaplab.ce.ncu.edu.tw/chinese/pcchang/course2013b/comsp/3/members.htm。

九、Arduino IDE，取自 https：//www.arduino.cc/。

十、OpenNI，取自 http：//wiki.ros.org/openni_launch。

十一、 YOLO：Real-Time Object Detection for ROS，取自 https：//pjreddie.com/darknet/yolo/。

十二、 YOLO ROS：Real-Time Object Detection for ROS，取自 https：//github.com/leggedrobotics/darknet_ros。

十三、 Pinhole camera model，取自 https：//en.wikipedia.org/wiki/Pinhole_camera_model

十四、 從 SGD 到 Adam_____深度學習優化算法概覽，取自 https：//zhuanlan.zhihu.com/p/32626442

【評語】052311

本作品是一套完全自行組裝且結合影像辨識之機械手臂，並藉由機器學習讓機械手臂擁有自動抓取物體和自我校正的功能，此外，在機械手臂本身不精準的狀況下，也透過演算法來補償其誤差。是一具實用性的作品。

作品海報

壹、摘要

本研究使用自製的機械手臂並透過ROS與MoveIt實現機械手臂的路徑規劃，且給予機械手臂自身的視覺系統，並透過深度學習進行物體辨識並使用深度攝影機獲取物體的距離，使機械手臂能夠自行移動到目標物的位置並抓取物體。將電腦視覺、深度學習與機械手臂結合，機械手臂便可以達到自動化，而無須人為操作。本研究所開發的機械手臂系統為完全開源、價格低廉且軟體方面可套用於其他安裝ROS的系統中，十分適合與其他領域結合。

貳、研究動機

在這個科技快速發展，機器逐漸取代人工的時代，機械手臂在工業中被大幅應用，而上次在實驗室參觀計畫中，電機工程系實驗室的教授及研究生向我們示範如何操控機械手臂及機器人運動學的相關理論，在他們的操作下，機械手臂可以流暢地移動並抓取物品，靈活的轉動、控制使我們看得目不轉睛。

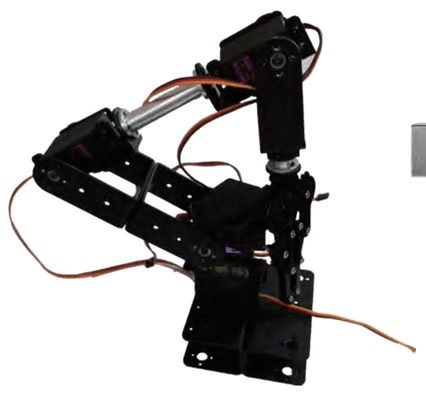
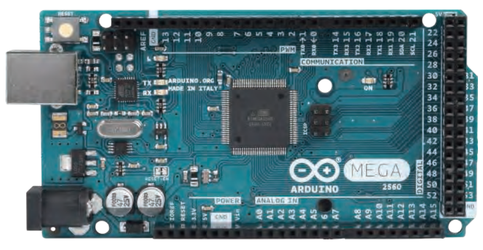
回到學校之後，我們也想要自己製作一個機械手臂，在給定目標物座標的情況下順利抓取物體，再結合電腦視覺，達到自動化的效果。

參、研究目的

一、設計並組裝出價格相對便宜且具開發空間的機械手臂。
二、設計並編寫出機械手臂計算及運動的軟體。
三、將計算出的數據套用在實體機械手臂。
四、結合電腦視覺讓實體機械手臂達到完全自動化的效果。

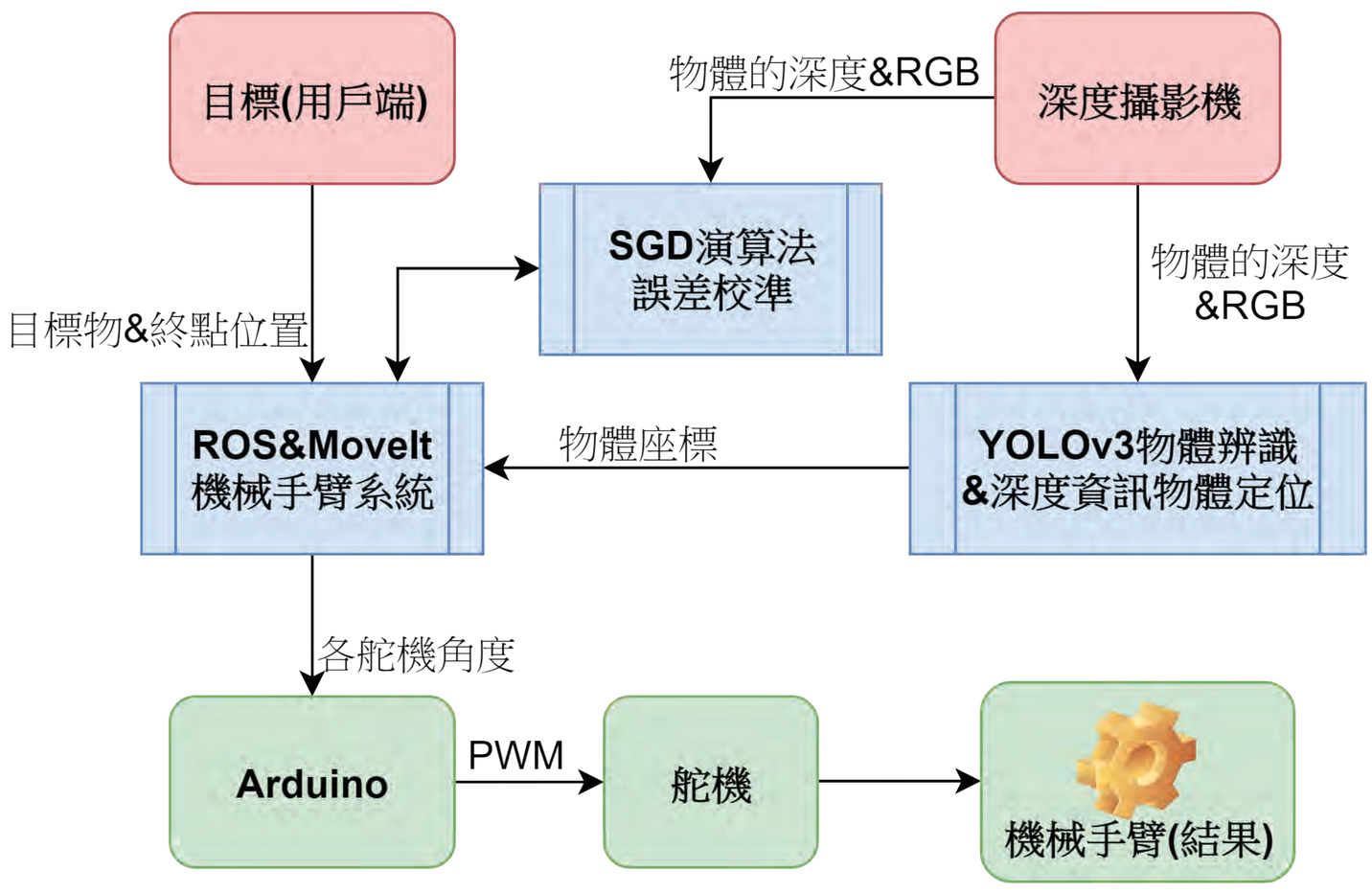
肆、研究設備及器材

1. 電腦　2. 自組機械手臂　3. Arduino MEGA 2560　4. 深度攝影機　5.電源供應器60W


伍、研究過程及方法

一、 系統架構圖



二、 理論基礎

機械手臂規劃路徑

逆向運動學
互相連結的物件中，末端點移動時，整體的節點會被牽動的行為。

深度攝影機的物體辨識

YOLOv3
物件偵測及影像辨識的類神經網路演算法(深度學習)。

物體座標系轉換

針孔相機座標成像
將攝影機的像素坐標及深度經由矩陣轉換為物體的世界座標。

誤差修正演算法

SGD&Adam
座標進行梯度迭代，將誤差逐步收斂到最小值的演算法。

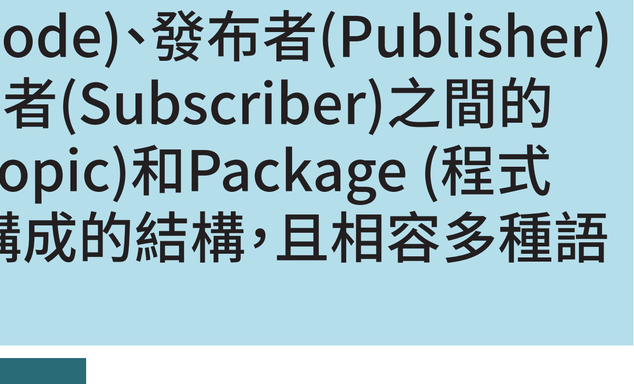
三、 系統環境

ROS

為本研究的核心，所有行為都建立在這個系統上，其特點在於由節點(node)、發布者(Publisher)和訂閱者(Subscriber)之間的傳訊(Topic)和Package (程式包)所構成的結構，且相容多種語言。

Moveit

是ROS系統中整合與移動和路徑規劃相關組合包的運動規劃庫，以move_group為核心。



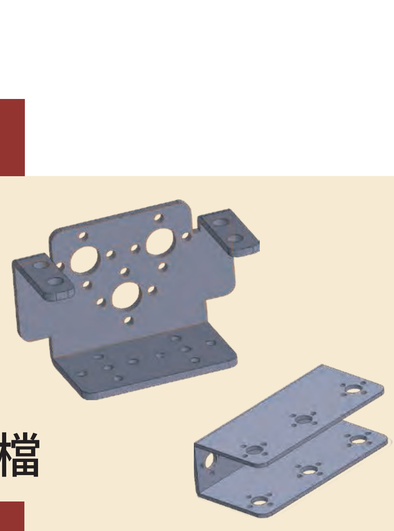
MoveIt架構圖

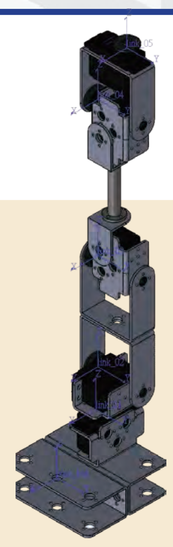
四、 研究過程

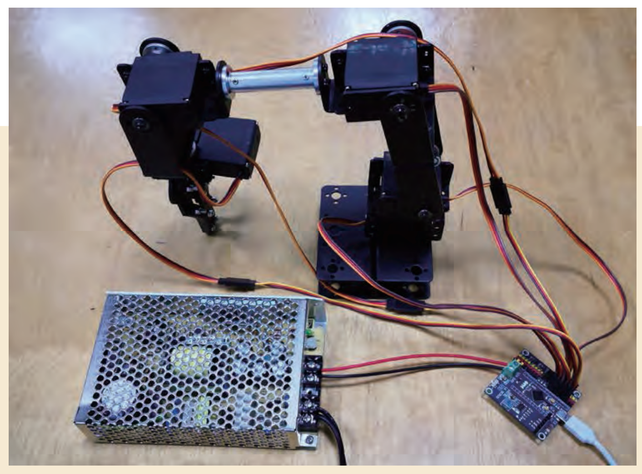
組裝機械手臂並建模生成URDF

1. 利用Arduino IDE將各個舵機設定至中央
2. 將舵機依照預設姿態組裝
3. 使用Solidwork建模
4. 將模型與機械手臂的物理性質

以SolidWorks to URDF Exporter插件生成URDF檔





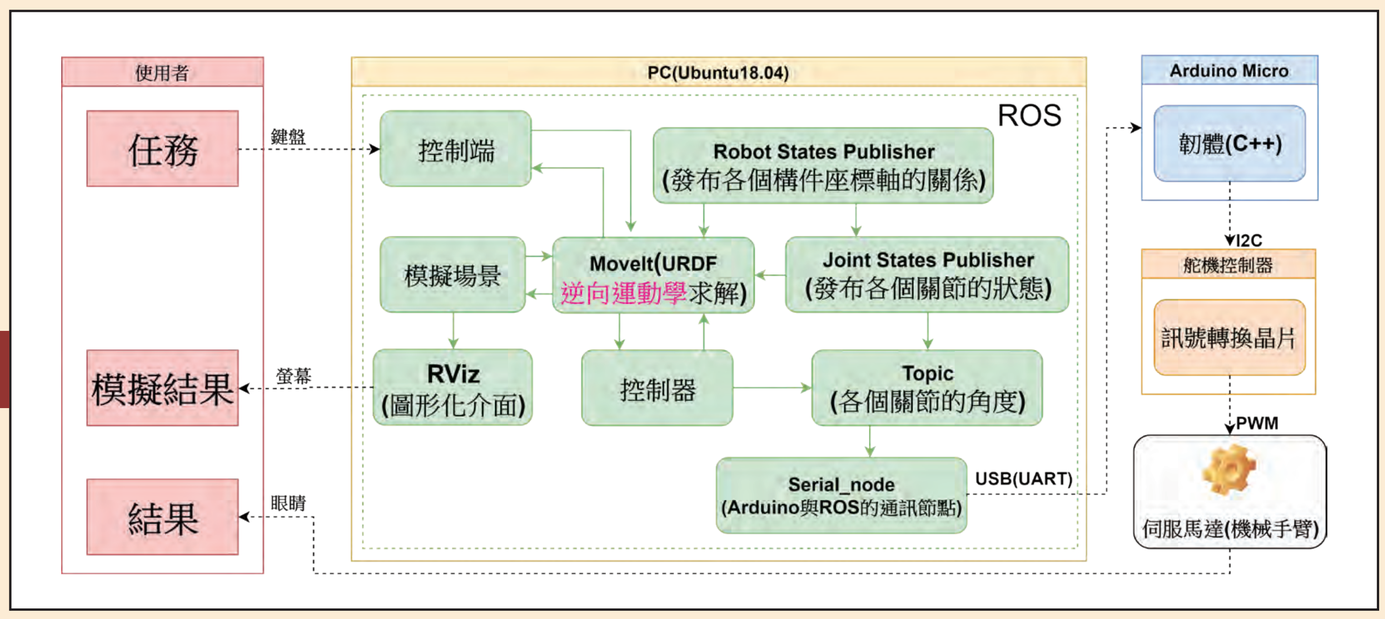


編寫機械手臂的驅動程式和控制端

以URDF生成並配置機械手臂的基礎環境 → 測試在預設模擬環境中的運動 → 編寫機械手臂的驅動控制器 → 編寫運動場景規劃的控制端 → 執行

驅動程式作為機械手臂的驅動核心，主要工作就是將控制端的要求給逆向運動學求解，並使模擬環境的機械手臂與真實的機械手臂同步。

控制端則是負責與用戶互動，如接收要求、顯示機械手臂狀況、要求是否成功等。



編寫Arduino的韌體

Arduino利用rosserial與PC的ROS溝通，並訂閱控制器所發出的各關節的角度，並透過舵機控制器將角度轉換成PWM訊號使伺服馬達轉至指定角度。

使用深度攝影機取得指定物體的座標

1. 用RealSense ROS作為深度攝影機的驅動程式。
2. 以RGB資訊進行YOLOv3物體辨識並發布物體的像素座標。
3. 將物體的向素座標與深度資訊轉為物體在機械手臂的座標系(針孔相機座標成像)。
4. 以物體的座標和用戶指定的物體放置位置達成機械手臂的自動化

加入演算法

藉由深度攝影機得到手臂末端點，並使用深度學習常用的優化演算法如Adam,SGD，將手臂末端校正回目標點。

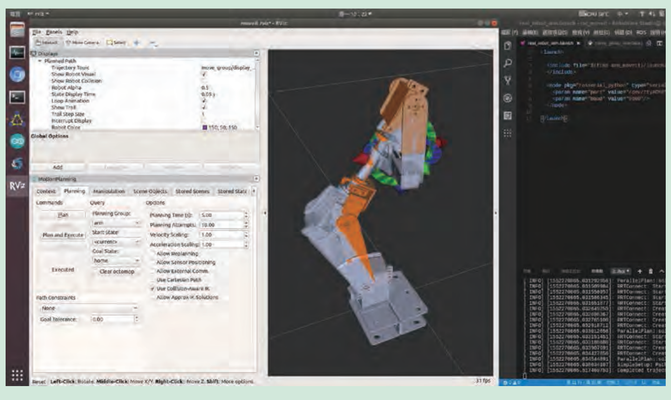
實測機械手臂

將機械手臂、深度攝影機架設好，輸入深度攝影機和機械手臂的xyz軸差和要夾取的物體，並執行。

陸、研究結果

一、 在ROS平台中模擬機械手臂

於ROS中模擬機械手臂的運動，並將角度傳至單片板使實體機械手臂運動。



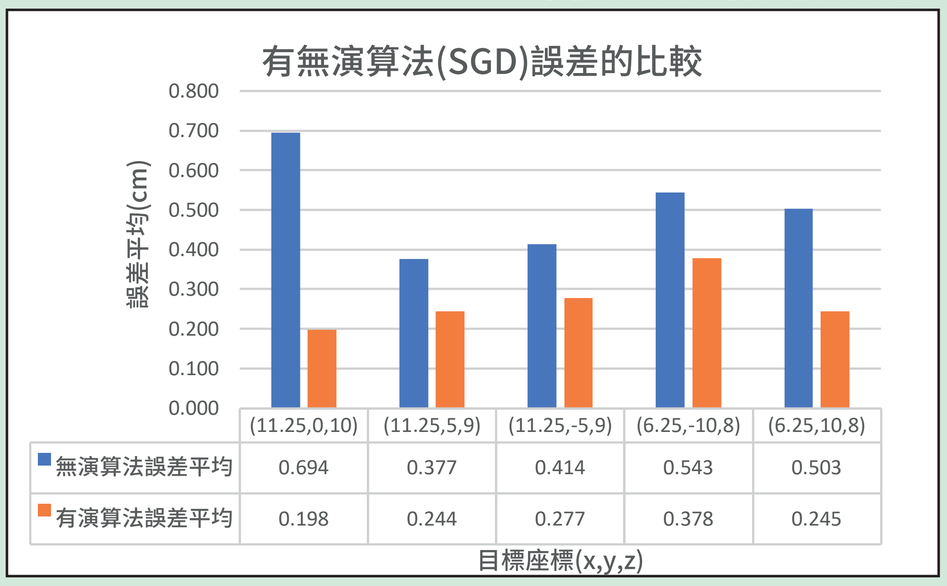
二、結合電腦視覺及機械手臂進行物體抓取

將電腦經YOLOv3學習後進行物體辨識，將辨識結果中處理為，並將深度資訊進行運算後傳角度給單片板。

三、使用深度學習優化法修正誤差

加入SGD演算法進行校準，並測量有無演算法的誤差距離。

▼有無演算法的誤差比較圖



	(11.25,0,10)	(11.25,5,9)	(11.25,-5,9)	(6.25,-10,8)	(6.25,10,8)
無演算法誤差平均	0.694	0.377	0.414	0.543	0.503
有演算法誤差平均	0.198	0.244	0.277	0.378	0.245

目標座標(x,y,z)

四、系統測試的成功率

統合電腦視覺、深度學習、深度優化和機械手臂進行運動並記錄抓取物體的成功率。

▼物體自動辨識並抓取的成功率

柒、討論

一、 舵機的不穩定現象

1. 供電不足→因舵機的耗電功率大於控制板可供的最大功率，故另外使用電源供應器供電。
2. 扭力不足→我們已將負載大的舵機更換為扭力大的舵機，並在成本與穩定中找到平衡。
3. 手臂結構不堅固→對此，我們將機械手臂的螺絲更換為螺帽較大的螺絲，並整線。
4. PWM訊號不穩定→我們將控制板更換為Arduino MEGA 2560,以解決PWM抖動的問題。

二、 伺服馬達角度不精確

為了保持機械手臂的價格低廉，我們決定從演算法來增加機械手臂的精準度。因機械手臂的抖動和舵機一次能轉的角度間隔大，使用Adam時，常因抖動而使他的動量項偏移，而使機械手臂偏離目標而失敗，所以最後我們使用較簡單且穩定的SGD作為校準的演算法。

三、 手臂未達六自由度

機械手臂能控制末端位置的自由度時為五個(三個控制移動，兩個控制旋轉)，但想在工作範圍內完成任意姿態需要六個自由度，故我們限制了目標的取向，並使用Trac-IK以增加成功率。

四、 物體辨識與測距不精準

我們將原本的Kinect V1攝影機更換為解析度和距離較佳的RealSense™ D435，並在YOLO所辨識出的範圍再進行處理，使深度攝影機所傳回的物體目標點能夠更加精準。

五、 對避障所做出的應對

本研究在ROS平台中模擬避障，並讓電腦以100Hz傳輸角度值給予機械手臂，使其能依照計算的軌跡移動並避障，而非直接移動至指定目標。

捌、結論

一、 總結

本研究成功結合機械手臂、電腦視覺、深度學習，開發出了可自動辨識並抓取物體、自我校準的機械手臂。在手臂精準度方面，加上SGD演算法以後誤差能夠降低至原本的1/3至2/3，準確度高；在成功率方面，工作範圍內的物體都可以達到85%以上的成功率。整體來說，我們成功將機械手臂達到自動化，又保有相當不錯的準確性。

二、 本研究製作之機械手臂的優勢

1. 程式容易更改和開源：可自行修改或增加功能，完全開放。
2. 逆向運動學：使機械手臂能夠直接指定目標，而非控制每個舵機的角度。
3. 可自動化：機械手臂能透過電腦視覺達成自動化效果，相較人工操控更加精準。
4. 視覺系統：加裝了視覺系統使機械手臂有更多的可能性，如精度提升，自動抓取。
5. 自適應校準：透過演算法和電腦視覺校正機械手臂的精度。

三、 未來展望

1. 若能利用控制板取代電腦，能達到遠距操控、輕量化的效果
2. 將機械手臂裝載於自走車上，讓其擁有更大的運動範圍
3. 將自由度增至6DOF
4. 將自動化功能擴展使其能有效應用在工業用途，如揀貨、自動分類、行動不便輔助系統。
5. 將舵機更換為有回饋角度的舵機，以使用PID控制減少角度誤差。

玖、參考資料及其他

一、 SolidWorks to URDF Exporter，取自http：//wiki.ros.org/sw_urdf_exporter
二、 ROS，取自http：//wiki.ros.org/
三、 MoveIt，取自https：//ros-planning.github.io/moveit_tutorials/index.html
四、 R.Patrick Goebel(民106年)，ROS By Example 1&2。中山大學出版社
五、 Stephen Prata，C++ Primer Plus中文版。碁峰出版社
六、 YOLO ROS：Real-Time Object Detection for ROS，取自https：//github.com/leggedrobotics/darknet_ros
七、 Pinhole camera model，取自https：//en.wikipedia.org/wiki/Pinhole_camera_model