下表是各個 ROS 版本以及支援的作業系統：

ROS 版本	支援的作業系統
Kinetic Kame（LTS）	Ubuntu 16.04（LTS）與 15.10、Debian 8、OSX（Homebrew）、Gentoo、Ubuntu Arm
Jade Turtle	Ubuntu 15.04、14.10 與 14.04、Ubuntu Arm、OSX（Homebrew）、Gentoo、Arch Linux、Android NDK、Debian 8
Indigo Igloo（LTS）	Ubuntu 14.04（LTS）與 13.10、Ubuntu Arm、OSX（Homebrew）、Gentoo、Arch Linux、Android NDK、Debian 7

ROS Indigo 與 Kinetic 都是**長期支援（LongTerm Support，LTS）**的版本，也會與 Ubuntu 的 LTS 版一起發布。LTS 版的好處在於我們可得到的產品壽命與支援是最久的。

支援 ROS 的機器人與感測器

ROS 算是一個非常成功的機器人框架，全球的大專院校對其都做出了貢獻。由於其活躍的生態系以及開放原始碼特性，ROS 已可用於大多數的機器人中，並相容於各家主流的機器人軟硬體。以下是一些完全以 ROS 來執行的知名機器人：

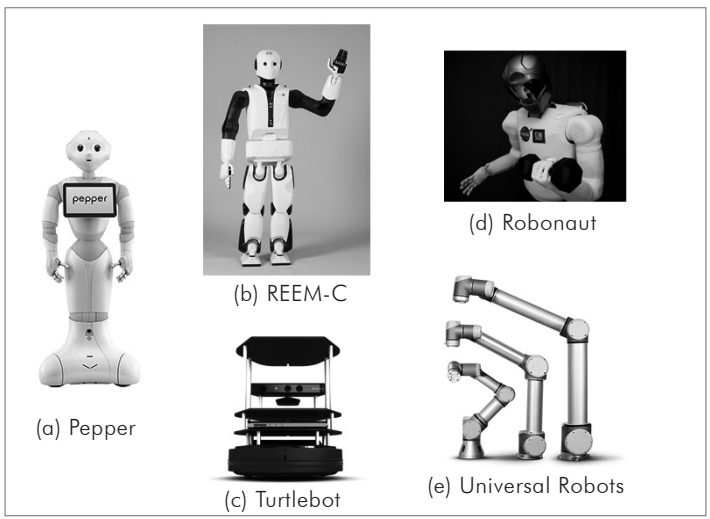

圖 1-4 支援 ROS 的知名機器人

支援 ROS 的所有機器人清單請參考以下網址：

http://wiki.ros.org/Robots

以下連結則是這些機器人的 ROS 套件：

- **Pepper**：http://wiki.ros.org/Robots/Pepper
- **REEM-C**：http://wiki.ros.org/Robots/Robonaut2
- **Turtlebot 2**：http://wiki.ros.org/Robots/TurtleBot
- **Robonaut**：http://wiki.ros.org/Robots/Robonaut2
- **Universal Robotic arms**：http://wiki.ros.org/universal_robot

下圖是支援 ROS 的常用感測器：

圖 1-5 支援 ROS 的常用機器人感測器

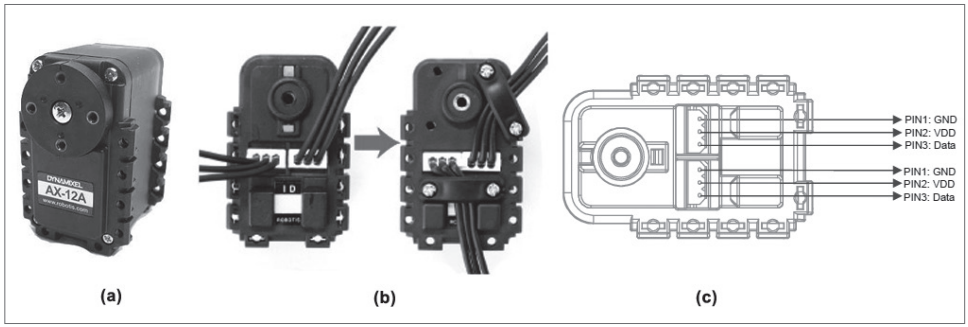

圖 2-6 AX-12A Dynamixel 伺服機與接線圖

與其他 RC 伺服機不同的是，AX-12 是款智能致動器，具備微控制器來監控伺服機狀態，還能自由調整所有參數。它具備了減速齒輪箱，並且馬達轉軸上已經裝了一個擺臂，您要在這個擺臂上額外加裝什麼東西都可以。伺服機背面還有兩個連接埠。

每一個連接埠都有 VCC、GND 與 Data 三隻腳位。Dynamixel 的連接埠採菊鍊（daisy chained）形式，因此伺服機可以彼此串接。以下是 Dynamixel 與 PC 的接線示意圖。

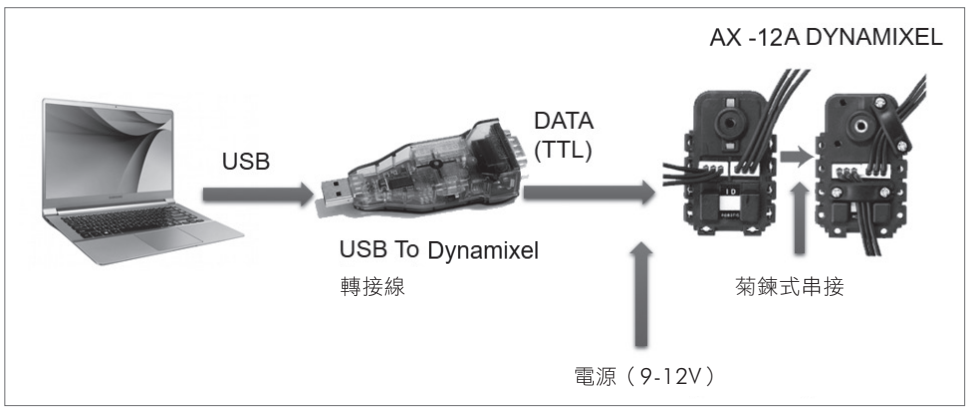

圖 2-7 AX-12A Dynamixel 伺服機與接線圖

- image_geometry：本套件會以幾何方式來解析影像。本節點可進行攝影機校正、影像矯正等作業。

我們在上述兩套件中比較常用的是 cv_bridge。透過 cv_bridge，臉孔追蹤器節點就能把來自 usb_cam 的 ROS 影像訊息轉換為 OpenCV 所需的 cv::Mat 格式。轉換為 cv::Mat 之後，就能用 OpenCV 的 API 來處理攝影機影像。

以下是 cv_bridge 在專題中的功能方塊圖：

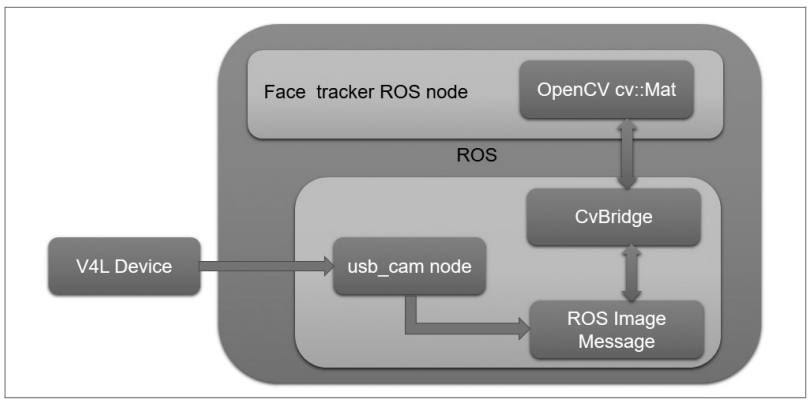

圖 2-14 cv_bridge 在本專題中的角色

cv_bridge 在此是介於 usb_cam 節點與臉孔追蹤節點兩者之間。下一段會深入討論臉孔追蹤節點。在此之前，如果您對於運作方式更有概念會比較好。

另外還要用到的套件是 image_transport[4]，它可在兩個 ROS 節點之間發送影像訊息。ROS 普遍將這個套件用於訂閱與發送影像資料。只要運送一些壓縮技術，本套件就能在低頻寬的條件下來發送影像。本套件已安裝在 ROS 的桌面完整版之中。

OpenCV 與 ROS 介面就介紹到這。下一段我們要開始操作本專題的第一個套件：face_tracker_pkg。

4 http://wiki.ros.org/image_transport

操作 ROS 的臉孔追蹤套件

我們已經把 face_tracker_pkg 套件加入 / 複製到工作空間了，也介紹了本套件的一些重要的相依套件。現在要來認識這個套件到底能做到什麼！

本套件包含了一個名為 face_tracker_node 的 ROS 節點，可以使用 OpenCV API 來追蹤臉孔並把臉孔的質心數值發布到指定主題。以下是 face_tracker_node 運作方式的功能方塊圖：

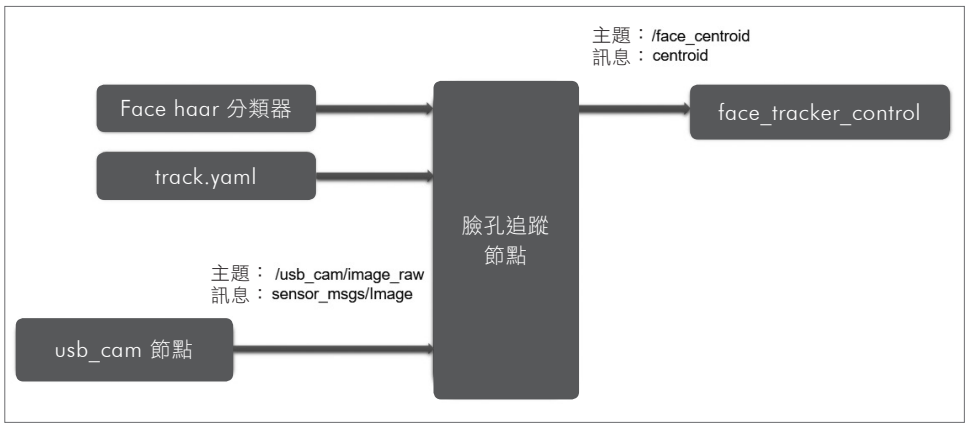

圖 2-15 face_tracker_node 功能方塊圖

現在來討論與 face_tracker_node 節點有連結的東西。其中 face haar 分類器這一段您可能會覺得有點陌生：

- face haar 分類器：基於 Haar 特徵的梯級分類器是用於偵測物體的機器學習方法。這個方法是由 Paul Viola 與 Michael Jones 於 2001 年的研究 [*Rapid Object detection using a boosted cascade ofsimple features*] 中所提出。本方法會運用 positive andnegative 影像來訓練梯級檔案，訓練好之後就能用這個檔案來偵測物體。
  - 我們會運用附於 OpenCV 原始碼中的一份已經訓練好的 Haar 分類器，放在 OpenCV 的 /data 資料夾中：https://github.com/opencv/opencv/tree/master/data。您可根據所需換成您所要的 Haar 檔。在此要用到的是臉孔分類器，這個分類器實際上是一個包含臉孔特徵標籤的 XML 檔。一旦

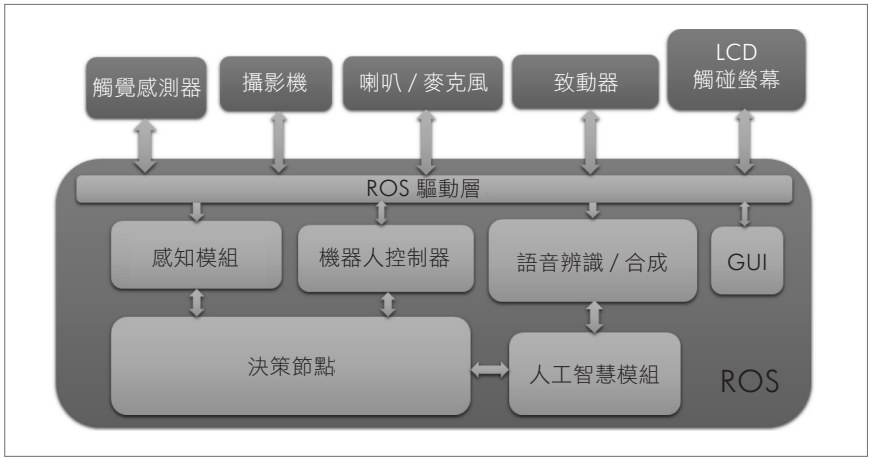

圖 3-2 常見的社交機器功能方塊圖

機器人通常會具備觸覺感測器、攝影機、麥克風與觸碰式螢幕，另外還裝致動器來做出某些動作。這類致動器可讓機器人得以帶動其頭部或身體。可四處移動的服務機器人甚至會有額外的馬達來導航呢！

在軟體區塊中，您可以看到能夠處理攝影機資料並從畫面中找出特定物體的感知模組、語音辨識 / 合成、各種人工智慧模組、用於控制各種致動器的機器人控制器模組、可整合所有感測器資料並決定下一步該怎麼做的決策節點。ROS 的驅動層可讓 ROS 去存取所有感測器與致動器，它的圖形化介面則可透過 LCD 面板作為互動式的視覺化介面。

本章要運用人工智慧技術來實作語音辨識與合成，可與我們人類一樣透過語音與文字來溝通。機器人的回應要愈貼近人類愈好。

我們要使用 AIML（人工智慧標示語言，Artificial IntelligenceMarkup Language）語言來開發一個簡易的 AI 聊天機器人，後續還能整合到社交機器人中。

現在來看看如何開發這類互動式機器人所需的軟體，先從開發軟體所需的東西開始吧！

在 ROS 中建立一個 AIML 機器人

上一段認識了 AIML 的各種標籤，以及如何運用 PyAIML 模組來操作這些標籤。現在要看看如何使用 ROS 作一個互動式的 aiml 機器人。下圖是這台互動式機器人的完整功能方塊圖：

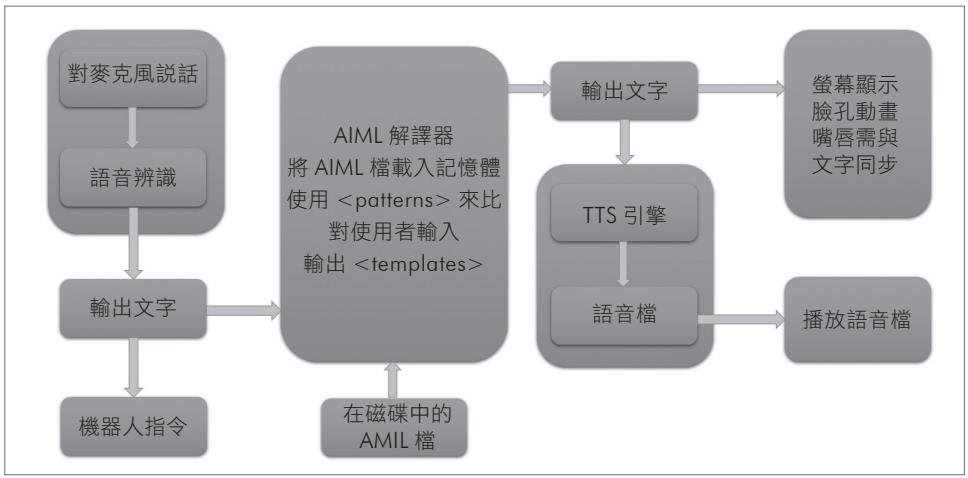

圖 3-4 互動式 AIML 機器人

以上是整體系統的運作方式。使用者說話的語音會透過 ROS 的語音辨識系統轉為文字，接著會丟給 AIML 引擎或變成機器人指令。機器人指令是用於控制機器人的特殊指令。如果得到的文字並非機器人指令的話，就會丟給 AIML 引擎並根據其資料庫給出一個智能回覆。AIML 解譯器的輸出則會透過文字轉語音模組轉為語音。不僅可由喇叭聽到這個語音，螢幕上的機器人臉孔也會跟著語音同步動起來。

本章主要是討論如何使用 ROS 來處理 AIML 與 TTS，您也可以在 ROS 網站找到一些關於語音辨識的參考資料。

Amazon 打算把自家倉庫中從貨架上挑選與放置物體這件事完全自動化。他們打算採用如上圖中的機器手臂把物品從貨架上拿下來。當機器人收到指令，要去拿取指定物體並放入籃中之前，它得先知道物品的位置對吧？那麼機器人要如何知道物體的位置呢？它應該需要某種 3D 感測器對吧？

另一方面，軟體端應該要有一些能辨識各種物體的演算法。機器人一定要在辨識完成之後才能得知物體座標。偵測座標需要與視覺感測器有一定的相對關係，這樣才能轉換為機器人端效器的座標，再讓機器手臂前端移動到物體位置。但在到達物體位置之後，機器人又該做什麼呢？它要抓取物體並放入籃中，這件事看起來不難，但也沒像我們想的這麼簡單就是了。下圖是機器手臂、端效器、Kinect 視覺感測器與物體各自的座標系統：

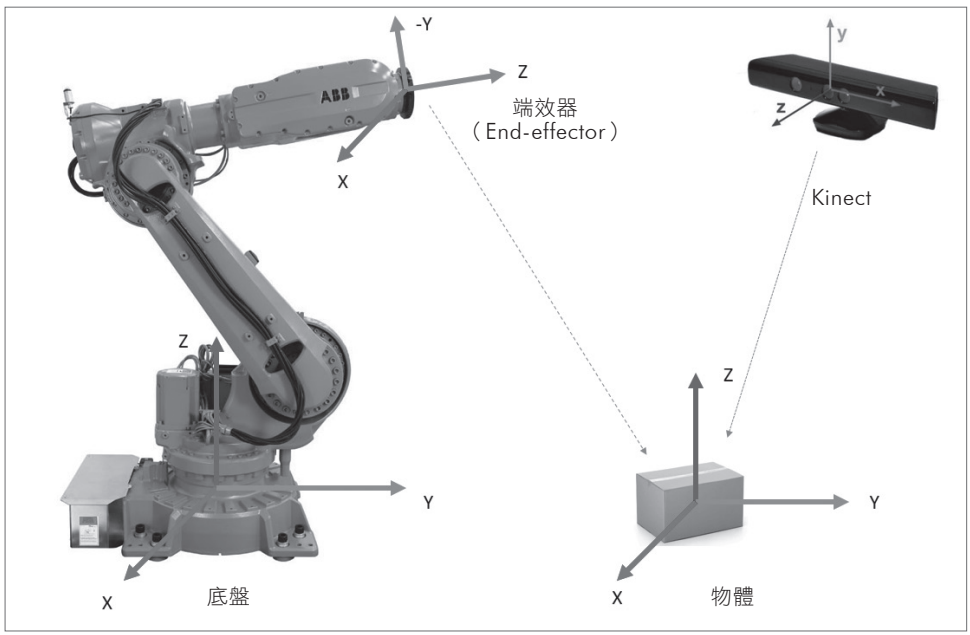

圖 6-2 各元件的座標系統

下圖說明了 Android 裝置與 ROS 機器人之間的通訊方式。下圖中的 Android-ROS 範例程式可透過 Android 裝置來遙控機器人。每個 Android 程式都繼承了 **RosActivity** 來取得 Android-ROS 介面，這樣才能在程式中取用 ROS 的各種 API。後續會深入介紹這些 API。

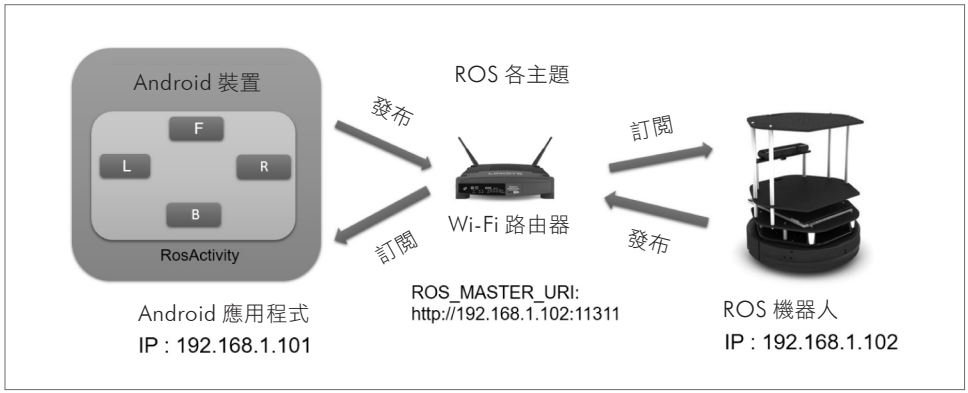

圖 8-22 Android–ROS 之 teleop 介面

Android ROS 函式庫的核心後端就是 RosJava[4]，也就是在 Java 中的 ROS 實作。另外也有使用 RosJava API 所製作的 Android 核心函式庫[5]。我們可以運用 Android-ROS 的各種 API 來建立 ROS 節點與 ROS master，但相較於正規的 C++/Python 之 ROS API，Android-ROS 的功能就少一點了。

那麼，Android-ROS 介面的重要性在哪呢？主要原因在於 Android 裝置可比做一台具備了所有必要感測器與周邊的迷你電腦。雖然 Android 裝置本身就能當作一台機器人了，但如果有了 ROS 介面，我們就能附加像是導航、地圖繪製與定位等高階功能來讓機器人更厲害。現今的 Android 裝置都有非常高解析度的照相機，所以我們甚至能透過 ROS 介面來處理影像。

4 http://wiki.ros.org/rosjava

5 http://wiki.ros.org/android_core

繪製機器人的 3D 模型

任何 3D CAD 軟體都可以繪製本專題機器人的 3D 模型。無論是 AutoCAD、SOLIDWORKS 和 CATIA 等廣受歡迎的商用軟體，還是 Blender 等免費軟體都可以。您可以依照自己的需求來設計模型。以下是我用 Blender 繪製的 3D 機器人模型。透過 3D 模型，我們不用真的做出硬體就可以調整機器人的設計直到完美為止，我們甚至還能用這個模型來模擬機器人的行動。下圖即為使用 Blender 繪製而成的 3D 模型：

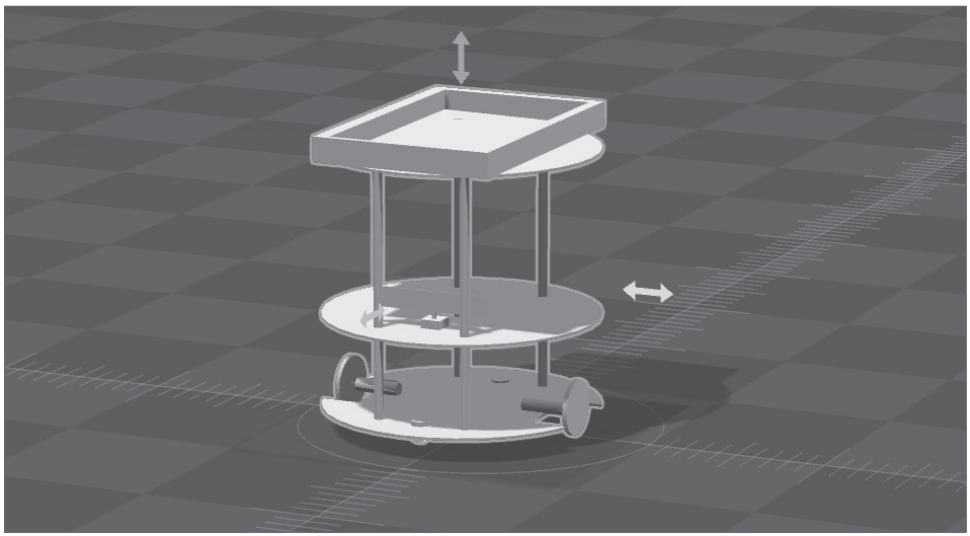

圖 9-7 3D 模型

在 chapter_9_codes/chefbot 檔案夾中有更多資訊。

在 Gazebo 上執行機器人模擬

機器人模型畫好之後，接著就是要模擬機器人的行為了。通常模擬上都會使用理想化的參數，但在真實操作中大多需要修改。我們可以用 Gazebo 來模擬。在開始之前，了解一下差速機器人的數學模型會對我們有所幫助，數理說明能夠提供您更多關於機器人運作背後的原理。另外，本章會使用既有的控制器，不會從頭開始實作。

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}$$

圖 9-9 差速驅動機器人的正向運動學模型

以下為上述方程式中未知的變數：

$R = l/2\ (nl + nr)\ /\ (nr - nl)$

$ICC = [\ x\text{-}R\ \sin\theta,\ y + R\ \cos\theta\ ]$

$\omega\delta t = (nr - nl)\ step\ /\ l$

nl 和 nr 為左右輪的編碼器。 l 代表輪軸長度，而 step 為編碼器每數一下時車輪行經的距離。

ICC 是瞬時曲率中心（instantaneous center of curvature），代表機器人車輪旋轉時的共同中心點。

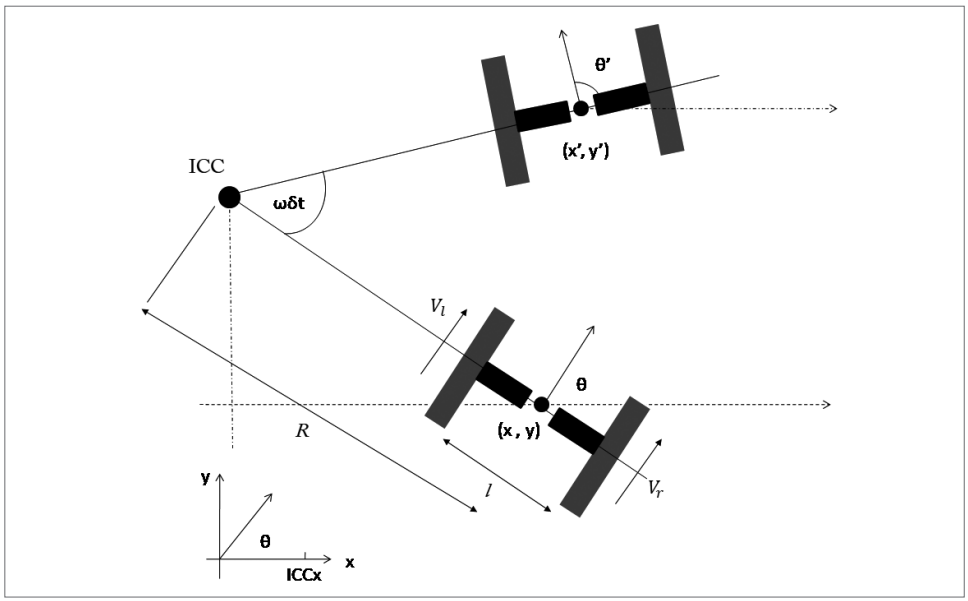

圖 9-10 差速驅動的正向運動學示意圖

請輸入以下指令以啟動定位：

```
$ roslaunch chefbot_gazebo amcl_demo.launch
map_file:=/home/<user_name>/hotel_world.yaml
```

這個指令將載入之前存好的地圖和 amcl 節點。輸入以下指令便可開啟 Rviz 以視覺化呈現機器人：

```
$ roslaunch chefbot_bringup view_navigation.launch
```

現在，我們可以讓機器人自動導航了。請點擊 2D Nav Goal 再點擊地圖來指定目的地。設好位置後，機器人便會試著由起點移動到目的地，如下圖：

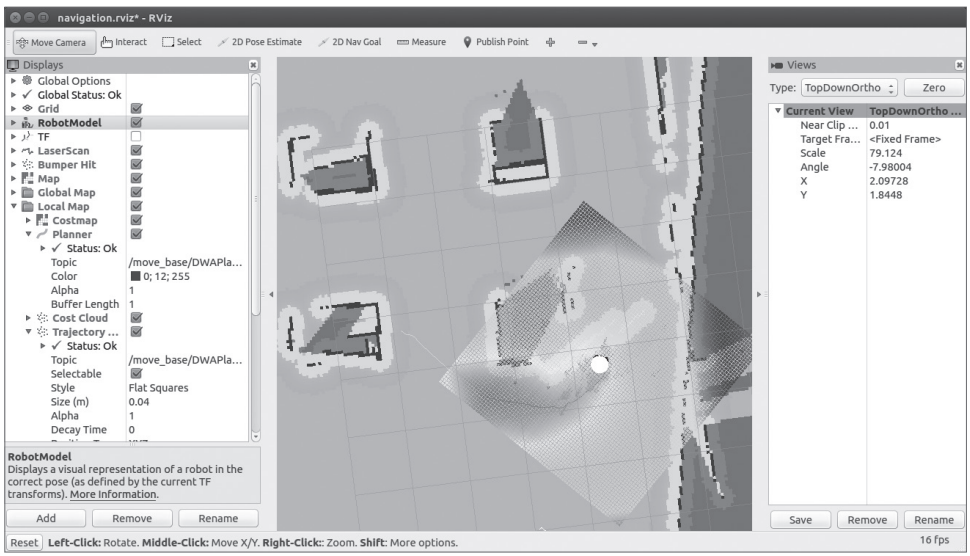

圖 9-15 透過 AMCL 粒子視覺化的自動導航

恭喜您！您已經成功地模擬出一台機器人，並在模擬器上完成了自動導航。現在，要來看看如何打造真正的機器人硬體還有程式了。