國立交通大學

多媒體工程研究所

碩士論文

模擬賽車遊戲中玩家個別差異之分析

The Analysis of Inter-player Difference in a Simulated Car Racing Game

研究生：邱俊予

指導教授：王才沛　教授

中華民國　一百年九月

模擬賽車遊戲中玩家個別差異之分析

The Analysis of Inter-player Difference in a Simulated Car Racing Game

研 究 生：邱俊予　　Student：Chun-Yu Chiu

指導教授：王才沛　　Advisor：Tsai-pei Wang

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Sep 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年九月

模擬賽車遊戲中玩家個別差異之分析

學生：邱俊予　　指導教授：王才沛

國立交通大學多媒體工程所

摘要

這篇論文的內容，是研究玩家在 TORCS 賽車遊戲平台上的駕駛行為。文章會從介紹研究玩家行為的原因與目的開始敘述，接著介紹 TORCS 賽車遊戲架構，以及一些 TORCS 相關的研究和其他人對賽車玩家行為的研究。每個人在駕駛賽車的行為上，都會有一些差異存在，為了研究每個玩家之間的差異，這篇論文提供一些比較各個玩家之間差異度的方法，並且用統計的方式建構出屬於一個玩家的個別模型，最後再用玩家模型做駕駛員預測實驗。

The Analysis of Inter-player Difference in a Simulated Car Racing Game

Student：Chun-yu Chiu　　Advisor：Tsai-pei Wang

Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

Abstract

In this thesis, we research the driving behavior of players in the TORCS racing game platform. At first we introduce the reason and purpose to research player behavior, and then we introduce the TORCS racing game architecture, some TORCS-related researches and other researches on the player behavior. There are driving behavior differences between each players, this thesis provides some methods to compare different players, and construct statistical individual player models, then use player models to predict the driver of racing trajectories.

誌謝

能夠完成這篇論文，首先要感謝我的指導老師王才沛教授的細心指導，還要感謝林俞丞、魏良佑、蘇裕傑三位同學的幫助，另外要感謝參加賽車資料收集實驗的學弟楊堡評和同學劉文皓，最後要感謝我的家人，讓我能夠無後顧之憂的完成這篇論文。

目錄

圖例

表格

第一章 簡介

1.1 研究目的

電子遊戲是現代最流行的娛樂項目之一，從遊樂場中的大型機台，到家用版的小型家用主機，從桌上型電腦到現在新型的平板電腦或智慧型手機，在這些平台上都擁有大量的電子遊戲產品。遊戲的類型也是五花八門，分成各種類型的遊戲，每種遊戲的遊玩方式以及遊戲目的都不相同，也都有各自的支持玩家群。其中一種分類為競技類遊戲，這種遊戲通常是讓玩家擁有一個屬於自己的角色，其他的玩家也會擁有相同或類似的各自角色，比賽開始之後，每個玩家就在同一個環境下，依照某種事先規定好的規則，靠著操縱自己角色的技巧，以達到獲勝目標為目的行動。這種類型的遊戲，就和平常的體育球類比賽的概念是相同的，事實上，大部分的體育項目都已經被做成電子遊戲，差別只在於真實的體育比賽是用自己的身體去和其他人比賽，通常都需要一定的天分與長期努力的訓練才能達到不錯的成績，而電子遊戲競賽是可以讓玩家用較輕鬆且容易上手的方式操縱遊戲中的虛擬角色，讓遊戲中的角色和其他玩家的角色比賽，體驗競技的樂趣。

競技類型的遊戲，有一個重要的基本要求，就是必須要有比賽的對手才能進行遊戲，通常在遊戲的設計上，會做成玩家和玩家比賽的模式，早期的單機版遊戲通常只跟認識的朋友一起玩，對戰的對手組合比較單調，為了能夠讓玩家能夠容易找到對手，現在的競技遊戲都傾向能夠從網路上搜尋對手並進行連線對戰。另外一個解決的方法是由電腦 AI 來操縱沒有玩家控制的角色，這樣可以滿足一些需求，例如單純想在和人對戰前先練習熟悉環境。或是因為在冷門時段遊玩而導致即使連上網路也找不到對手的情況。

由電腦 AI 操縱的非玩家控制角色，被稱為 Non-player character(NPC)，因為是由 AI 程式控制的關係，通常行為上會和平常的人類玩家有一段差距，而且這一點在人類玩家和 NPC 一起玩的時候可以察覺出來。NPC 的行為常會使用對遊戲勝

利目標的最佳化演算法，但這會造成 NPC 的行為單調缺乏變化，而且對於意料外的突發狀況缺乏反應能力。理論上 AI 程式應該要擁有能夠應對所有情況的能力，但是常會因為設計時沒有考慮到一些因素或是人類玩家故意製造出平常遊戲時不會出現的狀況，而導致 NPC 做出不適當的反應，有些玩家甚至會利用這種 bug 輕鬆的打倒 NPC。

電腦 AI 的強弱程度和遊戲的類型有很大的關係，在能夠精準的算出必勝策略或最佳化演算法的遊戲中，而行動時對手的行為也能夠精確計算的情況下，AI 是具有相當的優勢的，最經典的例子就是超級電腦「深藍」曾經在西洋棋比賽中擊敗世界棋王，但相反的如果遊戲變化度和隨機性較高，規則上的限制少，玩家可以做出較為複雜的行為表現，這樣 AI 因為要考慮的因素多，設計上也會困難許多，而且通常操縱的 NPC 會比普通人類玩家弱。因為這些原因，有一種設計 AI 的方式是研究人類玩遊戲的方式，然後讓 AI 模仿人類的行為模式，這種方法稱為「human imitation」，在人類玩家玩遊戲時收集遊戲過程人類對遊戲狀況的反應，然後用這些資料訓練 AI，讓 AI 在遇到相似情況時能夠做出接近人類的反應，而且可以讓多個不同的 AI 分別去學習不同人類的遊戲資料，製作擁有不同個性和風格的 AI。

這篇 paper 的目的是研究人類在賽車遊戲上的行為模式，我們使用 TORCS(The Open Racing Car Simulator)做為研究的工具。TORCS 就像他的名字所描述的一樣，是一款開放原始碼的模擬賽車遊戲，它的遊戲內容是模擬真實的賽車競賽，遊戲中還模擬了 F1 賽車比賽模式和路旁的修車補給區域，遊戲中提供了數十種車輛和數十種賽道。 TORCS 比較特殊的是它並不注重於人類玩家之間的競賽，主要是用來做 AI 與 AI 之間的競賽，它開放原始程式碼，讓其他人能夠設計自己的 AI，並且每隔一年舉辦 TORCS 的比賽，參賽者全部都是用自己的 AI 參賽,詳細規則可以參照[1]。TORCS 提供了一個相當不錯的比較平台，在 TORCS 出現之前，對賽車控制的研究通常都是用研究者自己定義的賽車及賽道

模型，有些是用真實的模型遙控車和自製的賽道或路線，有些是用電腦裡的模擬賽車程式，但是因為每個研究之間標準不相同，所以就不容易加以比較或實作，因為 TORCS 開放程式碼，讓想研究賽車遊戲的人可以直接使用，不必再建造自己的賽車模型，近幾年已經有一些 paper 選用 TORCS 當做研究賽車駕駛的平台。

第二章 TORCS介紹與文獻縱覽

2.1 TORCS 環境基本架構

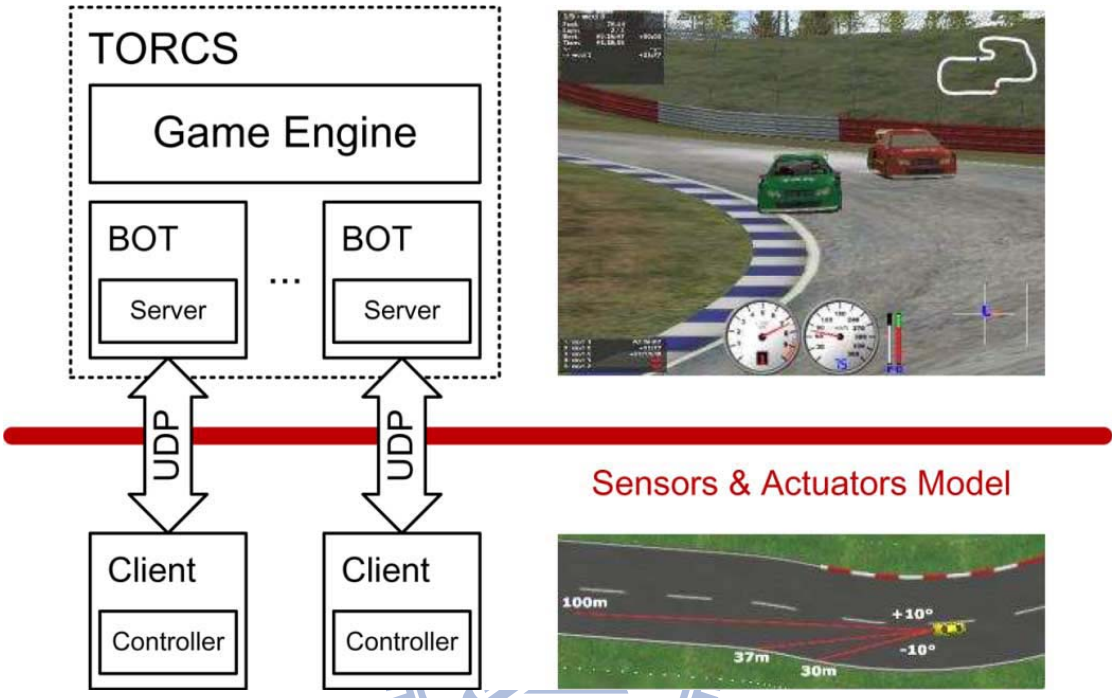


圖 2-1 TORCS 的 server-client 架構　圖片來源[1]

TORCS 的程式架構可以分成兩個部分：server 部分和 client 部分(圖 2.1)，client 部分為 AI driver 程式，server 部分連結主要的遊戲執行引擎，當一場賽車遊戲進行時，比賽中的每一台車輛都擁有一個各自的 client 連結到 server，遊戲引擎執行遊戲時，每隔 0.02 秒的間隔(稱為一個 game tick),server 會將現在的遊戲環境狀況傳給 client，並且等待 0.01 秒，client 中的 AI 程式根據現在的環境做判斷，傳送控制指令給 server，server 再根據收到的控制命令對賽車做控制，client 如果沒有在 0.01 秒之內做回應，就執行最近一次收到的控制指令。在一般的 TORCS 研究之中，不會對 server 做變動，只會去改寫 client 部分的 AI 控制程式，在 TORCS 競賽中，使用固定的 server 端，接上不同作者製作的 client 端，就可以進行不同 AI 之間的對戰。

表 2-1 和表 2-2 為 server 和 client 之間互相傳遞的變數:

表 2-1 TORCS server 環境變數

名稱	數值範圍 (單位)	說明
Angle	[-π,+π] (rad)	車頭方向和跑道中心線方向的夾角
curLapTime	[0, +∞) (sec)	在目前這一圈用掉的時間
damage	[0, +∞) (point)	車子受到的傷害值(數值越高表示傷害越大)
distFromStart	[0, +∞) (m)	車子和起跑線之間的距離
distRaced	[0,+∞) (m)	開始遊戲之後車子走過的距離
Focus	[0, 200] (m)	一組可以讓 AI 操作的距離偵測器,總共有 5 個,每個偵測器角度間隔 1 度,可以偵測到車子在特定方向上和賽道邊界的距離,由 client 指令決定真測的方向,但是有一些使用上的限制,focus 偵測器每秒鐘只能使用一次,在車子超出賽道範圍和 focus 偵測方向超出可用範圍(車子前方 180 度)的情況下,會得到不可靠的值.
Fuel	[0, +∞) (1)	車子的剩餘汽油量
Gear	{-1, 0, 1, …, 7}	車子的排檔，-1 表示倒車，0 表示空檔，1~7 為前進排檔
lastLapTime	[0, +∞) (s)	跑完上一圈賽道用掉的時間
opponents	[0, 200] (m)	包含 36 個對手偵測器,每個偵測器涵蓋了車子周圍 π/18 的角度,偵測距離 200 公尺,每個偵測器會傳回偵測範圍之中最接近的對手車輛的距離.

racePos	{1, 2, …, N}	目前在比賽車輛中的位置排名，N 代表賽車的數量
Rpm	[2000,7000] (rpm)	引擎的轉動速度
speedX	(-∞, +∞) (km/h)	車子在車頭面對方向上的速度
speedY	(-∞, +∞) (km/h)	車子在和車頭垂直方向上的速度
speedZ	(-∞, +∞) (km/h)	車子在高度軸上的速度
Track	[0, 200] (m)	包含 19 個距離偵測器,從車子前方的正右方(-π/2)到正左方(π/2)的範圍之內,每隔 π/18 有一個偵測器,可以偵測到車子在固定的方向上和跑道邊界的距離,在車子超出邊界的情況下,偵測到的值不可靠.
trackPos	(-∞, +∞)	表示車子和跑道中心線的距離，已經用賽道的寬度做 normalize，-1 表示賽道右側邊界，+1 表示賽道左側邊界，大於+1 或是小於-1 的值代表車子超出了邊界。
wheelSpinVeil	[0, +∞] (rad/s)	四個車輪的轉動速度
Z	[-∞, +∞] (m)	車子和賽道表面的距離

表 2-2 TORCSclient 控制指令變數		
名稱	範圍	說明
Accel	[0,1]	油門控制指令(0 代表不加油門,1 代表最大油門)
Brake	[0,1]	煞車控制指令(0 代表不煞車,1 代表最大煞車)
Clutch	[0,1]	離合器控制指令(0 代表無動作,1 代表最

		大)
Gear	{-1, 0, 1, …, 7}	排檔控制指令
Steering	[-1,1]	方向盤控制指令(-1 代表最大右轉,1 代表最大左轉)
Focus	[-90, 90]	指定 focus 偵測器的方向
Meta	0, 1	0 沒有作用,1 代表要求 server 重新開始遊戲

2.2 TORCS 相關的研究論文

在 TORCS 平台上已經有一些對賽車控制 AI 相關的研究，因為有許多研究是為了在 TORCS 競賽上勝利，大部分現有研究是關於如何使用某些演算法去製作跑得快並能夠妥善處理過彎動作的 AI 程式，或是能夠讓賽車 AI 最佳化的方法。在[2]中介紹一個 AI 的各種控制方式，而且使用了 fuzzy 控制的方法，根據收到的賽道距離探測器，決定駕駛的速度。[3]中也是用 fuzzy set 作賽車的控制，而且它使用基因演算法對 fuzzy 控制模型做調整，自動找出最佳化的控制組合，另外[4]使用 NEAT(NeuroEvolution of Augmenting Topologics) 演算法製作 AI，[5]和[6]也是使用演化是計算的方式對 AI 做最佳化。

少數一些論文則是對賽車駕駛的特殊行為做研究，在[7]中用一種名叫 Q-learning 的演算法學習演算法，研究多人賽車遊戲中賽車間互動行為。這篇 paper 對於跑道上兩台賽車之間的位置，研究適當的超車與阻擋方式。

另外有一些針對 TORCS 本身環境進行的研究，因為在 TORCS AI 競賽中賽車控制器只能夠得到賽車附近的一部份資料，在[8]中研究一種根據距離探測器偵測到的賽車和賽道邊界距離，推算出真實賽道形狀的方法，讓 AI 能夠擁有更多的資訊，做出更適當的駕駛判斷。

2.3 TORCS 平台上的 human imitation 研究

近年來有一些在 TORCS 平台上做 human imitation 相關的研究，使用人類玩家玩過 TORCS 的遊戲紀錄，學習人類的行為。 TORCS 本身並沒有記錄遊戲過程的功能，所以必須更改 TORCS 的程式碼，在不影響遊戲進行的條件下增加收集玩家行為資料的功能。在前面說的 client 部分是由電腦的 AI driver 決定控制指令，但是在 TORCS 遊戲裡內建了一個 human client，將決定控制命令的程式連接到電腦的各種輸入設備，如滑鼠、鍵盤、遊戲手把、方向盤等，讓人類玩家可以做為一個 client 參加遊戲。因為 TORCS 的 server-client 架構特性，每經過 0.02 秒的間隔 server 和 client 會互相傳遞一次訊號，這可以視為玩家對遊戲環境產生一次反應。想要收集到這些反應的資料，可以把記錄資料的程式加到每次 server-client 互動時會執行的部分，將當時 server 傳送的環境資料和 client 傳送的控制指令記錄下來，這樣可以在遊戲中每 0.02 秒得到一組玩家行為資料。

前面提到 server 傳給 client 的環境變數，是 TORCS 競賽時使用的版本，這些資料只讓 client 得到自己車子附近的一部分資訊，大部分是自己的賽車狀況，偵測外部環境的距離偵測器也都帶有距離上限和其他限制，希望 client 用當下所處的情況判斷需要做出的反應。事實上，在完全開放程式碼的版本中，我們在遊戲進行時 client 可以得到完整的賽道資訊，包括整個賽道的完整形狀、地面和圍牆的材質、以及其他對手車輛的位置等，在蒐集玩家資料時，可以把這些完整的整個比賽狀況一並記錄下來做研究。

使用這些玩家資料訓練一個 AI 程式，主要的步驟可以分成三個部分:

1.讀取環境變數做為 input：

input 的選用方式在每篇 paper 裡都不相同，因為可以選擇 TORCS 競賽用的 input，也可以自己改寫程式碼得到更多的賽道資訊，一般都會選擇車子附近的資料當作 input，因為距離太遠的跑道資訊對現在的駕駛行為影響不大.

2.經由 AI 做運算判斷處理：

AI 的運算處理步驟，會使用到某些學習演算法，利用玩家的遊戲紀錄去訓練 AI 的判斷模型，達到模擬人類行為的目的.

3.輸出控制指令做為 output：

Output 輸出方式可以分為兩種:直接式和間接式的控制指令。直接式的 output 是 AI 運算直接去算出控制的指令，算出加油門、煞車、轉向等數值。間接式的 output 是讓 AI 計算出帶有一些意義的控制目標，例如目標速度，目標魏志，目標方向等等，再由另外的控制程式去完成這些操作，以目標速度為例，AI 算出目標速度後，控制程式檢查現在車子的速度，如果比目標速度快就執行煞車的指令，如果比目標速度慢就執行加油門的指令。

以下為一些在 TORCS 上的 human imitation 研究:

在[9]之中使用 TORCS 競賽的 server 提供參數當作 input，但是並沒有全部用到，只使用以下的項目:

1. 車子的速度
2. 車子和跑道中心線的夾角
3. 車子的排檔位置
4. 車子的橫向(和車頭垂直的方向)速度
5. 馬達的轉速
6. 四個車輪的轉速
7. 19 個賽道邊界距離探測器

訓練 AI 時使用類神經網路的方式，模擬三組遊戲紀錄做比較，三組紀錄從不同的 client 收集而來，下面是三組紀錄的來源：

1. Human player： 人類玩家的賽車遊戲資料。
2. NEAT controller： 一個使用 NEAT 製作的賽車 AI 程式，這個 AI 是 2008 年 TORCS 競賽的冠軍。
3. Hand coded controller： 一個可以在不超出賽道邊界的原則下跑完全程的賽

車 AI 駕駛程式，不計較 AI 的速度成績。

Output 則使用直接控制的方法，輸出加油門、煞車、轉向、排檔指令。

在[9]的實驗結果之中，模擬人類玩家的結果並不好，製造出來的 AI 不能夠完整的跑完整個賽道，模擬第三組資料(不超出跑道範圍的 AI)時結果比較穩定。

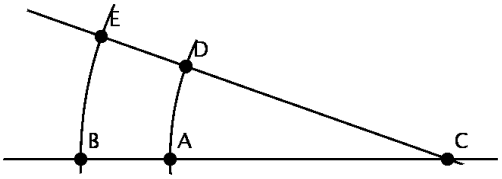


圖 2-2 車輪的路徑資料與賽車偏移角度關係圖　圖片來源[10]

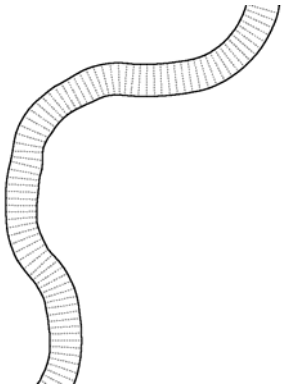


圖 2-3 由估計 segment 組成的賽道模型　圖片來源[10]

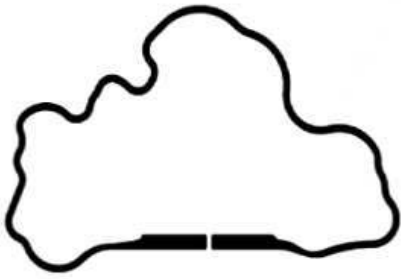


圖 2-4 TORCS 中一個賽道的形狀

圖片來源[10]

圖 2-5 模擬圖的賽道模型

圖片來源[10]

在 TORCS 競賽的規則中，每次會有之前未開放的新跑道當作比賽項目，增加比賽的公平性。而在比賽前允許參賽者在新跑道上試跑 100000 個 game tick，相當於大約 30 分鐘的時間，參賽者可以藉此調整自己賽車 AI 的參數。因為在

TORCS 競賽之中對環境偵測的限制，只能得到車子附近的環境資料，[10]的研究中提出了一種在比賽中建立出整個跑道模型的方法。先使用一個較單純穩定的 AI 順利的開過賽道一圈，然後每 0.02 秒紀錄一次車輪的移動距離，根據左右車輪的移動距離差計算行進的偏向角度(圖 2-2)，和計算行進的路線，以每 4 公尺為一個單位算出一個 segment，由多個 segment，組成一個完整的賽道模型(圖 2-3)，因為是估計出來的賽道，所以跟實際的賽道形狀會有些誤差,見圖 2-4 和圖 2-5。

在[10]之中使用這個模擬賽道模型做為 human imitation 的環境， input 的部分取用下列參數:

1. 賽車前方 50 個 segment 的偏向角度
2. 賽車後方 15 個 segment 的偏向角度
3. 跑道寬度
4. 下一個轉彎的方向(0 為左轉，1 為右轉，0.5 為直線前進)
5. 現在的轉彎方向(0 為左轉，1 為右轉，0.5 為直線前進)

在訓練 AI 的方法上使用類神經網路，使用間接式的控制，算出目標速度和目標位置，再用其他數學公式，計算賽車朝向目標前進應該用的控制指令數值。實驗的部分用製作出來的 AI 和模擬的人類記錄做比較，AI 比人類玩家大約慢了 20%~45%，他們認為 AI 最大的問題是在於當賽車超出跑道邊界時沒辦法像人類一樣，做出適當的回復跑道中心處理方法。

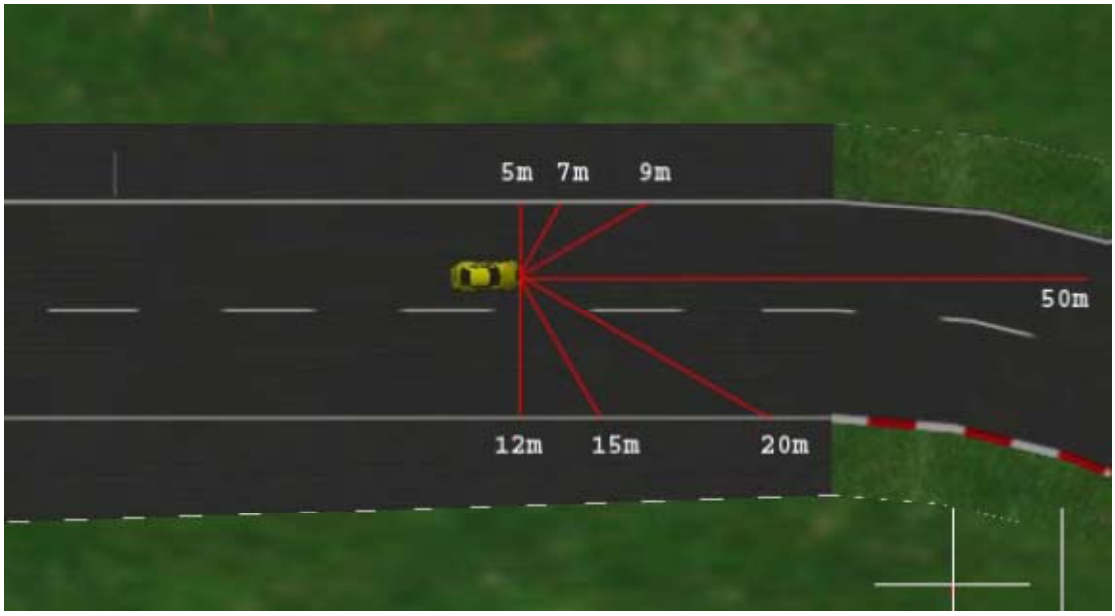


圖 2-6 Rangefinder sensor　　圖片來源[11]

圖 2-7 Lookahead sensor　　圖片來源[11]

在[11]之中有兩組 input 的偵測賽道環境方法，第一種是 rangefinder sensor(圖 2-3)，也就是在 TORCS 競賽中可以使用的距離探測器，第二種是 lookahead sensor(圖 2-4)， 把賽車位置前方的賽道取一段固定的長度，分割成許多等長的 segment，算出每個 segment 的彎曲程度(0 代表直線段，正數代表右轉，負數代表左轉)，將它串起來之後就得到一組 input 向量，例如上圖中的 lookahead sensor

可以得到[0,60,90,90,60,0,0,0]的結果。

AI 判斷的方式也分為兩種:第一種使用 k-nearest neighbor，接收到 input 之後，從玩家記錄中找出環境最相似的 k 組資料，然後將他們的反應行為取平均值，當作 output 的輸出結果。另一種則是使用類神經網路的方式訓練 AI。最後使用間接式的方法，輸出目標速度和目標位置(和跑道中心線的距離)做為 output。

這篇研究將製作出來的賽車 AI 放到 TORCS 賽道中做測試，以固定時間之內能夠跑的距離當作評分標準，並且和 inferno bot 做比較，inferno bot 是 TORCS 內建的 AI 程式中最快的一個，它可以使用完整賽道資料做最佳化駕駛。rangefinder sensor 的實驗結果不好，lookahead sensor 在某些場地可以做到比 inferno bot 慢 20%~30%的成績。

另外這篇 paper 還有對 lookahead 的偵測距離做研究，分別用 8 段 segment 和兩倍距離的 16 段 segment 做 input 取樣，比較製作出來的 AI 程式成績，結果發現取樣較多的 16 段 segment 這一組造成了 overfitting 的效果，使用某個跑道訓練出來的 AI，在同一個跑道上跑時 16 段 segment 的效果較好，但是拿到其他跑道上測試時 8 段 segment 的效果會比較好。

第三章　實驗方法

3.1　Checkpoint 取樣法

因為 TORCS server-client 程式架構的關係，在玩家玩 TORCS 賽車遊戲時收集到的玩家行為資料，是以 game tick 為單位，每隔 0.02 秒的時間對玩家的行為作取樣，而現在要介紹的 checkpoint sample 取樣法，則是對這些用時間取樣的資料在做一次空間上的取樣。取樣的方法如下：

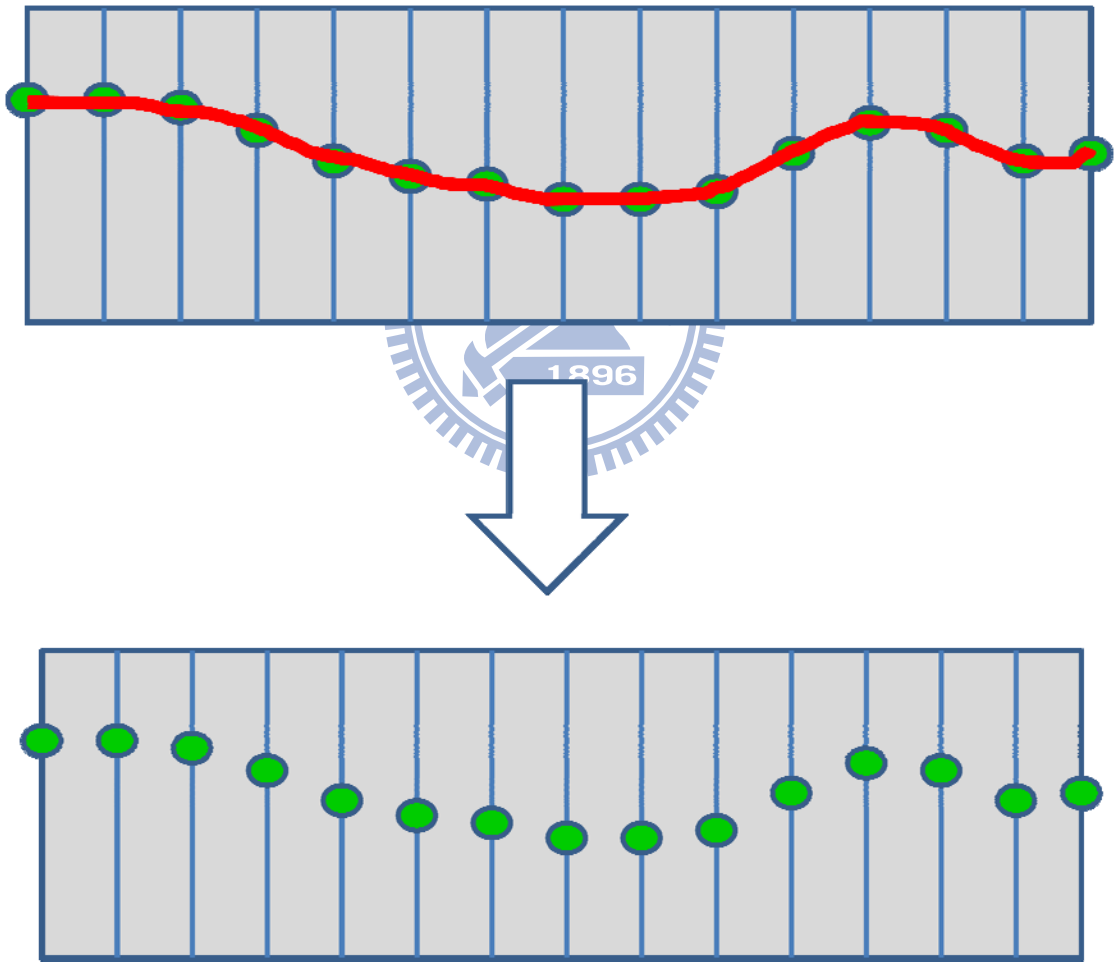


圖 3-1 checkpoint 取樣法

圖 3-1 中,長方形部分代表一段賽道，藍線表示 checkpoint，在賽道上每隔一段固定的距離設定一個 checkpoint，每個 checkpoint 都是跑道的一個橫切面。紅

線表示將每個 game tick 取到的資料位置點，依照時間的順序連起來，得到的一條代表賽車前進路線的軌跡線，綠色點表示藍線和紅線的交會點，也就是賽車通過每個 checkpoint 的位置，就是目標取樣點，將這些資料取出來，即是 checkpoint sample。因為在賽車的資料變數當中，一些較為重要的變數都會遵照物理引擎，做有連續性的變化，例如位置、速度、賽車方向等，所以這些 checkpoint 取樣點可以大略的表示整條軌跡的狀況，例如圖中的綠點連成的折線就和原本的紅線軌跡相近，這是在位置上的連續特性。我們可以用少量的 checkpoint sample 資料，代替原本收集的大量玩家資料。

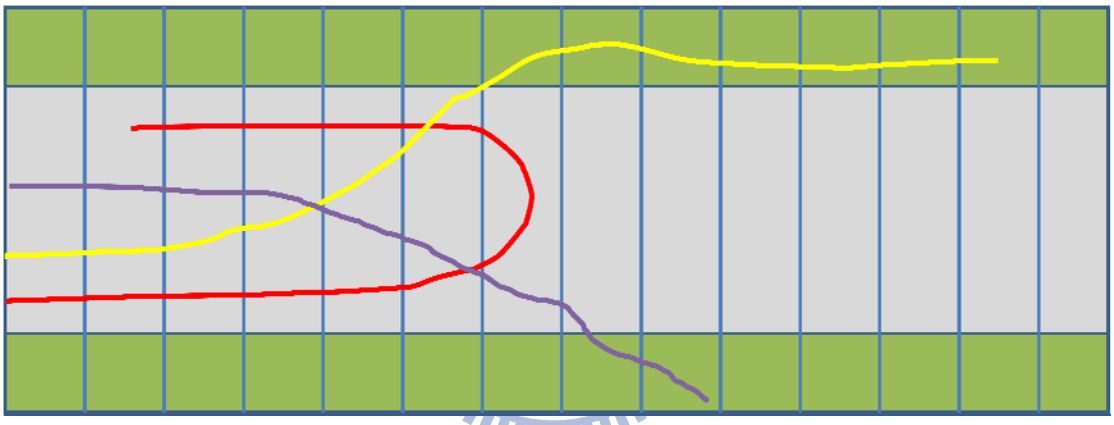


圖 3-2 三種失誤狀況下的賽車軌跡

人類在玩賽車遊戲的時候，不會像電腦一樣永遠用最佳化的方式控制，難免會發生一些失誤，會產生一些不適當的行為資料，在遊戲紀錄資料的研究和學習上，會產生一定的影響。checkpoint sample 的優點之一是對某些失誤狀況的反應不明顯，以圖 3-2 說明，灰色表示為賽道區域，綠色表示賽道外區域，鉛直線表示 checkpoint，紫、黃、紅三條線代表賽車的軌跡，一般最常見的失誤是像紫色線的情況，轉向控制不良而開到邊線外了，這種情況下，因為賽車方向和跑道中心線有蠻大的夾角，向跑道前方的速度小，又因為玩家的煞車動作和撞牆及草地的減速效果，失誤時通過的 checkpoint 不多，因為失誤而產生的停滯和調整回復動作會因為 checkpoint sample 而被淡化或過濾掉。會造成比較大影響的，是圖中

的黃線駕駛方式，即維持在界線外的狀況下前進，這種路徑常會發生在超出邊界但是前進方向正確，正在做調整回到中心賽道的狀況。紅色線則是嚴重的失誤，賽車往反方向前進，這可能會造成賽車經過同一個 checkpoint 兩次以上。這個問題不難解決，只要設定同一圈內 checkpoint 只有第一次經過的時候會作取樣，第二次經過就不做任何動作。

3.2 整圈賽道 checkpoint 差異度測試

在這個實驗裡，我們在每個賽道上，每隔一段相等的距離設一個 checkpoint，當玩家駕駛的賽車通過每個 checkpoint 時，記錄當時的賽車資料，記錄的資料有 4 種變數項目:賽車位置、速度、加速度、賽車方向和跑道中心線的夾角，玩家每跑完一圈就可以得到一組 lap data， data 的長度 L= 賽道長度/checkpoint 間隔，我們用平均的方式計算兩個 lap data 之間的差異度，設 lap_k 代表一組 lap data 中的第 k 筆資料(從一圈賽道中第 k 個 checkpoint 取到的資料)，這裡的資料可以帶入 4 種變數項目的其中一種做運算，兩組 lap data 的差異度就定義為公式(1)。

$$\text{dist}(\text{lapA}, \text{lapB}) = \frac{\sum_{i=1}^{L} \left| \text{data}_i^{\text{lapA}} - \text{data}_i^{\text{lapB}} \right|}{L} \quad (1)$$

我們想要知道每個玩家之間的差異性大小，因此把收集到的 lap data 依照收集來源的玩家分群，每個玩家有自己的一群 data，然後計算每群之間的差異度

設 M 為 A 玩家資料 groupA 的 lap data 個數

N 為 B 玩家資料 groupB 的 lap data 個數

lap data group 的差異度算法為公式(2)。

$$\text{dist\_group}(\text{groupA}, \text{groupB}) = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} \text{dist}(\text{lap}_i^{\text{groupA}}, \text{lap}_j^{\text{groupB}})}{M \times N} \quad (2)$$

另外我們想比較自己和自己 data 的差異度多大，也就是個人行為的穩定程度，定義為公式(3)。設某個玩家的資料群為 group，lap data 數量 N，

$$\text{dist\_group\_self}(\text{group}) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \text{dist}(\text{lap}_i^{\text{group}}, \text{lap}_j^{\text{group}})}{N \times (N-1)} \quad (3)$$

因為當 i=j 時比較的 lap data 是一樣的，差異值一定是 0，所以作平均時去除這種

狀況，只除以有用比較數 Nx(N-1)

3.3 賽道區段的 checkpoint 差異度測試

前面的 lap checkpoint 差異度測試，必須要用整個賽道為單位才能看出玩家之間的差異度，不容易理解玩家的差異之處，為了做更精細的研究，我們把整圈賽道分成許多區段，每個賽道區段為一段曲率半徑保持不變的賽道片段，使用一個區段為單位做細部的研究，一個區段的形狀可以簡單的用三個變數來表示：

1. 區段彎曲度
2. 跑道寬度
3. 區段長度

我們想研究玩家在某種賽道情況下的行為模式，但是很少有賽道區段形狀的三個參數完全相同的情況，因為玩家資料帶有隨機性，在取樣數少的情況下，做出來的玩家對賽道反應模型並不準確，所以我們用某些方法把形狀不同的賽道區段做疊合，讓它們也可以互相做比較。下面說明對三個變數的疊合方法：

彎曲度的疊合方法：

我們取曲率半徑的倒數作為區段彎曲度，右轉時再乘上-1.算出來的區段彎曲度會是(-∞，∞)之間的值，-∞時為最急右轉，0 為直線區段，∞為最急左轉。把區段依照彎曲度分成 9 個區間，(-∞, -0.018)、[-0.018, -0.012)、[-0.012, -0.006)、[-0.006, 0)、[0, 0]、(0, 0.006]、(0.006, 0.012]、(0.012, 0.018]、(0.018, ∞),也就是取 0(直線段)為中央區間，左右各取 3 個 0.006 的區間，剩下的部分依正負分到兩個區間。

跑道寬度的疊合方法：

因為我們收集玩家跑道時用的跑道，寬度相差不會太大，因此我們以跑道左

邊線為 1，右邊線為 0， 作為跑道寬度方向的座標值，把跑道區段做 normalize，將寬度調整成一樣。

跑道長度的疊合方法：

每一個區段的長度相當大，最短的不到 10 公尺(遊戲中的長度單位)，最長的可以達到幾百公尺，所以不適合用像跑道寬度的 normalize 方法，因此我們對區段中的玩家軌跡資料做觀察，根據軌跡在跑道區段中的特性做出了三階段模型，下文會有較詳細的說明。

3.4 賽道區段三階段模型

賽車在同一個區段內部行駛的時候，因為玩家會維持在相同的環境狀況下(跑道前進方向，寬度，彎曲度等)，玩家的賽車行為會變得較為穩定。想像在極端的狀況下，例如無限長的直線跑道，或是無限長的固定曲率彎道(假設一個完美一圈圓形的賽道，不設定終點跑起來就像是無限長的彎道)，玩家會因為環境條件沒有變化，把賽車維持在他認為適應這個環境最佳的狀況下，像是賽車位置維持在賽道中間(直線跑道)或是內側位置(彎道)，速度也會調整到最快速度(直線跑道)或是以能平衡離心力為條件的最大速度(彎道)。

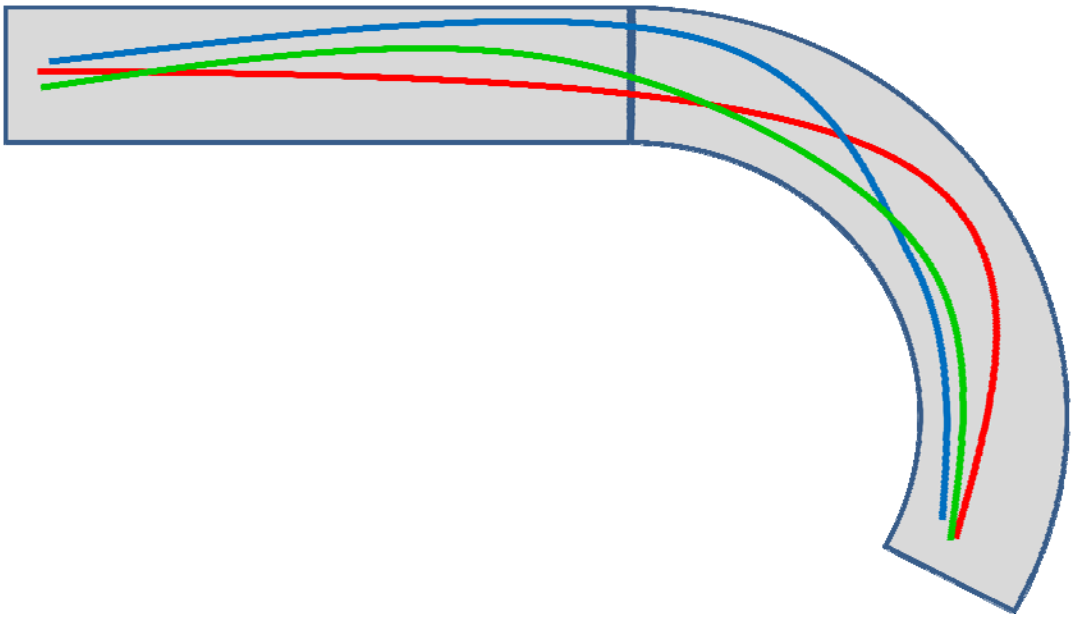


圖 3-3 不同玩家在賽道區段切換處的不同駕駛策略路徑

在賽道區段與區段交接的部分，也就是跑道的轉折處，玩家就沒有一個特定的規則可以依循，會用自己的方式去判斷從交接點前一個區段到後一個區段的變化，做出自己的反應行動(圖 3-3)。玩家在這個部分的駕駛行為，因為駕駛時須要考慮的環境變數較為複雜，不同玩家之間對於過彎時最佳化的路徑及策略看法並不相同，而且完成自己的賽車控制策略困難度比單純區段時較高，容易偏離自己的計畫路線。比較情況下這些情況導致區段之間交接處玩家資料比較不穩定，變化較為複雜。

以一個賽道區段為單位來看，區段的前端是和另一個區段的交界區，中間區域是維持這個區段的環境，而尾端也是和另一個區段的交界區。玩家在前端時剛離開上一個區段，正在做調整的動作，讓賽車適應這個新的賽道區段。玩家在區段的中間部分，因為離其它區段有一段距離，不會受其他區段干擾，所以會有比較穩定的行動。賽車到了區段的後端，因為發現了下一個轉折點，所以會改變駕駛行為，做出預備下一個區段的動作。

3.5 賽道區段前段和後段取樣範圍

我們因為上面的原因，把賽道區段分為三段，這就是三階段區段模型。假設賽道的前段調整和後段調整須要有一個固定的長度緩衝，因為賽道區段的總長度不同，所以有下面三種情況。

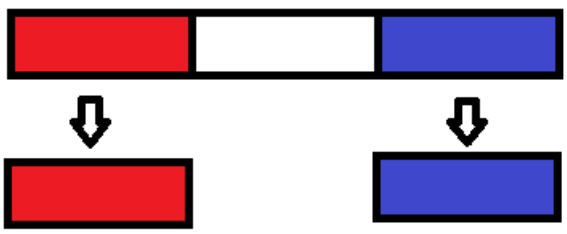


圖 3-4 長賽道區段分解圖

在長度比較長的賽道區段(圖 3-4)，取前方固定長度為前段，後方固定長度為後段，剩下的部分就是中間段。

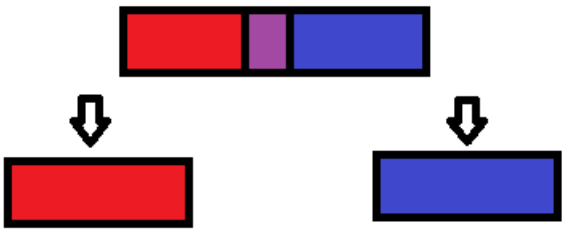


圖 3-5 中賽道區段分解圖

中等長度的賽道區段(圖 3-5)，仍然是採用前端和後端固定長度的部分作前段和後段，中間重疊的部分會被前段和後段重複取樣，但是因為長度不夠，所以就沒有中間段的部分。

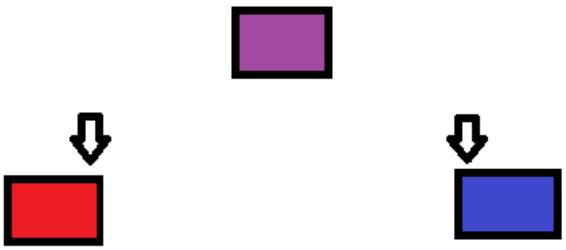


圖 3-6 短賽道區段分解圖

長度短的賽道區段(圖 3-6)，前端和後端都取樣到整段賽道區段長度為止，得到的長度會比完整取樣資料短，中間區段省略。

3.6 軌跡資料群統計模型

在以賽道區段為單位的玩家行為比較之中，因為玩家的賽車軌跡資料被切成許多較小的片段，軌跡資料長度變短但是資料組數量大增，所以不再採用每一條軌跡互相比較的方法，改用較為單純的統計式模型代替軌跡資料群，用來代表一個玩家的特性。

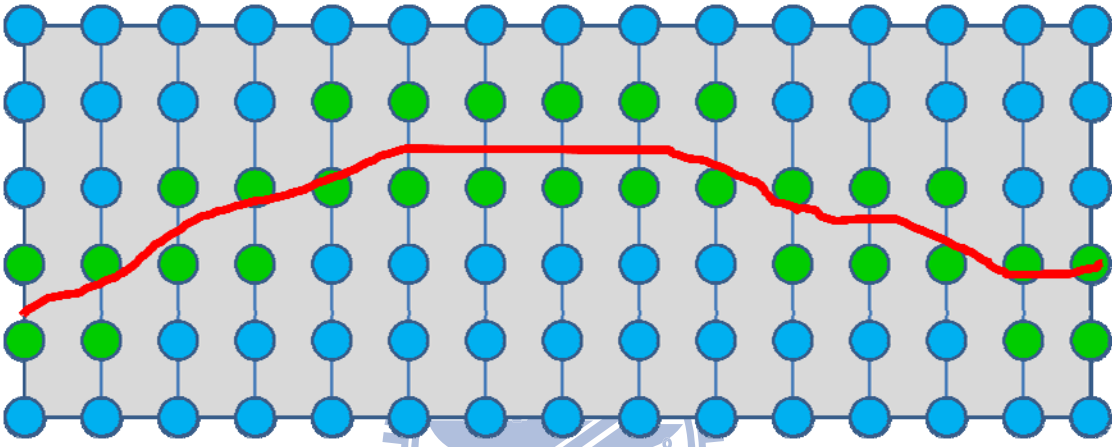


圖 3-7 checkpoint 上的統計點陣列

我們在每個 checkpoint 的相同位置上設定若干個統計點，形成一個棋盤狀的統計點矩陣(圖 3-7)，使用一組軌跡資料群對這個資料點矩陣作 training，最後每個統計點都會得到各自的統計值，這一整組統計值矩陣就是一個統計模型，可以代表整組軌跡資料群的某些統計特性。這個模型統計資料分成兩種，一種是針對賽車軌跡通過 checkpoint 位置的機率分布模型，另外一種是統計其他變數用的統計平均值模型，這裡的變數可以代入速度、車子方向或加速度等資料。

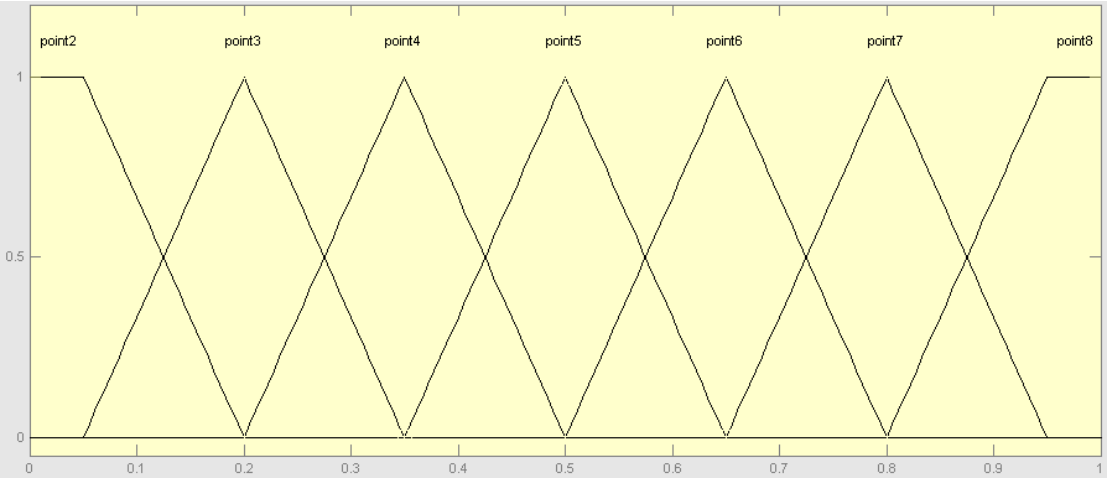


圖 3-8 統計點的偵測函數

偵測函數(圖 3-8)代表通過賽車軌跡通過 checkpoint 位置和統計點位置的對應關係，上圖為每個統計點的偵測函數，橫軸為賽道寬度方向的座標，0 代表右邊邊界，1 代表左邊邊界，縱軸代表偵測值，我們在每個 checkpoint 上設置了 9 個統計點，第 1 個統計點代表位置值小於 0，也就是超出右邊邊界，第 2~8 個統計點涵括了整個賽道(位置值 0~1)的部分，第 9 個統計點代表位置值大於 1，也就是超出左邊邊界的部分。偵測函數的部分是使用線性內插的方式計算，不論賽車從哪個位置通過 checkpoint，這個 checkpoint 上全部統計點的偵測值總和都保持為 1。

設統計點的偵測函數為 detect(x)，軌跡資料通過 checkpoint 位置為 pos，軌跡資料數量為 N

在製作位置統計模型時，每個統計點的值為

$$\sum_{i=1}^{N} \text{detect}(\text{pos}_i)$$

在製作其它變數統計模型時，設 var 為變數值，每個統計點的值為

$$\frac{\sum_{i=1}^{N} \text{detect}(\text{pos}_i) \times \text{var}}{\sum_{i=1}^{N} \text{detect}(\text{pos}_i)}$$

圖 3-8 舉一個用軌跡資料 training 統計模型的範例，training 分三個回合，每

次加入一條軌跡資料做統計，圖 3-8 中每一張小圖的縱軸表示 checkpoint 的位置，橫軸代表統計點的位置，顏色代表統計點的值。這裡是用位置統計模型，統計值會不斷累加變大，在軌跡交叉點的地方統計值會特別高，代表玩家常用的通過路徑。

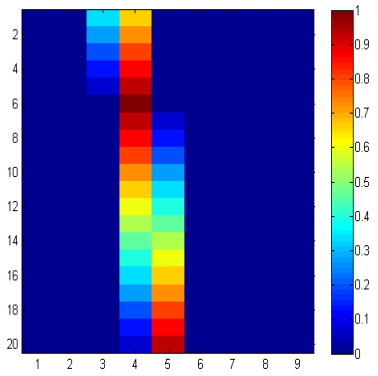
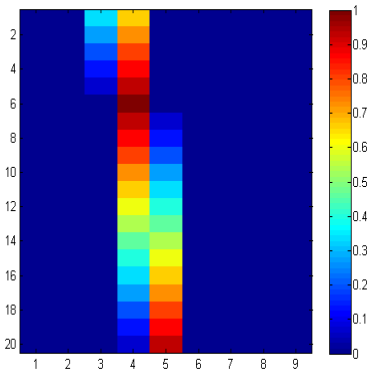
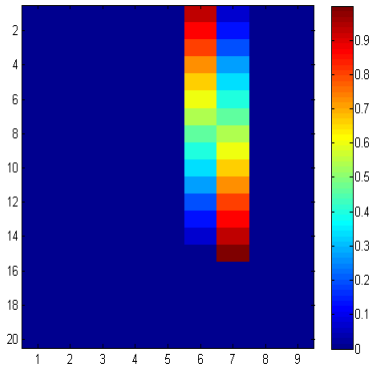
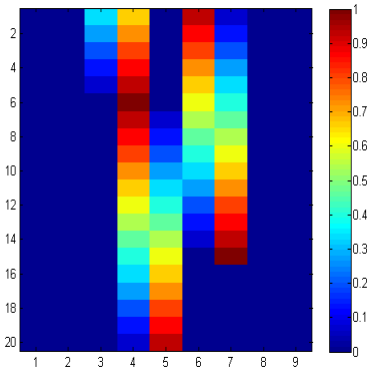
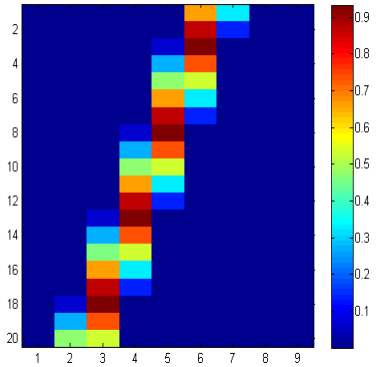
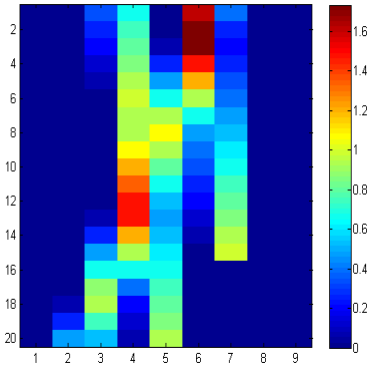
	加入的軌跡資料	位置統計點矩陣值
第一回合		
第二回合		
第三回合		

圖 3-9 三條軌跡資料合成的位置統計模型

在位置統計矩陣中，因為 training 用軌跡長短不同，會造成後面的 checkpoint 統計點沒被經過，取不到值，因此需要做 normalization，把每個 checkpoint 總和調整到 1，這樣位置統計矩陣就可以表是每個點的通過機率，圖 3-10 為每一個橫排(checkpoint)normalize 的效果。

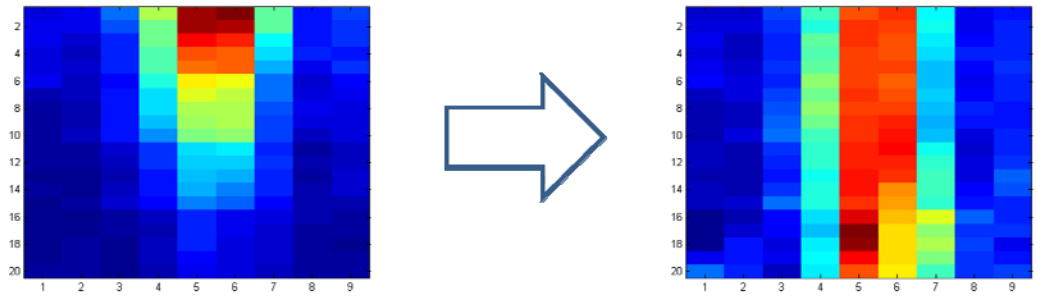


圖 3-10 位置統計點 normalization

圖 3-11 是和圖 3-9 同樣的一條賽車軌跡，但是使用速度的統計模型，和位置的統計公式不同，在做速度統計的時候軌跡交叉點會做平均的動作。

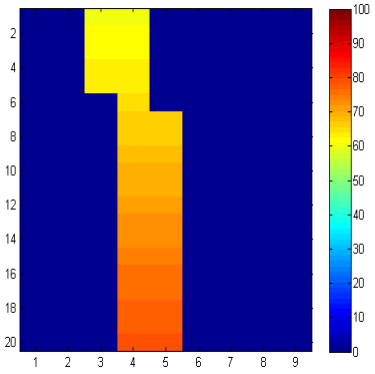
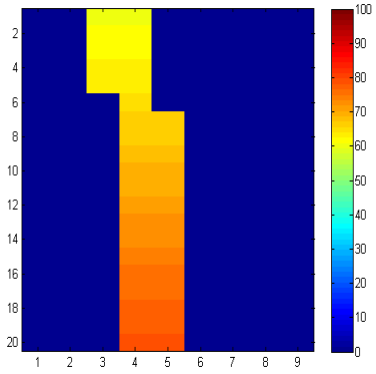
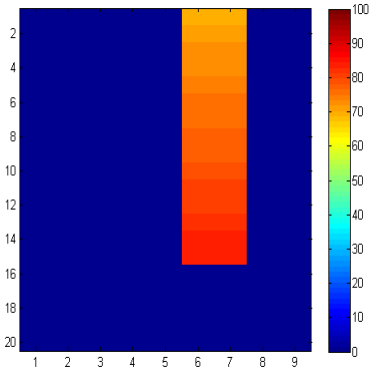
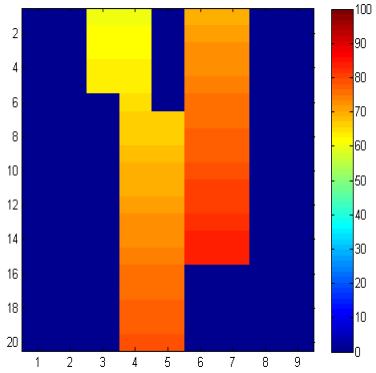
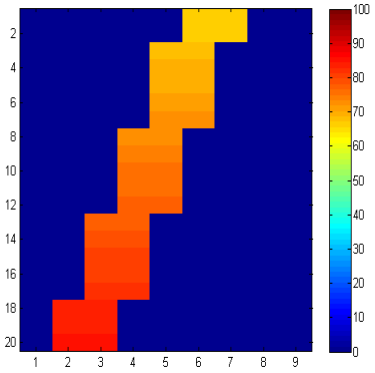
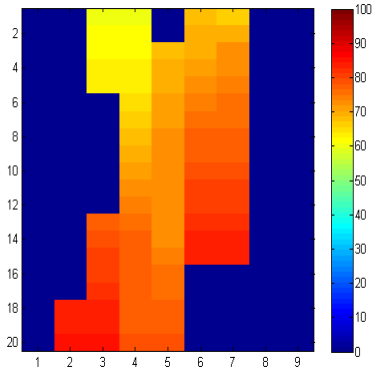
	加入的軌跡速度資料	速度統計點矩陣值
第一回合		
第二回合		
第三回合		

圖 3-11 三條軌跡資料合成的速度統計模型

把屬於同一個人的所有賽道區段軌跡資料的前段和後段分別用來製作統計模型，最後就可以得到一組代表這個玩家行為模式的統計模型，因為把曲率變數分成 9 個區段，所以統計出來的矩陣有 9 個，分別由 9 張圖表示(圖 3-12 及圖 3-13)。

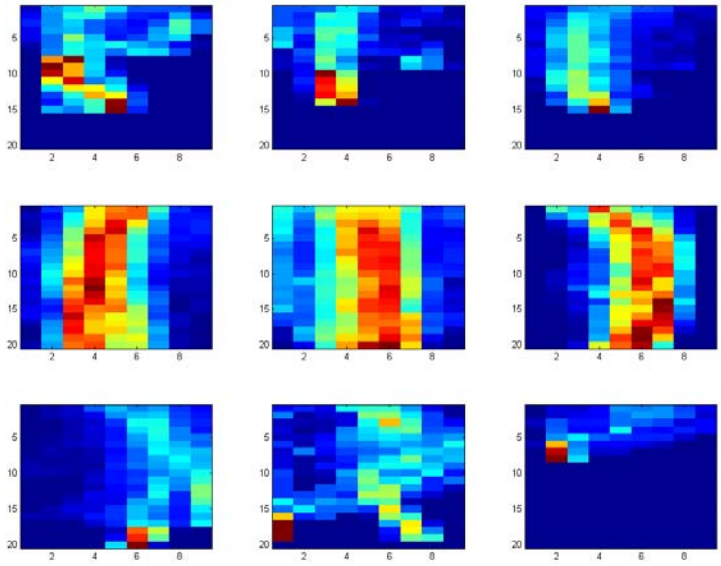


圖 3-12 代表一個玩家的 position 變數統計圖

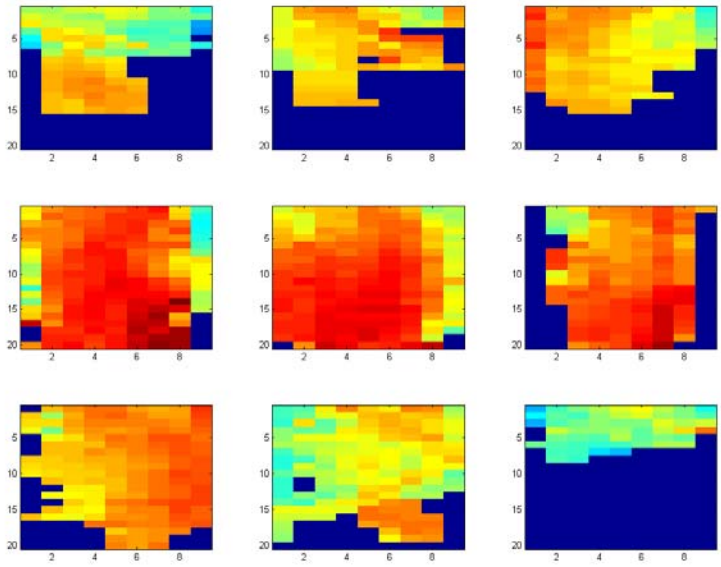


圖 3-13 代表一個玩家的 speed 變數統計圖

3.7 使用玩家統計模型做駕駛員推測

我們希望能夠做出一種分辨玩家特色的方法，能夠從一組遊戲資料之中，依據它的駕駛特色找出駕駛人是誰。為了達成這個目標，我們把收集到的駕駛資料根據來源玩家分群，用每個玩家所屬的遊戲資料，製造出個人的統計模型，然後使用這些統計模型推測出軌跡資料的駕駛員。

要預測一個軌跡資料是由誰駕駛，只要將軌跡和每個人的統計圖上相同的位置做比對，結果最接近的就判定為駕駛員，因為統計圖有兩種，所以比對方法也分成兩種。第一種是位置的統計圖，因為統計的結果 normalize 後代表通過的位置機率分布，所以計算相似度的方法是將軌跡通過每個 checkpoint 統計點的機率值相乘，判定相似度最高的統計圖為駕駛員。第二種是其它變數的統計圖，以速度為例，根據軌跡位置在每個玩家統計圖上的相對位置算出每個玩家對應的速度，把每個 checkpoint 上的誤差值加總起來代表軌跡資料和這個玩家的相異度，相異度最小的玩家判定為駕駛員。

第四章　實驗結果

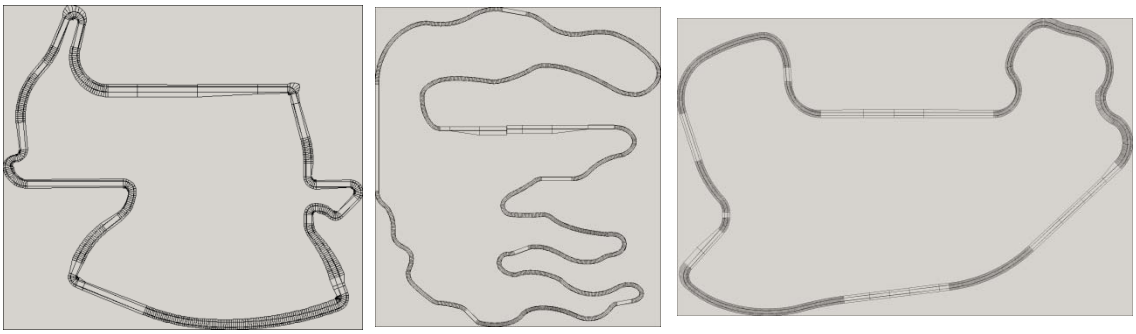
4.1　實驗資料

我們收集玩家在 TORCS 上駕駛的資料作為研究材料，總共有 6 名玩家參加這個實驗，每個人都用一套相同的流程做資料收集，得到相同數量的玩家軌跡資料，這樣會讓不同玩家間容易做比較。

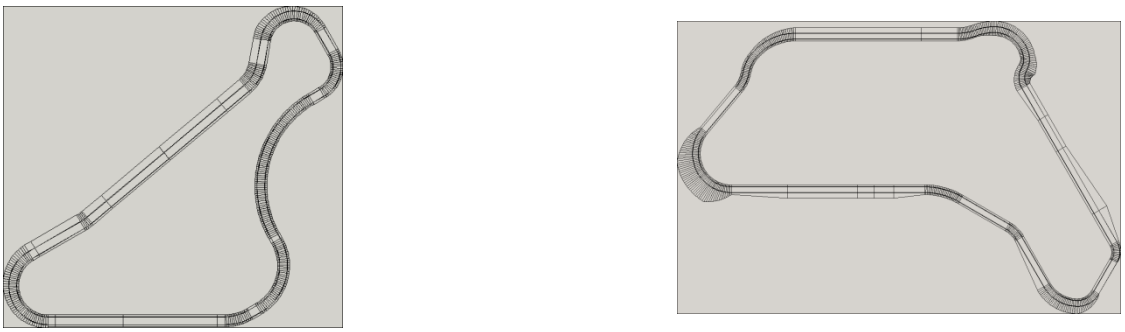
在採取資料的方法上，一個場次設定為賽車沿著賽道跑完 3 圈，玩家收集資料時，在一個固定的賽道跑完 3 個場次，之後在換到下一個賽道，總共 5 個賽道都個跑 3 場，實驗用 lap data 當單位時，每個人都可以收集到 5 個賽道x3 場x3 圈=45 個 lap data。

玩家在遊戲中的設定，玩家使用車輛為 Formula one，不使用其他的電腦 AI 車輛或是其他玩家對手，以免影響到玩家的正常駕駛行為，賽道選用 Aalborg、Alpine-1、E-track 4、CG speedway number 1、CG track2 這五個場地(圖 4-1)。

遊戲難度設置為 rookie 新手級,玩家的賽車不會因為撞牆太多次而損壞。



(a) (b) (c)



(d) (e)

圖 4-1 實驗用跑道(a)Aalborg (b)Alpine-1 (c)E-track 4(d) CG speedway number 1 (e) CG track2

4.2 整圈賽道 checkpoint 差異度測試結果

我們把 6 名玩家的資料，使用 3.2 節的整圈賽道 checkpoint 差異度測試方法，判斷每個玩家和其他玩家之間以及玩家和自己之間的相異度比值，使用 4 種變數當作測試項目：position 為通過 checkpoint 位置，speed 為賽車速度，accerate 為賽車加速度，angle 為賽車方向和跑道中心軸夾角。計算結果列在表 4-1 ~ 表 4-4。

表 4-1 Position 變數下的整圈賽道 checkpoint 差異度測試結果

	track1	track2	track3	track4	track5
user1	0.8614	0.8300	0.8119	0.6569	0.7440
user2	0.7665	0.7609	0.7103	0.6400	0.8355
user3	0.9610	0.9312	0.9911	0.8515	0.9886
user4	0.8280	0.9801	1.0294	0.8259	0.9316
user5	0.8910	0.9351	0.8561	0.7461	0.7978
user6	0.9057	0.9928	0.7934	0.7590	0.9720
平均值	0.8689	0.9050	0.8654	0.7465	0.8783

表 4-2 speed 變數下的整圈賽道 checkpoint 差異度測試結果					
	track1	track2	track3	track4	track5
user1	0.7907	0.6346	0.5776	0.8316	0.7929
user2	0.6663	0.5791	0.5544	0.6191	0.6608
user3	0.9580	0.8746	1.0103	0.9282	0.9789
user4	0.6302	0.9895	0.9021	0.7262	0.8160
user5	0.8533	0.7605	1.0113	0.9142	0.8434
user6	0.6162	0.5355	0.7057	0.7151	0.9843
平均值	0.7525	0.7290	0.7936	0.7891	0.8461

表 4-3 accelerate 變數下的整圈賽道 checkpoint 差異度測試結果					
	track1	track2	track3	track4	track5
user1	0.9327	0.9214	0.9035	0.8849	0.8320
user2	0.9185	0.9023	0.9542	0.8932	0.9708
user3	0.9720	0.9124	0.9803	0.9988	0.9710
user4	0.8349	0.9765	0.9955	0.9826	0.9979
user5	0.9587	0.9032	0.9429	0.9476	0.9038
user6	0.8311	0.9705	0.9587	0.9851	0.9583
平均值	0.9080	0.9310	0.9559	0.9487	0.9390

表 4-4 angle 變數下的整圈賽道 checkpoint 差異度測試結果					
	track1	track2	track3	track4	track5
user1	0.8902	0.8609	0.8000	0.7418	0.8696
user2	0.8241	0.7996	0.7437	0.6648	0.8712
user3	0.9507	0.9021	0.9914	0.9034	0.9238
user4	0.8661	1.0720	1.1266	0.9128	0.9333
user5	1.0428	1.0358	1.0847	1.0562	0.9467
user6	0.8576	0.8640	0.8281	0.8438	1.1372
平均值	0.9052	0.9224	0.9291	0.8538	0.9470

從表 4-1 到表 4-4 的結果來看，大部分值都小於 1，代表每個玩家和自己的差距值比和別人的差距值低，每個玩家確實具有自己的駕駛風格存在，而不是完全用固定的物理最佳演算法開車，並且每個人的風格都有些差異。

4.3 駕駛員預測實驗結果

我們把每個玩家的所有賽車軌跡資料，依據賽道區段切成片段，並且用 3。7 節的方法分別預測這些片段的駕駛員，我們分別使用了三種不同變數的統計模型做預測，position 為賽車通過 checkpoint 位置，speed 為賽車速度，angle 為和賽道中心線夾角，表 4-5 ~ 4-7 為統計 6 個玩家資料來源分配到 6 個駕駛預測的結果，左上-右下對角線的值代表預測正確。

表 4-5 Position 變數的駕駛員預測結果

	預測 1	預測 2	預測 3	預測 4	預測 5	預測 6
來源 1	1133	42	405	75	324	181
來源 2	980	52	501	150	288	189
來源 3	932	46	533	159	315	175
來源 4	904	36	466	216	317	221
來源 5	983	27	401	151	388	210
來源 6	900	45	573	159	271	212

表 4-6 speed 變數的駕駛員預測結果						
	預測 1	預測 2	預測 3	預測 4	預測 5	預測 6
來源 1	476	238	106	586	419	335
來源 2	436	268	124	593	387	352
來源 3	385	189	194	714	386	292
來源 4	420	187	138	728	384	303
來源 5	349	143	76	826	512	254
來源 6	489	198	178	428	393	474

表 4-7 angle 變數的駕駛員預測結果						
	預測 1	預測 2	預測 3	預測 4	預測 5	預測 6
來源 1	596	188	424	321	304	327
來源 2	588	215	404	360	258	335
來源 3	504	196	527	355	268	310
來源 4	482	183	408	447	301	339
來源 5	553	165	416	359	346	321
來源 6	513	182	471	371	269	354

在結果表格中，每一橫排代表一個玩家整組資料的預測結果，可以看出來對每個玩家的偵測成功率並不高。在這些結果表格中，左上到右下對角線上的值代表資料來源和預測結果是同一人，表示預測正確的狀況。把對角線上的值相加，再除以總預測數，就可以得到預測正確率。實驗中三種變數的預測成功率分別是 Position 19.55%、speed 20.46%、angle 19.17%，結果都比隨機猜測的值 1/6 = 16.67%好。這個結果表示這個偵測方法還是帶有某些效果，並不是完全無效或隨機的。

表 4-8 混和三種變數的駕駛員預測結果

	預測 1	預測 2	預測 3	預測 4	預測 5	預測 6
來源 1	697	144	315	284	437	283
來源 2	572	210	375	386	360	257
來源 3	466	170	498	410	418	198
來源 4	451	134	364	489	477	245
來源 5	439	127	313	447	606	228
來源 6	578	145	428	304	337	368

我們用一種簡單的方法把 position、speed、angle 三種變數合併起來做駕駛員的預測，先分別使用三種變數做預測，在每個變數項目下排出和駕駛員相似度的排名，最後再把三個項目的排名相加，得到的排名總合最低者就判定為駕駛員，表 4-8 為混合變數的預測結果。混和三種變數後的預測成功率為 22.13%，效果比三種變數個別使用的駕駛員預測更加準確。

第五章 結論與未來展望

在這篇論文中，我們研究在 TORCS 賽車遊戲平台上，每個不同玩家之間行為的差異，並且用 checkpoint 取樣的差異度計算每個玩家之間的誤差，證明每個玩家確實有自己特殊的玩家個性。之後我們使用一種三階段模型對賽道做分解，讓不同的賽道區段可以互相做比較，最後我們製作出一種代表玩家個性的統計模型，並且用這個模型預測駕駛員的方式，結果說明這個模型記錄了部分的玩家特色，但是仍然不足以明確的辨別出各個玩家。因為代表玩家特色的統計模型裡，可以代入不同的變數，但是這篇論文只簡單的實驗了數個代表性的變數，如位置和速度等，未來我們可以利用這個模型，用其他的變數比較玩家差異度，並且研究各種參數組合的方法，達到更精確的玩家分辨效果。

參考文獻

[1] Daniele Loiacono, Luigi Cardamone, Pier Luca Lanzi, “Simulated Car Racing Championship 2010: Competition Software Manual", Technical Report 2010.8,Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 2010.

[2] Enrique Onieva, David A. Pelta, Vicente Milanés, Joshue Pérez, "A fuzzy-rule-based driving architecture for non-player characters in a car racing game" *Soft Computing - A Fusion of Foundations, Methodologies and Applications* Volume 15, Number 8, pp. 1617-1629, 2011.

[3] Diego Perez, Gustavo, Yago Saez,"Evolving a Fuzzy Controller for a Car Racing Competition" *Computational Intelligence and Games*, pp. 263 – 270, 2009.

[4] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, "Evolving Competitive Car Controllers for Racing Games with Neuroevolution" *GECCO '09 Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1179-1186, 2009.

[5] Luigi Cardamone, Daniele Loiacono and Pier Luca Lanzi, "Applying Cooperative Coevolution to Compete in the 2009 TORCS Endurance World Championship" *Evolutionary Computation (CEC)*, pp.1 – 8, 2010.

[6] Marc Ebner and Thorsten Tiede,"Evolving Driving Controllers using Genetic Programming" *Computational Intelligence and Games*, pp. 279 – 286, 2009.

[7] Daniele Loiacono, Alessandro Prete, Pier Luca Lanzi, Luigi Cardamone,"Learning to Overtake in TORCS Using Simple Reinforcement Learning" *Evolutionary Computation (CEC)*, pp. 1-8, 2010.

[8] Jan Quadflieg, Mike Preuss, Oliver Kramer, Gunter Rudolph,"Learning the Track and Planning Ahead in a Car Racing Controller" *Computational Intelligence and Games (CIG)*, pp. 395 – 402, 2010.

[9] Jorge Munoz, German Gutierrez, Araceli Sanchis, “Controller for TORCS created by imitation” *Computational Intelligence and Games*, pp. 271 - 278 2009

[10] Jorge Munoz, German Gutierrez, Araceli Sanchis, “A human-like TORCS controller for the simulated car racing championship” *Computational Intelligence and Games (CIG)*, 473 – 480, 2010.

[11] Luigi Cardamone, Daniele Loiacono and Pier Luca Lanzi, “Learning drivers for TORCS through imitation using supervised methods” *Computational Intelligence and Games*, pp. 148-155, 2009.