行政院國家科學委員會專題研究計畫　成果報告

子計畫二：動態 VR 運動復健輔助系統之智慧型感測與控制

計畫類別： 整合型計畫
計畫編號： NSC92-2213-E-009-017-
執行期間： 92 年 08 月 01 日至 93 年 07 月 31 日
執行單位： 國立交通大學電機與控制工程學系

計畫主持人： 林進燈

報告類型： 完整報告

處理方式： 本計畫可公開查詢

中　華　民　國 93 年 11 月 1 日

行政院國家科學委員會補助專題研究計畫 **■ 成 果 報 告**
□期中進度報告

泛用型動態虛擬實境操控與運動復健輔助系統研發

子計畫二: 動態 VR 運動復健輔助系統之智慧型感測與控制

(3/3)

計畫類別：□ 個別型計畫　　■ 整合型計畫
計畫編號：NSC 92 - 2213 - E - 009 - 015 -
執行期間：90 年 08 月 01 日至 93 年 07 月 31 日

計畫主持人：林 進 燈 教授
共同主持人：
計畫參與人員：

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學電機與控制工程學系

中　華　民　國　93　年　7　月　　日

泛用型動態虛擬實境操控與運動復健輔助系統研發
子計畫二: 動態 VR 運動復健輔助系統之智慧型感測與控制(3/3)

計畫編號：NSC92-2213-E-009-015
執行期限：90.8.1-93.7.31
主持人：林進燈　國立交通大學　教授
執行機構：國立交通大學電機與控制工程研究所

一、摘要

本計畫為「泛用型動態虛擬實境操控與運動復健輔助系統研發」整合計畫之子計畫二，研發重點在於電動動作平台之設計與分析、運動訓練輔助機制之感測與控制及操作者運動狀況之偵測與分析。在前二年本子計畫的重點之一擬開發一智慧型控制系統於電動動作平台的姿態控制設計，以為整個動態運動訓練輔助系統的根基平台。第二個重點在為了提高系統的穩定度與安全性，系統對外界訊息的反應，必須要更為迅速，以確保在系統出現問題時能立及做出適當的處理，而構成即時動態運動訓練系統的整合機構。第三個研究主題是以 LART 實驗單板的 Intel StrongARM SA-1100 處理器之 Linux 嵌入式系統來取代舊有的 IPC 控制方式。本子計畫也將發展即時計算環境的軟、硬體，以達到整體系統的即時控制效果。第三年本子計畫的第一個重點擬開發一動態模擬駕駛系統以結合動作平台的姿態控制，以為整個動態運動訓練輔助系統的虛擬互動場景。本子計畫的第二個重點在為了提高系統的穩定度與安全性，系統對外界訊息的反應，必須要更為迅速，以確保在系統出現問題時能立及做出適當的處理，而以同時具有 ARM 和 DSP 微處理器之 OMAP 之嵌入式系統，補足以往嵌入式系統運算能力之不足，以達到整體系統的即時控制效果。

關鍵字
電動運動平台、適應性小腦模型控制器、即時作業系統、嵌入式即時硬體單板、動態模擬駕駛系統、互動場景、OMAP、嵌入式系統

二、前二年研究：

針對本子計畫所將完成的三大研究主題，在前二年本子計畫目前朝向電動動作平台的控制系統、平台即時作業系統開發、嵌入式即時硬體單板之設計與發展等部分。以下就分別針對此幾個方向來加以說明。

A. 電動動作平台的控制

近二十年來電氣伺服逐步取代傳統油壓伺服在工業界的應用，因為電氣伺服可靠且容易維護，再加上精度高、成本低等優勢，表一就油壓與電氣作動之優缺點作一簡單的比較。

基於本實驗室過去開發油壓運動平台的經驗，本研究已成功完成電動運動平台的機構設計、加工及驅動系統研究，由於電動運動平台之動態模型十分複雜度，所以不易使用傳統的控制理論來達成控制的目的，為解決此問題，本研究提出一架構簡單、具快速學習且不需要受控系統動態模型的適應性小腦模型控制器來解決此控制問題，經由實驗結果發現本研究所提出之適應性小腦模型控制器可以有效地準確控制電動運動平台之腳長長度，其方塊圖如圖一所示與設計步驟與想法如下簡單描述：

步驟一：

定義追蹤誤差 $e=q_d-q$ ，其中 q 代表伺服馬達實際的轉子位置，q_d 代表參考模式命令訊號，並定義一滑動表面 $s=\dot{e}+k_1e+k_2\int e$。

步驟二：

利用一小腦模型控制類神經網路線上學習近似一理想控制器。

步驟三：

依據最佳近似定理我們可得知存在一近似誤差，為了克服此誤差往往使用一切換控制器補償之，但卻因而造成控制力有嚴重的顫抖現象。

步驟四：

使用一個極限值估測器來監測不確定量邊界值，在此定義不確定量邊界值估測誤差為 $\tilde{E}(t) = E - \hat{E}(t)$。

步驟五：

適應性小腦模型控制系統設計成

$$u(t) = \hat{u}_{CMAC} + u_{cp}$$

其中 $\hat{u}_{CMAC}$ 為主要追蹤控制器用來近似理想控制器;而補償控制器 u_{cp} 則被設計來消除理想控制器與小腦模型控制器之間的誤差。

步驟六：

依據李雅普諾夫穩定法則推論而得之線上學習法則

$$\dot{\hat{\mathbf{a}}} = h_2 s(t)\hat{\mathbf{F}}$$

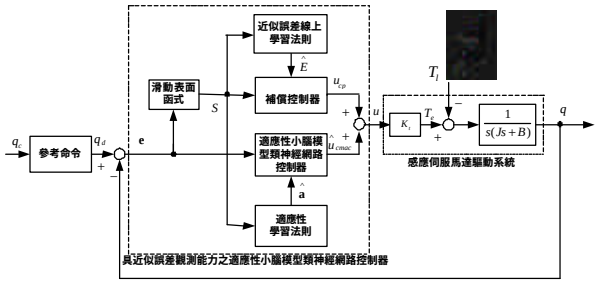
$$u_{cp} = \hat{E}(t)\,\mathrm{sgn}(s(t))$$

$$\dot{\hat{E}} = h_1 |s(t)|$$

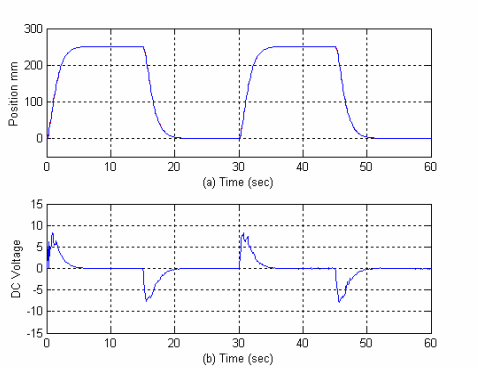
其中 h_1 與 h_2 是學習速率。

步驟七：

為了驗證所設計之控制器對於機械參數變化與外來的負載干擾的性能，我們利用方波位置命令來驗證控制器追隨控制角度變化的軌跡來觀測平台的動作是否正確，以驗證我們所發展式的小腦模型控制器設計是否正確，其實驗結果如圖二所示。



圖一　適應性小腦模型類神經網路控制系統



圖二　適應性小腦模型類神經網路控制系統實驗響應圖

表一 油壓式/電動式平台控制系統之比較

油壓式平台		電動式平台	
優點:	缺點:	優點:	缺點:
•可產生較高之加速度 •作動元件尺寸較小 •耐用的作動元件 •非常高的承載能力	•有漏油顧慮 •效率低造成高性能之作動閥對環境要求較高 •較多的突發狀況-如閥軸卡死 •安裝較麻煩需另加油壓單元	•乾淨 •不需另加其它附屬裝備 •效率高 •維護容易 •安裝容易 •不用複雜的作動閥	•加速性較差 •作動缸結構複雜 •安全裝置十分複雜 •複雜的電子設計 •系統運作動力需求變化大

B. 平台即時作業系統

虛擬實境的應用非常的廣泛，例如進行飛行器的模擬，汽車的駕駛訓練，或是一些精密控制的模擬。在一個複雜的虛擬實境模擬系統中，時常需要處理大量的外界訊息，當系統呈現負載的情況時，其中有些重要的訊息必須是不能忽略或是要優先處理的，以保持模擬的精確性，其次，和實際的器具一樣，系統的穩定性、可靠性及安全性都是我們所考量的，為了提高系統的穩定度與安全性，系統對外界訊息的反應，必須要更為迅速，以確保在系統出現問題時，能立及做出適當的處理。故我們希望能提高系統中的即時性作業能力，因此，也確立了即時系統的必要性。以下是我們今年針對虛擬實境控制平台，進行即時性分析與設計的成果。

我們所使用的即時作業系統是建構在 Linux 作業系統之上的微核心模組 Real-Time Application Interface(RTAI)，這個微核心負責處理所有與硬體間的動作，針對需滿足即時性的工作，會在這個即時核心的工作空間中處理。其特性包括：

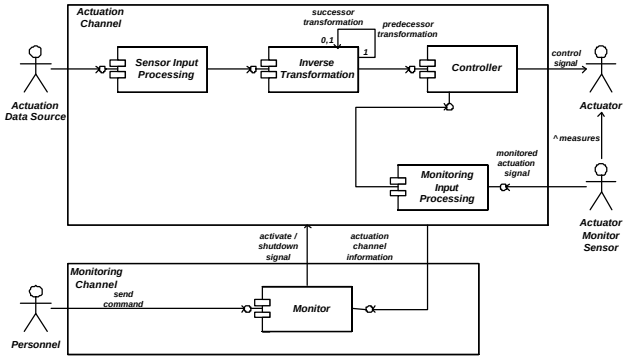
- 即時工作的排程器，可進行不同的即時性排程策略。
- 解析度更高的計時器，可供精密度更高的控制。
- 完全強取式(Preemptive)模式，使得優先權高的工作先處理。
- 豐富的程序間通訊(IPC)機制，如：FIFOs, shared memory, mailboxes。
- 維持 Linux 原有豐富資源，不需重新開發應用程式與裝置驅動程式。

首先，是有關即時控制六軸運動平台的分析，我們進行了即時 (Actuation channel 部份)與非即時 (Monitor channel 部份) 工作的分工，細部分工如圖三 UML 使用者關係圖所示。根據我們模擬系統的流程，大至上可分成五個狀態，分別是 initial, ready, running, shutdown, emergency。以下是五種狀態的在即時控制系統裡狀態轉換關係圖，如圖四所示。

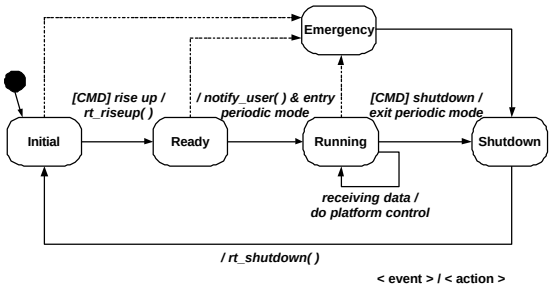
為了達到即時運算、處理的能力，我們將平台的逆向運動學及位置控制部份，放入即時系統核心中工作，另外，分別以週期性與偶發性工作來分派低階 D/A 與 A/D 硬體工作，除此之外，透過即時系統提供的程序間通訊機制，讓使用者空間的監測程式與在即時核心中運作的工作，進行資料通訊。圖五是虛擬平台即時控制系統的實現架構圖。

為了確保有些關鍵性工作能夠被優先執行，如 Shutdown 的工作，因此，我們進行了工作優先順序的指派動作，表二是有關工作優先權的分配表，其中 `1`代表最高，`10`代表最低。

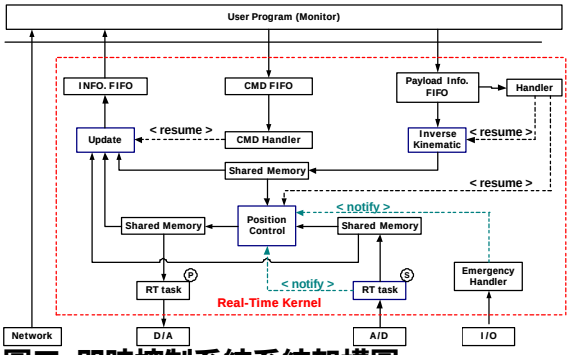
圖六為透過網路與 FlightGear 虛擬場景結合，實際截取即時控制系統之活動資料的展示，其中的 $Tick\_Time = 100ms$，代表的是每隔 $100ms$ 即會檢驗是否有其它外部中斷發生，並迅速地做出反應，或著是否有其它週期性工作，即將要執行，則會迅速進行排程動作，以喚醒工作執行，因此，不僅提高了控制系統的反應能力，也增加了週期性工作排程的精度。而圖中 $Period_{D/A} = 100ms$，也就是以頻率為 $10kHz$ 的速度，進行精確的平台控制。相較於一般的作業系統，就 *Linux* 而言，其 *Tick* 的時間，標準為 $1ms$，此段時間則視系統受負載的程度而有所變動 ($1ms \sim 100ms$)，故其延遲(*Latency*)時間，為 $1ms \sim 100ms$，因此它所能提供的控制環境，是屬於反應性較差，精確度較低的環境，不適合用來進行精密的虛擬實境模擬。而在本年度成果中，我們為了將來進行一些高精度的控制，如：電動平台的控制，而成功地增加了即時控制系統，且大大提高模擬的精準能力。



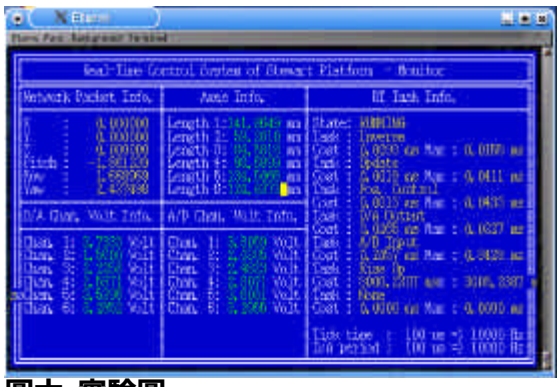
圖三 UML 使用者關係圖



圖四 即時控制系統狀態圖



圖五 即時控制系統系統架構圖



圖六 實驗圖

表二 Task 之優先順序表

Task	*Priority*	*Comment*
Shutdown platform	1	Command-driven
Inverse Kinematics	2	FIFO-driven
Rise up platform	3	Command-driven
Analog Output(D/A)	4	Periodic
Position Control	5	FIFO-driven
Analog Input(A/D)	9	Command-driven
Update	10	Command-driven

C. 嵌入式即時硬體單板之設計

在本計劃中由於在資料傳輸上受限於 RS-232 串列傳輸， 同時以 LART 實驗單板的 Intel StrongARM SA-1100 處理器之 Linux 嵌入式系統來取代舊有的 IPC 控制方式，這不僅使控制系統體積大幅減少, 也使我們能在 Linux 作業系統上發展新的控制程式，並且可以達到多平台及同步接收資料。而嵌入式系統週邊亦支援 TCP/IP 網路。然而在 LART 實驗單板上並沒有 CAN bus 的控制裝置與 driver 可提供虛擬實境場景的六軸腳長 data 與 LART 實驗單板作為控制資料的連結與傳輸，所以必須另外增加 LART 實驗單的 interface 與 CAN bus 裝置.並且須要建立在 LART 實驗單板上的 CAN bus driver。 然而在 LART 實驗單板上亦是沒有 A/D 及 D/A 控制器，所以也必須增加 LART 實驗單的 interface 與 A/D 及 D/A 的控制裝置，並且建立 LART 實驗單板上的 A/D 及 D/A 驅動程式。

在今年進度報告中，我們完成以下的工作:

- A/D，D/A 裝置的製作
- LART 實驗單與 A/D 及 D/A 間的 interface 電路設計
- CAN bus 裝置的製作
- LART 實驗單與 CAN bus 間的 interface 電路設計

詳細細節請見後面之敘述

在嵌入式硬體部分，我們所採用的是 LART 實驗單板，選擇這塊板子的原因是因為它選擇這塊板子的原因是因為它有豐富的序列傳輸介面，包括 IrDA、RS-232，也有內建 10Base-T 網路，並且支援包括 Linux 嵌入式作業系統, 可以發展的嵌入式種類及相關應用程式可謂相當豐富。此塊單板實體圖片如圖七所示。

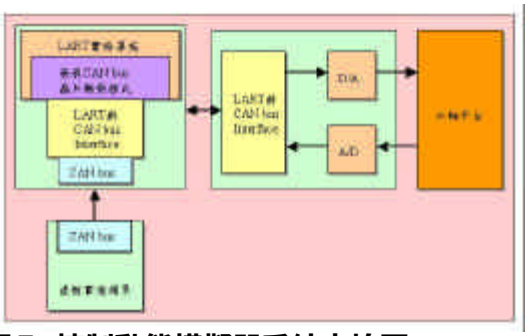
除此之外，所使用的微處理器為 SA-1100，而在 LART 的計畫中亦是使用 Strong ARM 微處理器系列。所謂 LART 計畫是由國外的 Delft University of Technology 所主持的研究計畫，主要研究在消耗不到一瓦特功率而可以達到 250MIPS 指令的 Linux 嵌入式系統。它有一套較為完整的 Linux 嵌入式系統文件及 mailing list, 更難得的是它亦將所有的軟硬體公開。因此在未來發展 StrongARM SA-1100 嵌入式系統時可以有較為完整的相關文件可以參考。

在 LART 實驗單板上發展嵌入式系統來控制虛擬實境動態模擬器，並沒有 CAN-Bus 的裝置與驅動程式，因此我們必需要發展 CAN-Bus 的裝置與驅動程式，來作為與 LART 實驗單板的傳輸介面。如此才可以與 LART 實驗單板溝通，接收由虛擬實境場景的六軸腳長 data，然後將腳長轉電壓，控制動態模擬器。其系統方塊圖如圖八所示。

在動態模擬器六軸姿態的傳遞方式上，我們將發展 CAN-Bus 通訊協定，因此必須在嵌入式系統上發展 CAN-Bus 驅動程式，A/D 及 D/A 驅動程式；除此之外，還需發展 LART 實驗單板之 CAN-Bus 程式，如此才能以 CAN-Bus 傳遞六軸控制姿態至動態模擬器驅動控制盒上。圖九為 CAN bus 實際之控制電路板。

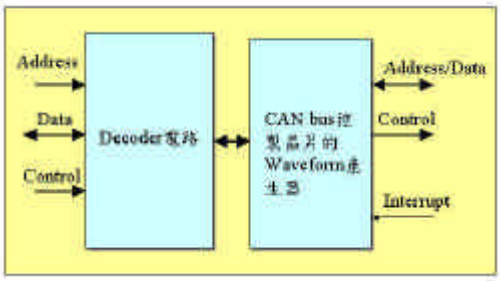
LART 實驗單板提供了 GPIO， data bus 與 address bus pin 可作為其他額外的控制裝置使用，同時我們利用 address，data bus 與 read/write 的控制信號 pin，與 ALTERA 公司所提出的 FPGA 晶片，作為 LART 實驗單板與 CAN bus 裝置間的控制。圖十為 LART 實驗單板與 CAN bus 間的 interface 電路方塊圖。

圖七 單板實體圖片

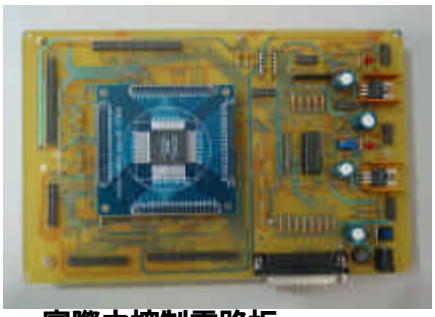


圖八 控制動態模擬器系統方塊圖

圖九 CAN bus 實際之控制電路板



圖十 LART 實驗單板與 CAN bus 之間的電路方塊圖



圖十一 實際之控制電路板

三、第三年研究：

針對本年度所將完成的二大研究主題，在今年度本子計畫目前朝向動態模擬駕駛平台系統、OMAP 嵌入式即時系統之發展等部分。以下就分別針對此幾個方向，來加以說明目前進度狀況。

A. 動態模擬駕駛平台

在動態模擬駕駛平台系統的發展上主要探討兩部分，為車輛動態的研究以及虛擬實境場景的建立，以達到更擬真的感覺。

在汽車的動態模型之中，探討了一些有關於汽車轉向、引擎加速及懸吊系統的問題。

在汽車的轉向中，是由前輪轉一角度，後輪跟著一起前進的，而場景程式裡，雖可以得到前輪轉的角度，但是還需要車體真正轉的角度，以及前進的距離。要求得這兩項重要的參數，以車子轉彎的瞬間，移動的軌跡中，分析其幾何的變化來求得。假設汽車轉彎時，是以內側後輪為支點而旋轉，而在前輪順著它自己轉的角度前進的時候，後輪包含支點的輪子也會跟著被拖行一段距

離，而後輪所前進的距離及方向，採用近似的方法，假設支點在移動的瞬間，是以筆直朝原來的方向移動一段距離。

用這樣的關係和已知的條件：以瞬時速度外側車輪移動距離為 v，車輪轉 α 度，加上前後輪距 l，輪軸長 w。求出車體真正轉的角度 β，和支點輪移動的距離 z。

圖十二為車子右轉的假設圖，以及經過簡化後我們需要的資訊如，接著做輔助線以求得 z、β，如圖十三所示。

以四邊形上下兩邊相等，可求得 β。

$$l\sin b + w = w\cos b + v\sin a$$

$$\sqrt{l^2 + w^2}\left(\sin g \sin b - \cos g \cos b\right) = v\sin a - w$$

$$\cos\left(b + g\right) = \frac{w - v\sin a}{\sqrt{l^2 + w^2}}$$

$$b = \cos^{-1}\left(\frac{w - v\sin a}{\sqrt{l^2 + w^2}}\right) - g$$

以四邊形左右兩邊相等，可求得 z。

$$l + v\cos a = w\sin b + l\cos b + z$$

$$z = v\cos a + l - w\sin b - l\cos b$$

$$z = v\cos a + l - \sqrt{l^2 + w^2}\,\sin\left(g + b\right)$$

從引擎到車輪之間，可分成幾個部分：引擎→變速箱，變速箱→傳動軸，傳動軸→輪軸，輪軸就轉動車輪。在以下分別探討這幾部分中，扭力和加速度的關係。

引擎→變速箱：由引擎開始傳輸至傳動系統之實際扭力必須扣除用來加速所有轉動元件之慣量，在這裡我們只考慮一些主要的元件慣量。應用牛頓第二定律，從引擎輸出扭力 Te，扣除用來加速引擎慣量 Ie ae（Ie＝引擎轉動慣量，ae＝引擎之轉動加速度），而成為變速箱的輸入扭力 T_c 為：

$$T_c = T_e - I_e a_e$$

變速箱→傳動軸：變速箱輸出的扭矩會經由齒輪比 N_t 而放大，但也會由於齒輪及各軸之慣性損失而有所降低，若變速箱之慣量可由其輸入端之值來決定，其損失可表示成 $I_t\ a_e$（I_t＝變速箱之轉動慣量，轉動加速度和引擎相同），那麼輸出至傳動軸之扭力 T_d 可以下式估算：

$$T_d = (T_c - I_t a_e) N_t$$

傳動軸→輪軸：輸入至傳動軸的扭力，扣除用來加速傳動軸轉動的慣量 $I_d\ a_d$ 後（I_d＝傳動軸之轉動慣量，a_d＝傳動軸之轉動加速度），經過最終傳動比率 N_f 放大後，會等於輸入到輪軸上之扭力 T_a，此扭力亦會等於地面上之牽引力 F_x 之扭矩 $F_x\ r$（r＝車輪半徑），加上加速車輪及輪軸之轉動慣量 $I_w\ a_w$（I_w＝車輪及輪軸之轉動慣量，a_w ＝車輪之轉動加速度）。

$$T_a = (Td - I_d a_d) N_f = F_x r + I_w a_w$$

接下來把上列一些式子串起來，讓引擎輸出扭力可以直接和地面上之牽引力做關連，利用齒輪比和加速度的關係把整個式子簡化，其關係為：

$$a_d = N_f a_w \qquad a_e = N_t a_d = N_t N_f a_w$$

又由於車輪前進之加速度 a_x 為車輪旋轉加速度 a_w 乘以車輪半徑 r，因此可得：

$$F_x = \frac{T_e N_{tf}}{r} - \left\{\left(I_e + I_t\right) N_{tf}^2 + I_d N_f^2 + I_w\right\}\frac{a_x}{r^2}$$

其中 N_{tf}＝$N_t\ N_f$。

至於傳動系統諸元件：變速箱、傳動軸、差速器及輪軸，因機械和黏性損失所產生之效率降低，在我們的場景建構之中暫不考慮這個問題。由上列的方程式，可以分成兩部分：（1）右邊第一項為引擎扭力乘以整體齒輪比，再除以輪胎半徑。此分項代表在地面上之穩態牽引力，以此克服空氣動力及滾動阻力之道路負載力量，用來加速或爬坡。（2）右邊第二項代表由引擎及傳動系統各元件慣量所產生之牽引力損失，括弧內之項代表經由元件及車輪間數值齒輪之平方放大後，各元件的慣量。。

以場景中的懸吊系統來說，其主要功能在於：（1）提供垂直的柔度，使得車子在不平坦的道路移動的時候，車子的姿態可以改變的很平順。（2）對由縱向力（加速、減速）、側向力（轉向）及車身傾斜所產生之負載轉移的影響，

做出合適的反應，在懸吊系統上，採用最簡易的方法來模擬它，就是在四個輪子上各加一個彈簧及阻泥來代表它，而加諸在各彈簧上的力為承受車子本身的重力，再加上加速時所產生之力矩的變化。以下分為縱向及橫向負載的轉移來探討。

車子受到縱向力的時候，其受力的關係如圖十四所示，在此忽略空氣阻力、滾動阻力…等，單純只考慮車子受到縱向加速力、地形坡度 θ 的影響，施加給前輪 F_f 及後輪 F_r 之力的變化。

各車軸所承受之負載包含一靜態分量，以及受到其他力量的作用所產生之由前輪到後輪（或後輪到前輪）的負載轉移，可由車輪與地面之接觸點之力矩和來求得。假設圖中車子未在傾斜上加速，那麼對 A 點之力矩和為零，如下所示：

$$F_f L + F_x h + Wh\sin q - Wc\cos q = 0$$

化簡可求得前輪之負載 F_f：

$$F_f = \left(Wc\cos q - F_x h - Wh\sin q\right) \Big/ L$$

後輪之負載 F_r 可由 B 點的力矩和來求出：

$$F_r = \left(Wb\cos q + F_x h + Wh\sin q\right) \Big/ L$$

如圖十五所示為車子右轉時受到橫向力作用時的受力情形，與縱向力的分析一樣，考慮轉向加速力及地形傾斜角 φ 的影響，以得到左側車輪 F_i 與右側車輪 F_o 受到的負載。

左輪之負載 F_i 可由 C 點的力矩和來求出

$$F_i = W\cos f\frac{1}{2} + \frac{h}{t}\left(F_y + W\sin f\right)$$

右輪之負載 F_o 可由 D 點的力矩和來求出

$$F_o = W\cos f\frac{1}{2} - \frac{h}{t}\left(F_y + W\sin f\right)$$

在合併的時候，令 b＝c＝L/2，除了把兩輪受到的負載除以一半（如縱向方面的前輪，要分給左前輪跟右前輪）之外，對應在同一輪的值相加之後也要再平均。另外，兩邊的式子還必須做一點變化：

縱向負載要部分加上路面橫向的傾斜：

W => W cosφ

橫向負載要部分加上路面縱向的傾斜：

W => W cos θ

合併之後的左前輪的負載 F_{if} 為：

$$F_{if} = \frac{1}{4}\left[W\cos f\cos q + \frac{h}{t}\left(F_y + W\sin f\cos q\right) - \frac{h}{L}\left(F_x + W\sin q\cos f\right)\right]$$

右前輪的負載 F_{of}：

$$F_{of} = \frac{1}{4}\left[W\cos f\cos q - \frac{h}{t}\left(F_y + W\sin f\cos q\right) - \frac{h}{L}\left(F_x + W\sin q\cos f\right)\right]$$

左後輪的負載 F_{ir}：

$$F_{ir} = \frac{1}{4}\left[W\cos f\cos q + \frac{h}{t}\left(F_y + W\sin f\cos q\right) + \frac{h}{L}\left(F_x + W\sin q\cos f\right)\right]$$

右後輪的負載 F_{or}：

$$F_{or} = \frac{1}{4}\left[W\cos f\cos q - \frac{h}{t}\left(F_y + W\sin f\cos q\right) + \frac{h}{L}\left(F_x + W\sin q\cos f\right)\right]$$

從這些式子可以看出，在各輪的負載數值上，可分為三個分量。（1）左邊第一分量為車子靜態負載，為車子本身的重量垂直於路面的分量。（2）左邊第二項為車子左右兩邊之動態負載，會隨車子轉向及路面傾斜而加到左右兩側。（3）右邊第一項為車子前後之動態負載，會跟著車子前後加速及路面傾斜而影響前後兩側。

經由推導之後，可以得知在各輪上的負載，與彈簧壓縮 x 所產生之力 kx（k 為彈性係數）及阻泥力 CV，可推算出懸吊系統震盪之加速度。

在虛擬實境場景的建立上，是運用繪圖函示庫 WTK 構成的，在 WTK 的場景中，要得知一 3D 物件與另一 3D 物件的距離，可以使用 WTK 的函示 WTnode_rayintersect()，此函示功能在於給定一個起點座標、偵測方向、及測量物件，從起點循著偵測方向出發，若是有碰到偵測物件，則會給一個距離，否則回傳 NULL。在一般場景裡，要得知車子的四個輪子距離地面的高度，需偵測四次，越複雜的地形偵測時間越久。所以在較複雜的地形時，預先建立好地形高度的資料，將可以節省很多運算的時間，使得場景速度可以加快。

我們所建構的地形資料庫是把欲偵測的地形用一方形區域框起來，把這區域平均切割成 1000X1000 個小區域，再紀錄每個小區域的高度。而在決定用多

大區域框起來之前，必須先求得上下左右的邊界值，先利用 WTK 提供的函示 WTnode_getradius() 得到中心點到最遠端的距離，用此來求出最遠有可能會到達的邊界，接著使用 WTnode_rayintersect()，在每一個小區域的上空由高到低偵測是否有地形存在。

量測左邊界之方法為：從中心點出發，有偵測到地形時，就持續向左邊筆直前進，若是偵測到無地形時，先向上下附近的區域偵測，有地形的時候就繼續從此區域向左邊移動，若是附近都沒有，就從上面最遠的邊界開始向下偵測，若是又碰到地形就繼續前進，若是偵測到最下面的邊界都沒有地形時，其右邊的區域即是最左的邊界了。其他三邊的也是一樣。

找到邊界後，接著就在邊界裡用 WTnode_rayintersect()，由高到低求出每一小區域的高度，在得到所有的高度後，如圖十六 (a) 所示有些地區如圍牆、草叢、森林…等，都得不到其高度的資訊，原因在於他們在建構的時候，都沒有使用到水平的面，這樣由高處垂直往下偵測的時候，自然就碰不到了。為了要克服這個問題，必須要從水平的方向再進行高度資料的修正。

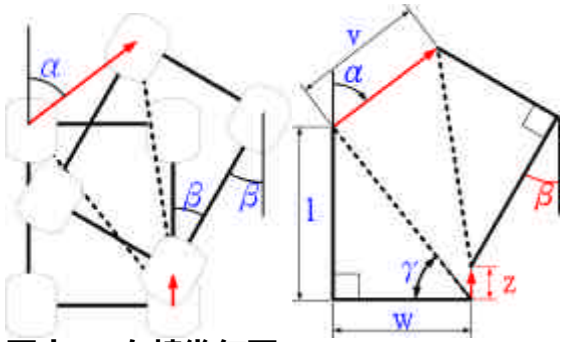
在水平偵測方面，分成上下及左右來回的偵測。如圖十六 (b) 以向左為例：由已建好的地形資料庫中，從最右邊開始向左，先找到一凹地，這凹地邊界的定義為，相鄰兩點的高度差距在一距離以上，通常這差這距離會使得車子無法向上通行。接著從凹地最右邊開始，從此地區的高度加上一距離，當作 WTnode_rayintersect()偵測的原點，向左邊偵測，若是碰到障礙物時，則把偵測原點放到障礙物前幾格做第二次確認，其原點座標一樣是當時的高度加一距離，若是偵測到的距離和障礙物的座標吻合時，即可以把他的高度加到地形資料庫裡，加上去的高度只要是車子無法通行的高度即可。接著移動到障礙物的後面，向左繼續做重複的動作，直到碰到凹地的最左邊為止，之後再尋找下一個凹地，重複以上的偵測直到地形最左邊的邊界。做第二次的確認動作，可以防止在偵測時碰到坡地或其他特殊的情形，而產生判斷錯誤。

圖十七（a）是我們採用的場景之一，圖十七（b）為第二次製作地形高度的結果，顏色從藍到紅為高度愈來愈高，白色的部分是空的部分。而圍牆、草叢、樹林也確實加入到高度資料之中。

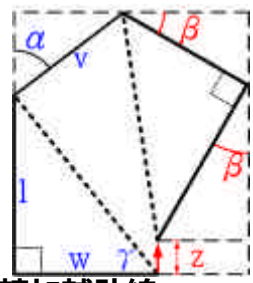
直接偵測場景和使用地形高度資料的比較，在同一部電腦中使用預先建立好地形資料庫的場景，每秒鐘會比直接偵測的多 10-25 個畫數（frame），只是在建地形資料的時候會很久，不過只需建立一次，建好之後輸出到檔案之中，下次開啟時就可以直接讀檔輸入了。其缺點是當遇到騎樓、涼亭、地下室…等地形，就會受到限制，必須把地形的 3D 物件做修改，讓他可以偵測到車子可走的地區，若是直接偵測的方法，將不受此限制。

在另一方面，地形高度資料可以讓使用者很快的知道附近的地形，讓他可以多做一些應用，若是要增加多一點的動態方程式在車子上，直接偵測地形的將會變的更慢。

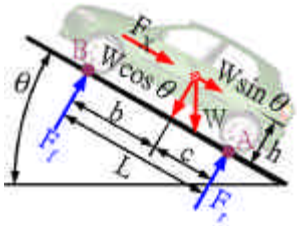
圖十八為開發場景之一的畫面。



圖十二 右轉幾何圖

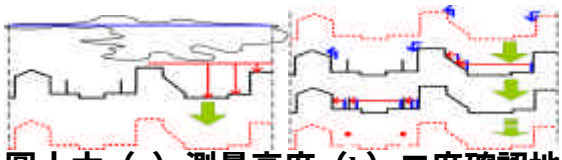


圖十三 右轉加輔助線

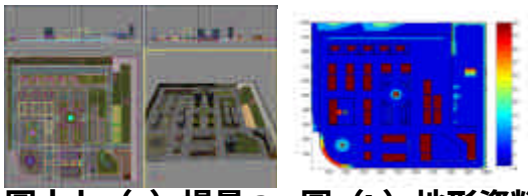


圖十四 車身受縱向力關係圖

圖十五 車身受橫向力關係圖



圖十六（a）測量高度（b）二度確認地形



圖十七（a）場景 3D 圖（b）地形資料

圖十八 開發場景

B. OMAP 嵌入式即時系統

在人們的生活中，嵌入式系統應用於訊號處理上十分廣泛，如多媒體語音與影像資訊處理等。為了讓訊號得到較佳的處理，往往需要大量的數學運算，但有鑑於嵌入式系統在運算能力不足，所以採用同時具有 ARM 和 DSP 微處理器之 OMAP 來彌補這個問題。OMAP 是由 TI 針對嵌入式系統所開發的一個具有優秀運算能力的微處理器，使用 OMAP 架構可以使數學運算上更為迅速有效率，且 ARM 和 DSP 可以同步處理各自不同的行程。

隨著多媒體與無線通訊的發展，訊號處理演算法不斷更新。為了追求更為真實的感受，在多媒體系統上必需做出更為精確的運算。對於腦波生理訊號處理應用，在生理訊號分析技術層面上，運算處理上要求更為嚴苛。就軟體的層面而言，需要一個好的演算法與良好的計算程序；就硬體的層面而言，處理器運算之能力亦是關鍵。大型主機，空間大且昂貴，為了縮小體積，嵌入式系統的發展勢必是個趨勢，強調小與省電。一般較有名的嵌入式系統的微處理器架構有 ARM、MIPS 等。雖然這些微處理器相當省電、運算能力不錯且支援多種作業系統，但處理大量的數位訊號資訊上，還是稍嫌不足，為了使系統運算能力更為強大，部分廠商引入了 DSP 微處理器來加強嵌入式微處理器的運算能力。

實驗室早期將既有虛擬實境發展之技術，已經由工業電腦逐漸轉至嵌入式單板電腦(SBC，Signal Board Computer)的應用。DSP 具有優越的計算能力，硬體精簡，高效率，可即時處理大量訊號等特性，但不足的是缺乏作業系統的支援，且 DSP 系統軟體程式移植不易，故採用 LART 嵌入式單板做為開發環境。LART(Linux Advanced Radio Terminal)是以 StrongARM 為微處理器的嵌入式單板，具有不錯的執行效率，體積小，支援多種作業系統，同時具備低成本的優點，採用這種精簡的嵌入式硬體單板可以取代原來的電腦和 DSP 控制卡。就其作業系統而言，是採用 ARM 架構的 ARMLinux，具有 Linux 作業系統開放原

始碼的特性，在許多資料整合上，都能得到協助。但隨著演算法複雜度的增加，在運算處理上，以 ARM 架構為主的 StrongARM 微處理器，處理速度漸漸不合所需。

目前為了處理日益複雜的訊號處理的演算法，嵌入式單板勢必面臨大量的特殊運算，但一方面希望在嵌入式單板上，保有 ARM 處理器的移植性，發展性和作業系統的支援，另一方面，又希望加入 DSP 所擁有的優秀計算能力，因此，採用德州儀器(TI)所發展出的新處理器 OMAP。OMAP 結合了 ARM 和 DSP 的長處，支援多種作業系統，並且在運算上能使用 TI 所開發的 DSP 技術。就嵌入式的作業系統而言，以原先的 ARMLinux 為藍本，加入對 TI DSP 系統的支援，成 DSPLinux。裡面包含了對於 OMAP 處理器上 DSP 部分的驅動，和 ARM 與 DSP 溝通的通道。於此，使得整個系統依然可以擁有開放原始碼的特性進行整合，容易維護開發，並且提升在運算上之能力等優點。

OMAP 為 Open Multimedia Architecture Platform 的縮寫，是一個有雙核心的微處理器，使用 TI-enhanced ARM925 微處理器並結合 TMS320C55x DSP 核心。OMAP 具有高效能平衡以及低功率消耗的能力，且擁有極佳的數值能力。程式開發上，Linux 平台下的跨平台環境(Cross Compiler)開發 ARM 微處理器相容的應用程式。DSP 端的程式開發，則主要是以 TI's CCS(Code Composer Studio)做為開發環境，使用 DSP 多執行緒的作業核心—DSP/BIOS 所提供的功能開發 DSP 應用程式。

基本上 OMAP 架構微處理器內部的 DSP 微處理器核心與 ARM RISC 微處理器核心分別由兩個不同的系統所控制，DSP 微處理器核心是採用 TI 的微核心多工即時的系統，稱為 DSP/BIOS。提供軟體工程師方便開發符合即時運算效率的軟體元件工作。在 ARM 部分，一般常看到的嵌入式作業系統，Linux 和 WinCE 等都有支援，可以用來控制整個系統的運作。由於兩個部分分屬於不同的系統所管理，為了發揮由 ARM 操作和 DSP 運算的分工效能，兩個處理器之間的資料流通就成為研發重點，TI 針對這點提出 DSP/BIOS Bridge 架構，能夠讓應用程式開發人員在雙處理器架構下撰寫程式。

在一般 PC 的 Linux 的環境下開發程式，採用 gcc(GNU Compiler Collection)編譯器(Compiler)。以 ARM 而言，若想要在 x86 環境下開發 ARM 程式需要建立一個跨平台開發的工具（cross-platform development tool）。對於 ARM 程式的開發，採用 GNU 的工具鏈（toolchain），對 ARM 平台的 cross-compiler(交叉編譯)—arm-linux-gcc 來進行程式的編譯。arm-linux-gcc 將程式碼針對 ARM 的架構進行編譯，產生出來的執行檔便可以於 ARM 的架構上執行。

CCS (Code Composer Studio)是德州儀器所提供用開發 DSP 程式的軟體，包含了跨平台的開發工具，CCS 提供一個完善的整合發展環境(IDE，Integrated Development Environment)。CCS 包含了程式編譯器、模擬器以及程式除錯器，藉由這些工具可以輕易地開發 OMAP 的應用程式，在撰寫 DSP 的應用元件時，透過 CCS 可以很容易在 DSP/BIOS 上整合支援影像及音訊資料處理的套件。

具有強大運算功能的 DSP 處理器系統中，如何有效地利用 DSP 的資源，並在一複雜的系統中，如何做到即時多工處理，是必定面臨的一個問題。TI 的 DSP/BIOS 發展環境的核心，就提供了一個多執行緒(Multi-thread)的管理系統，讓使用者能在 DSP 上發展出一套嵌入式即時系統(Embedded real-time system)。CCS IDE 裡面 DSP/BIOS 具有即時核心(Real-time kernel)的特性，可以快速且簡便地設計發展出複雜的應用。

OMAP1510 微處理器結合 ARM925T 和 TMS320C55x DSP 處理器，但是要如何將這兩個處理器的功能

相互結合應用，是一個問題所在，DSP Gateway 提供了一個解決的管道。DSP Gateway 是一個機制，能使 OMAP1510 微處理器能在 ARM 和 DSP 之間溝通，使得系統能同時使用 ARM 以及 DSP 的資源。如此一來能在 ARM 的架構上建立一個 Linux 作業系統，同時又可以使用 DSP 的運算能力。將針對「DSP Gateway 架構」、「DSP Gateway Linux API」、「DSP Program」，將 ARM 和 DSP 溝通的機制逐一的說明：

DSP Gateway 主要可分成 DSP Gateway 架構和程式編寫的方式，兩個部分來探討之。其中包括「DSP Driver 與 Linux API」、「Mailbox 傳輸命令機制」與「DSP Gateway BIOS(tokliBIOS) 與 DSP APIs」。DSP Driver 與 Linux APIs 包含 ARM 的環境下提供與 DSP 溝通操作的方式及介面。Mailbox 機制，將 ARM 與 DSP 之間操作動作所傳送的 Mailbox 命令做解譯，並且傳送資訊，其中包括傳送的 DSP Task ID、傳送方式以及傳送的 buffer 位置。DSP Gateway BIOS 與 DSP APIs，主要是與 DSP 溝通和 DSP 系統管理機制有關，建構於 TI 的 DSP/BIOS 提供的即時多工的核心和 APIs 所建構而成的。圖十九可用來表示 DSP Gateway 整個架構的關係圖。

DSP 處理器，利用給予不同的 TID(Task ID)來區分不同的工作，達到處理多工(Multi-Tasks)的效果。由表三可以知道在傳送命令時包含 TID(包含在 Command Low 的部分)，如此 MPU(MicroProcessor Unit)便能處理混合 TID 的資料。處理器內部 ARM 和 DSP 的資料傳輸是透過內部的 buffer(IPBUF，InterProcessor Buffers)。利用 IPBUF 做處理器內部 ARM 和 DSP 之間區塊(block)資料傳輸的 buffter 時，IPBUF 必須先將 IDs(BID)先定義出來，且 IPBUF 不同於一般的區塊，僅有處理器內部，才可以使用。當使用 IPBUF 傳輸資料時，IPBUF 的所有權亦會隨著資料轉移，跟著轉移至接收者。IPBUF 的 BID 資料亦被定義於 Mailbox 中資料暫存器內，會隨著 Mailbox 傳到 DSP，然後 DSP 便知道要到那一個 IPBUF 去取得，所需要的處理資料。圖二十表示，Mailbox 和 IPBUF 在 ARM 和 DSP 之間的關係。除了表示了資料傳輸之間的關係外，還可以看出其流程，ARM 端的 user program 要將資料交給 DSP task 處理時，也要將資料傳給 DSP 端。傳輸過程中，一開始要傳送 Mailbox command 和 data 的資訊(BID)給 DSP，同時將要傳送處理的資料寫入 IPBUF，這時在傳送的 data 資訊中將知道 IPBUF 的 BID，並且和命令一起傳送給 DSP task，這時 DSP task 便會知道要去那一個 BID 取得資料來進行處理。

在圖十九架構圖中的 DSP Gateway driver 的部分，一般而言，使用者想要使用 DSP 工作時只需要存取 DSP task device。DSP device 是利用 devfs 的機制，在使用時才動態產生的系統。建立 DSPLinux 系統時，不需要特別建立這些裝置檔。裝置的溝通是透過作業系統的系統呼叫(System Call)，透過驅動程式，參考裡面的檔案操作(File Operations)定義，來對特定的裝置做輸入輸出 I/O。這些輸入輸出的函式，一般較常用的包括有 open、close、write、read、poll、selet 和 ioctl。

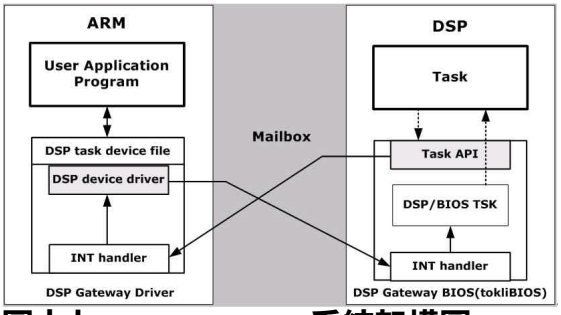
DSP 程式設計者在設計 DSP 工作程式時，必須先對工作的型式做個定義，表三表示了一個 DSP 工作所需要定義的結構。有工作 ID，工作名稱，工作型式，和一些有關於 DSP I/O 及 DSP 功能的函式使用定義。其中 DSP 最大的工作數能到 126(TID 0x00~0xfd)，因為 TID 0xfe 和 0xff 是接收 IPBUF 所有者的資料特別用途。

ARM 和 DSP 分工中，一項重要的部分便是 ARM 和 DSP 的資料傳輸。資料傳輸的型式，所採取的方式是 block send 的方式。但是在傳輸時 IPBUF 的大小有限，一個 IPBUF 僅有 128words 的大小，也就是說，會有資料無法一次傳輸，

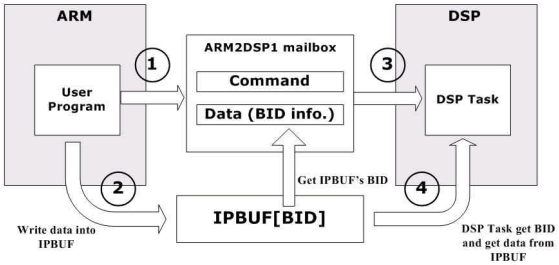
需要分批傳輸。將資料分一段一段傳輸。定義一個資料封包格式，將傳輸資訊和資訊藉由封包傳輸，可以達到分段傳輸的目的。表四是傳輸封包的定義。圖二十一和圖二十二分別是資料傳送和接收的順序圖。

ARM 傳資料給 DSP 了傳輸要分批傳之外，另外，需要注意的一點是 ARM 和 DSP 溝通時如果使用 32bit 大小以上的資料區塊，需要先經過資料位元的置換，這是因為 ARM 和 DSP 資料的堆疊方式不同。TI 的 DSP 為了支援於 Intel-base 的 x86 系列溝通，在設計時是採用 little endian 的模式，但是 ARM-base 是屬於 big endian 的順序，和 DSP 順序不同。整個系統架構和資料傳輸的設計，表示於圖二十三。圖二十三中，整個順序分別以數字標號表示，1~6 的流程表示，單板收到訊號資料，執行 ICA 程序，將 ICA 運算交由 DSP 處理，DSP 再將處理完的資料更新至 IPBUF 中，7~9 的流程表示，ARM 的部分給予一個接收結果的 Mailbox 命令，然後從 IPBUF 中接收傳回的命令，並且在 10 的地方得到 ICA 分離出的成分。

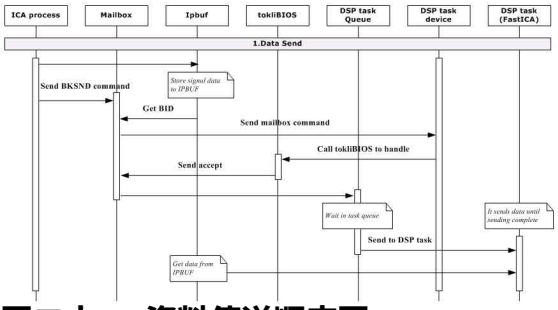
圖二十四為 OMAP 硬體架構圖。



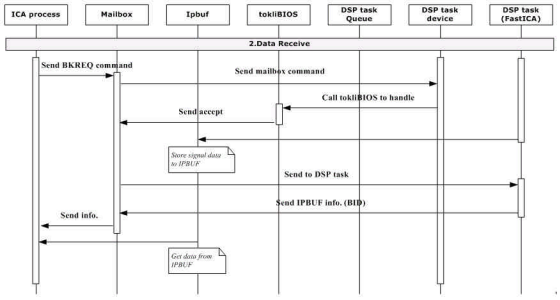
圖十九 DSP Gateway 系統架構圖



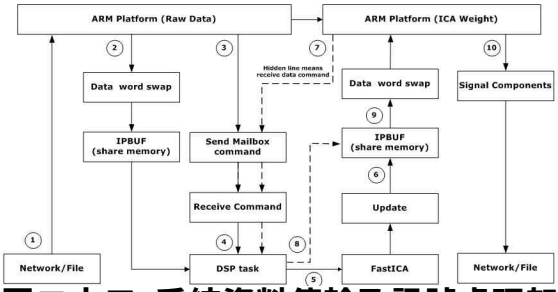
圖二十 ARM 與 DSP 命令及 IPBUF 傳輸



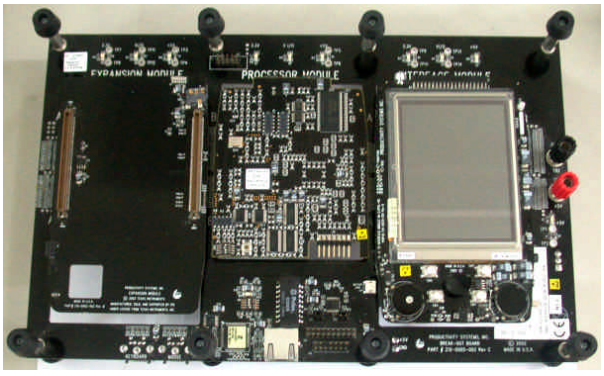
圖二十一 資料傳送順序圖



圖二十二 資料接收順序圖



圖二十三 系統資料傳輸及訊號處理架構圖



圖二十四 Innovator Development Kit 的 Break Out Board 配件模組

表三　　dsptask 結構

Structure	*Data member*	*Data type*	*Description*
Dsptask	tid	unsigned short	Task ID，DSP 工作的 ID，當工作初始化時，DSP Gateway BIOS 設定 TID。
	name	String	Task name(工作名稱)，亦為 device file 的名稱
	ttyp	unsigned short	Task type，工作的型式設定，例如：BKMD(block transfer for MPU→DSP)
	*rcv_snd()	unsigned short	Task function 。DSP Gateway 會呼叫這些函式，回應 ARM 的 Mailbox command。
	*rcv_req()	unsigned short	
	*rcv_tctl()	unsigned short	
	*task_attrs	struct TSK_Attrs	DSP/BIOS TSK 特性。如果為 NULL，則為預設值。

表四 資料傳送封包結構

Structure name	*Data member*	*Data type*	*Description*
DATA_T	data_startnum	unsigned short	傳輸資料的起啟點
	data_endnum	unsigned short	傳輸資料的結束點
	signalnum	unsigned short	第幾筆訊號資料
	finishflag	unsigned short	是否傳輸完成
	signaldata[]	float	訊號資料

三、結論

本子計畫為了配合整個整合型計畫執行，在前二年完成三大部分，研發重點分別如下：

(1) 開發一適應性小腦模型類神經網路控制系統做為動作平台之姿態控制系統。

(2) 為了提高系統的穩定度與安全性及系統對外界訊息的反應，以確保在系統出現問題時，能立及做出適當的處理。我們提高系統中的即時性作業能力。

(3) 以 LART 實驗單板的 Intel StrongARM SA-1100 處理器之 Linux 嵌入式系統來取代舊有的 IPC 控制方式。

而今年本子計畫配合整個整合型計畫之執行，研發重點為：

(1) 開發一動態模擬駕駛系統以結合動作平台的姿態控制，以為整個動態運動訓練輔助系統的虛擬互動場景。

(2) 為了提高系統的穩定度與安全性及系統對外界訊息的反應，以確保在系統出現問題時，能立及做出適當的處理。使用同時具有 ARM 和 DSP 微處理器之 OMAP 之高運算能力嵌入式系統。

四、參考文獻

1. C. T. Chiang, and C. S. Lin, "CMAC with general basis functions," *Neural Networks*, vol. 9, pp. 1199-1211, 1996.
2. Y. H. Kim, and F. L. Lewis, "Optimal design of CMAC neural-network controller for robot manipulators," *IEEE Trans. Syst., Man, and Cybern.*, vol. 30, pp. 22-31, 2000.
3. W. Q. D. Do and D. C. H. Yang, "Inverse dynamics analysis and simulateion of a platform type of robot", *Journal of Robotics Systems*, Vol. 5, pp. 209-229, 1988.
4. G. Lbret, K. Liu and F. L. Lewis, "Sigularities and dynamics of a stewart platform manipulator", *Journal of Intelligent and Robotics System*, Vol. 8, No. 3, pp. 287-308, 1993.
5. Thomas D. Gillespie 著，車輛運動力學，林筱增譯，成陽出版社，台北，2002 年 5 月。
6. Toshihiro Kobayashi, Kiyotaka Takahashi, ***Linux DSP Gateway Specification*** Rev2.0, Nokia Corporation, November 13 2003.
7. ***Innovator Development Kit for the OMAP Platform User's Guide***(SPRU667), Texas Instruments.
8. ***Innovator Development Kit for the Texas Instruments OMAP™ Platform Deluxe Model User's Guide***, Texas Instruments.
9. ***TMS320 DSP/BIOS User's Guide*** Rev B(SPRU423B), Texas Instruments.
10. ***TMS320C55x Optimizing C/C++ Compiler User's Guide*** Rev E(SPRU281E) Texas Instruments.
11. ***Code Composer Studio Getting Started Guide*** Rev C(SPRU509C), Texas Instruments.
12. Alessandro Rubini, Jonathan Corbet, ***Linux Device Driver*** 2nd Edition, O'Reilly, June 2001.
13. Karim Yaghmour(2003), ***Building Embedded Linux Systems***, O'Reilly, April 2003.
14. W. Richard Stevens, ***Advanced Programming in the UNIX Environment***, Addison-Wesley, June 1992.
15. Kay A. Robbins, Steven Robbins, ***UNIX Systems Programming-Communication, Cocurrency, and Threads***, Prentice Hall, 1996.
16. Ashfaq A. Khan, ***Practical Linux Programming：Device Driver, Embedded Systems, and the Internet***, Charles River Media, 2002.
17. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, ***Operation System Concept***, John Wiley & Sons, 2003.