國立清華大學 電機工程學系

實作專題研究網頁成果摘要

Smart wearable sensor

智慧型穿戴式感測器

專題領域：系統組

組　　別：A97

指導教授：馬席彬

組員姓名：葉冠霆

研究期間：109 年 9 月 13 日至 110 年 5 月底止，計 9 個月

報告摘要

目的：

本研究旨在六軸感測器之改良精進，為了避免過去感測器在接收資料後，透過藍芽將數據傳入電腦端時，因過程中的人為和環境因素，導致藍芽連線不穩而造成中斷的情形，最終無法得知中斷期間內所量測的數據。因此，本研究將透過改變讀取資料的狀態，在藍芽中斷期間之內，以快閃記憶體持續性地接收資料，並在藍芽恢復連線之後，再從記憶體中提取數據，並透過藍芽傳入電腦端以進行分析。

方法：

首先確認感測器本身的精確程度，包含三軸加速計和三軸陀螺儀，一開始做靜止測試，確認其設計的平衡點為何，而後透過自由落體、等速運動、小角度單擺等實驗驗證此感測器的數據有無異常，是否符合預期的行為，確認完成後進行取樣率的調整，一開始以最高速率進行取樣，分別對加速度、轉速進行探討，經多次取樣後取平均值，接著分別調整取樣率，使其逐漸下降，同時進行以上測試，取平均值後作圖分析，得到不同取樣率對時域軸上加速度的變化情形，並可做出連續與不連續資料的比對分析。

接著進行取樣率上升的改良，原本的 MCU 是透過 I2C 串接埠運作，負責連接 icm 20649 sensor 和 MSP432，作為彼此資料傳輸和控制的橋樑，採取雙線的從主架構，一條線為傳輸資料的串列資料線（SDA），另一條線是啟動或停止傳輸以及傳送時鐘序列的串列時脈（SCL）線，但這樣的設計會造成傳輸速率降低，最高取樣率也會因此降低。倘若以 SPI 串接埠的方式進行設計，透過四線傳輸，並在裝置之間使用全雙工模式通信，是一個主機和多個從機的主從模式，透過這樣的設計，可以使資料傳輸的速率上升，也能使取樣率大幅提升，此即為第一階段的改良。

第二階段的改良為目的中所述，在測試過程中，即便藍芽連線中斷，也能透過重連後提取 flash memory 裡面的資料進行分析。此外將外接壓力感測器，透過不同的壓力變化，轉換資料的讀和寫，即可不用透過 PC 端而決定感測器的量測狀態，最後透過 Matlab 作圖進行分析，得出結論。

結果：

一開始對感測器進行驗證，發現結果符合預期，唯雜訊較多，受到一些干擾程度，數據傳輸過程中容易受到藍芽傳輸和電腦接收端的影響，但是對整體而言的行為並不影響。

其二，SPI 序列埠完成硬體方面的設計，透過外接的線路和設定軟體上的接腳位置，完成四線串接埠的傳輸；至於軟體方面則是完成腳位設定和序列埠串接的部分，然 MCU 中的設定尚未完成，因此未能有效地提升取樣率。

最後是快閃記憶體提取方式的改良，順利完成資料傳輸和驗證的部分，外接壓力計即可直接在感測器上進行不同狀態的轉換，例如棒球投手投球時，能夠直接在球上設定，並且立即得到資料，亦不受投球距離過遠時藍芽中斷，導致資料遺失的情形。

報告內容

觀察測量:

本感測器使用的是 TDK InvenSense ICM-20649，搭配的 MCU 為 TI MSP432P401R，其為德州儀器(TI)的混合信號微控制器系列，與 MSP430 相比，具有更大的數據地址空間以及更快的整數和浮點數計算能力，功耗也較低，適合作為本研究之搭配。在過去的專題研究中已經完成本感測器之測量與應用，因此本節重點只在於簡易的觀測，完成定性定量的檢查。

其運作過程為，將藍芽接收器裝置於電腦上，安裝 Node.js command prompt，同時安裝 Zadig 以替換藍芽接收器的韌體，搜尋並找到對應的感測器型號，接著透過 command prompt 執行 JavaScript file，完成藍芽的初始化並開始連線，接著數據將顯示於 command prompt 中，同時，這些數據也將在 JavaScript file 中同步進行處理，並將初步的數據儲存在.txt file 中，最終透過 Matlab 換算數據並作圖分析。

首先進行靜止的測試，將感測器靜置於桌面，進行上述的測試，最終透過 Matlab 作圖，可以得到三軸的加速度(g)對時間(t)的關係圖。從 acc 作分析，可以發現 x、y、z 的初始值皆不同，因為 x 和 y 的值很接近 0，推測 x 和 y 是蒐集數據後重設值的誤差，而 z 則是接近 1g 左右($g = 9.8m/s^2$)，推測是有意將 az 設定初始值為 1g，在計算上必須再進行扣除。

接著進行自由落體的實驗，設定落下距離為 0.6m，根據等加速度公式計算得到的落下時間為 0.35s，重力加速度 $g = 9.8m/s^2$，經過多次實驗取平均，將數據進行作圖，可以發現在 az 上的變化，在一秒處開始落下，acc 的值由 1g 轉變為 0g，符合需要扣除初始值為 1g 的情況。

從上述的兩個實驗為例，可以發現加速度的部分大致符合預期的行為，但實驗中存在各種環境上的變因，像是阻力、重力場大小和軟體測量上的干擾，甚至是感測器本身的精確度，都會導致數據上出現誤差，因此在實驗上能有效排除環境上的干擾是非常重要的。

原理分析:

整個 project 皆放置於 I_Baseball 的資料夾中，在了解內部的功能和原理前，必須先安裝 CCS7.4.0.00015_win32，即為 Code Composer Studio，用來燒入和執行程式碼，使用的是 C 語言進行編寫。

由於其版本相容性的問題，導致在安裝時出現大量錯誤，如安裝路徑問題、general property setting、RTSC problem 等，為解決這些問題花費了一段時間。此外，需要安裝 TI XDC tool、SDK、tirtos 等工具包，其內含 TI 所架構的 header file 和 source 以及編譯環境的設定，完成上述步驟後即可開始了解內部程式碼架構。

整個 project 包含了 flash、藍芽(BLEsend)、MCU 和 sensor 等，以其中幾個檔案為例，flash driver: 每個 sensor 都有自己的 driver file，用這個檔案寫其 driver，而 flash

用的是 SPI control。MSP_EXP432P401R: 所有接腳的 information 都會記錄在這裡，包含相關的參數、位置。sensor_boosterpack_icm20649: 這裡會進行訊號的取樣和處裡分析，可以調整 sampling rate、sensor data 和其他 parameters。

第一步要改變 sampling rate，首先要先了解資料傳輸的過程，可以在 sensor_boosterpack_icm20649 找到 Interrupt_Handler_Taskfuction，其中的 accRead 和 gyroRead 分別呼叫出處理加速度和轉動的 function，並在其中接收來自 sensor 的 data，為 sendData。這裡 sendData 的 for 迴圈總共跑了 6 次，原因是 acc 和 gyro 分別有 x、y、z 三軸，而每筆的 data 都占了 16 個 bit，也就是 2 個 byte，因此總共需要跑 6 次；sendData 從 2 開始是因為前面設定了這筆資料的辨識碼，因此設定[2]~[7]為 acc、[8]~[13]為 gyro，而[14]~[19]為 force sensor 的部分，這部分暫不考慮。

接下來開始傳送 data，enqueue 的部分是將 data 傳入藍芽，而 sendtoStore 則是將 data 送入 flash，以 sendtoStore 為例，透過 sem_wait 到 sem_post，進入 getdata 的 function，flash 開始收資料，並將資料送入 Taskfuction 中。接著透過 hextostr，將 sendData 從 16 進制轉變為 string，以此才能夠 write in flash。而 cnt 為 counter，假設這裡的 sampling rate 為 1125 Hz，當開始計時後 flash 開始收 data，直到 cnt = 11250，也就是 10 秒後，才將 data output。最後將 string 轉回 16 進制，即可完成 read 的部分。而在藍芽的部分，則是透過 enqueue 和 deqeue 完成 data 的輸入，並送入 Taskfuction 中，概念和 flash 類似，最後進入到 setParameter 中，即為藍芽 GATT 的部分。

在 BLEsend.c 可以透過 usleep，設定藍芽的 sample rate，例如當 parameter 為 5000，代表 sampling rate 為 $10^6/5000 = 200$Hz。至於 flash data 的 sampling rate 可以從 icm20649 datasheet 中得知，計算公式為，acc 最快為 1125Hz，gyro 最快則為 1100Hz，然而這也代表了直接從藍芽傳輸的速率遠小於間接透過 flash 傳輸。

當 sampling rate 下降時，波形可想而知會變的較為曲折離散，若持續降低，根據 Nyquist theorem，sampling rate 必須大於被取樣訊號頻寬的兩倍，否則會出現混疊的現象(aliasing)，也就不能從訊號取樣中重建回原始訊號。

既然 sampling rate 有其下限，那其上限該如何提高?若要實現最高 4500Hz 的 sampling rate，根據 datasheet 可以發現以往使用的 I2C 排流必須改為 SPI 排流。

I²C（Inter-Integrated Circuit）其實是 I²C Bus 的簡稱，中文名為積體匯流排電路，它是一種串列通訊匯流排，使用單一或多個主從架構，也是本感測器原先使用的模式。

使用 I²C Bus 的優點很多，例如：只使用了兩根電線支持多個主伺服器和多個從伺服器、ACK / NACK 位可以確認每筆 data 都已成功傳輸、硬體沒有 UART 那麼複雜等，然其缺點也非常顯著，例如：數據傳輸速率比 SPI 慢和數據幀的大小限制為 8 位等，因此，若要提高傳輸速率及 sampling rate，勢必得將 I²C 轉換成 SPI Bus。

SPI 為串行外設介面（Serial Peripheral Interface Bus），是一種用於晶片通信的同步串行通信四線式介面規範，主要應用於單晶片系統中。是一個主機和一個或多個從機的主從模式，也是目前想要設計的模式。

整體而言，SPI 是全雙工模式，速度遠高於 I2C，而且結構相當的直觀簡單，容易實現，並且有很好擴展性。缺點是系統需要額外的邏輯和線路，同時構建特定的通信

協議軟體，需要花費額外的成本。

系統設計:

首先實行 SPI 軟體改寫，在 MSP432P401R.c 中進行接腳的設定，參考其 datasheet，這裡用到的是 PZ 的 package，共有 100 個 pin 腳，其中 SPI 的 eUSCI (Enhanced Universal Serial Communication Interface) 兩邊皆為 4 pin。

在其 datasheet 中提到 " In 4-pin master mode with UCSTEM = 0, UCxSTE is a digital input that can be used to prevent conflicts with another master and controls the master. If UCSTEM = 1 in 4-pin master mode, UCxSTE is a digital output. In this mode the slave enable signal for a single slave is automatically generated on UCxSTE."

其中需要增加 gpioPinConfigs 和 SPIMSP432DMA 兩個部分完成連線，其中的 gpioPinConfigs 就是在處理 General-purpose digital I/O，設定這些 pin 腳的功能和用途，並且進行 configuration。同時加上 MSP_EXP432P401R_SDSPI_CS2 進行設定，新增 SPI 的 CS (Chip Select)，讓它可以從多個 slaver 中選擇一個來回應 master 的請求。

接著是硬體接線的部分，可以發現整個電路板大致可分為 Power、MSP432、SPI flash、cc2640 和 sensor 共 5 個部分。

- Power: 通過 USB 接口來接收電能，例如 TXS0102 為 level shifter，將不同邏輯電路的電壓進行雙向轉換；FT230X 是 IC 的 USB port；TPD4S012DRY 則是用來當作靜電保護(ESD)的電路元件，避免用戶接觸到連接器時產生靜電等。
- MSP432: 這裡提供了 MSP432 的接腳和接線的關係，其中的參數、名稱都可以透過 PZ100 的 datasheet 進行確認。
- SPI flash: 使用 SPI control，這裡使用的型號是 GD5F2GQ4UF9IGR，總共有 8 pin，包含 VCC、GND、CS、SCLK 和 4 個 Serial I/O，其中的 CS 就是 chip select，也就是 SS 的部分，active low；而 Serial I/O 則是 MOSI 和 MISO 的部分。

實際接線的結果如 Figure1 所示，主要多了 4 條線，分別為 CS、SCLK、MOSI 和 MISO ，利用焊錫的方式將它們位置接在一起，電路圖上則為 INT1、INT2、SDI、SDO、NCS 和 SCLK 這六個部分，將其連接到 MSP432 的 SPI ports，將這 4 線分別接到對應位置: SDI (MOSI) 對應 TP7 port；SCLK 對應 U9 port；nCS (CS)對應 TP31 port； SCO (MISO)對應 U4 port。其中 U4 和 U9 連接到 STE amplifier 的部分，完成 SPI 接線的部分。

Figure1. PCB for SPI control

接著改寫 flash control 的部分，目前的 sensor 是在收到 data 候 10 秒才傳入 flash，之後再透過藍芽 output PC 端，但缺點是它必須是當藍芽 connect 時才能運作，一旦藍芽連接中斷，所有的 data 不僅會中斷，flash 所收的 data 也會被 clear，如果收 data 的時間不到 10 秒，當下次再次連接藍芽時，flash 會繼續收完剩餘的秒數，無法重新開始。

希望透過改寫 flash control，改變的 flash initial 的條件，讓它在藍芽中斷時能夠保有裡面的 data，當下次藍芽連接時再把 data 送入 PC 端。因為目前使用的藍芽為 Esense D704 的版本，有效接收距離為 50m，如果新增上述的模式，可以避免例如長距離投球時可能導致藍芽中斷，接收不到後續 data 的問題。

從 flash task_function 可以看到 task_initialization 的部分，也就是每次連接藍芽時都會清除 data，因此第一步會從 initial 的部分進行改寫，先將數據保留在 flash，再丟出來 debug。當藍芽的 initial 設定完成，並且確認使用的 sensor 資訊，會將 sensor call out，開始收 data，並且透過 sendtostore 將 data 送入 flash memory。

Flash data 的 initial 其中之一其實是來自 write 的部分，flash write 裡面有 erase function，其中包含了 clear data，會在每一筆 data 寫入時，將相同 address 的 data 清除，也就是每當重新連接藍芽並寫入時才會將 data 清除。

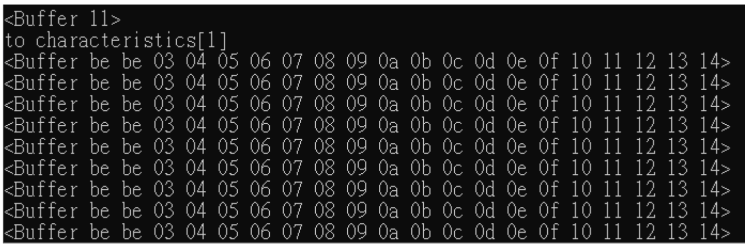
此外，在定義藍芽和 USB 連接埠的 file 中有關於 data 的清除，也就是在 AP taskfunction 中有個 all_sensor_disable，其中有個 closeflash 的 function，裡面有關於 flash write 和 udAddr 的關係，也包含了 data clear 的部分，因此修改了上述部份解決資料清除的問題。

接著設定兩種狀態，分別為 0X11 和 0x13Buffer，當處理 0X11state 時，會進行原本的讀寫；反之，當處理 0X13 state 時，只會 read out flash data。

最後則是 force sensor 的部分，透過外接壓力計，能夠在輸入端進行狀態設定，能夠在兩狀態中自由地切換，省去了間接使用 PC 端的麻煩，這裡使用的是 Flexiforce A301 Force Sensor，它能夠有效的測量出相對變化值和外加負載，可以有效的測量出變化率識別力閾值和觸發適當的行動，將壓力轉化為電阻監測接觸或是觸摸薄膜力量，在一些對測量空間有限的地方得以發揮它的優勢，也相較於傳統大型的壓力感測器還來的適用。

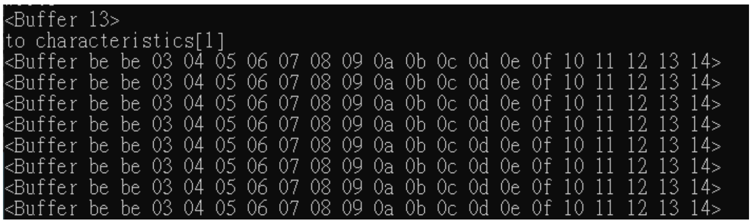
結論:

可以發現 0X11 state 和 0X13 state 皆可正常運作，彼此之間也能夠自由轉換，進一步作定值分析，給定固定 input，可以發現其資料皆正常運作，數值並未因藍芽中斷或是重連而有所變化，接著進行盲測，將 sensor 固定靜置於桌面，觀察兩狀態之間的數據變化，並透過作圖分析，得到兩者平均值接近，再次驗證其數據並不會因此而受到改變。



```
<Buffer 11>
to characteristics[1]
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
```

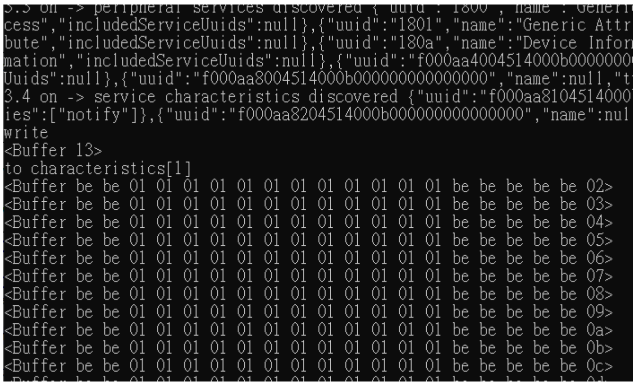
Figure2. Data of 0X11 state



```
<Buffer 13>
to characteristics[1]
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
<Buffer be be 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14>
```

Figure3. Data of 0X13 state

接著進行第二步的驗證，將資料進行編號，測試其是否因為狀態之間的轉換，或是藍芽連接中斷而導致資料提取順序錯亂。首先固定 input 為 0X01，設在 sendData [2]~[13] 的 acc、gyro data，同時將最後一項[19] 設為每筆資料的編號，並且將 sensor getdata function 拿掉。同時為了方便檢驗，將 0x11state 的 read 拿掉，0x13 維持 read，結果如下圖，可以發現整體的編號順序是正常的，但是前兩筆 data 會不見，推測是在 pthread 同時進行序列時，收 data 的速度過快導致轉換時會漏掉前面的 data。



```
cess","includedServiceUuids":null},{"uuid":"1801","name":"Generic Attr
bute","includedServiceUuids":null},{"uuid":"180a","name":"Device Infor
mation","includedServiceUuids":null},{"uuid":"f000aa4004514000b000000
Uuids":null},{"uuid":"f000aa8004514000b000000000000000","name":null,"t
3.4 on -> service characteristics discovered {"uuid":"f000aa8104514000
ies":["notify"]},{"uuid":"f000aa8204514000b000000000000000","name":nul
write
<Buffer 13>
to characteristics[1]
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 02>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 03>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 04>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 05>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 06>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 07>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 08>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 09>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 0a>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 0b>
<Buffer be be 01 01 01 01 01 01 01 01 01 01 01 01 be be be be be 0c>
```

Figure4. Examination of 0X13 data

首先對感測器進行驗證，除了少數的雜訊干擾外，結果符合預期，並未因先前使用出現狀況。接著，SPI 序列埠則是完成硬體上的設計和軟體上所設定的接腳位置，完成四線串接埠的傳輸，而 MCU 中的設定尚未完成，因此取樣率依舊無法提升。

最後是快閃記憶體提取方式的改良，順利完成資料傳輸和驗證與外接壓力計的部分，可直接在感測器上進行不同狀態的轉換，目前設定兩種狀態，能夠即時地傳輸資料，亦能在收、讀資料之間作轉換，即可不受距離過遠時藍芽中斷，而導致資料遺失

的情形。未來也將持續在感測器上進行改良，設法不透過壓力計的方式，利用對感測器的搖晃、拍打或是旋轉等動作，觸發感測器上的狀態進行轉換，即可設定讀寫資料的狀態，並且利用 SPI 的設計，完成取樣率的上升，約可達到目前的 4 倍，這些都有利於未來運動科技的發展，不僅方便選手使用，也能有所提升其訓練效果。

心得感想

這次的專題讓我學到了很多關於嵌入式的觀念和實作，從一開始的觀察測量開始，到後來的 SPI 接線、程式燒入和 flash memory 的操作等，充分了解到許多觀念，還有實際實行的過程。過程中雖然遭遇了許多挫折與阻礙，例如硬體設備上的問題、燒入程式碼遇到的問題、編譯過程出現狀況和讀取資料卡死的問題等，但經過不懈的努力和堅毅的精神，隨著時間的進展，最終還是順利完成，其中當然也要感謝學長和教授的指導，在艱困的過程中幫了我許多，也期望在未來有人能夠將這門專題繼續精進，將這門學問作更多的進展。