Embedded System Lab Final Project Report

B08202033 楊智凱、B08202054 李杰銘

一、 Github repository URL

https://github.com/jminglee/2022_fall_eslab/tree/main/Final%20Project

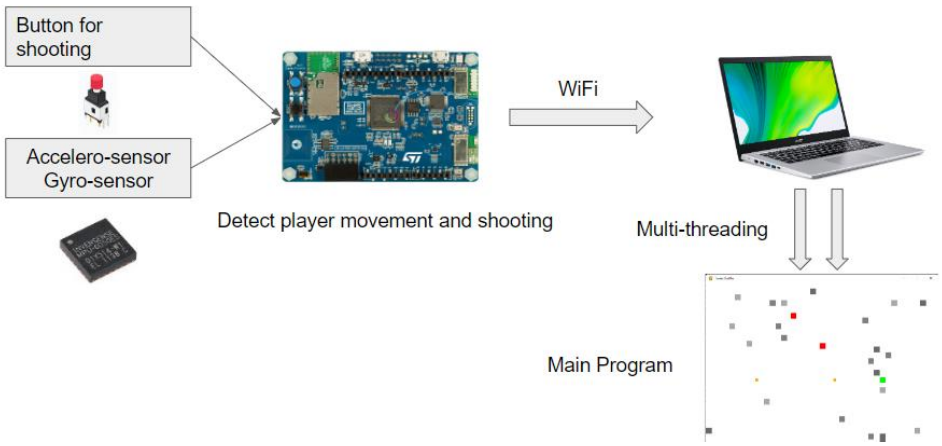
二、 題目與動機

一開始我們對主題相當茫然，也想不出可以用於實際生活中的應用。但是在看到助教整理的 github 後，發現有不少學長們都是選擇做遊戲。因此我們開始往遊戲方面思考後便從一些常見的遊戲之中選擇了一款相當懷舊的遊戲：坦克大戰。

另外，選擇這個主題也可以使用到許多課堂上提到的內容與技巧，如：STM32 on-board sensor、Wi-Fi socket、Multi-thread programming、Interrupt handling、Code optimization (ex: loop unrolling)、DSP 等等，且我們嘗試的成果（像是如何偵測上下移動轉動）經過一點點修改與類比之後也可以做為其他類似或甚至不同領域的主題實作的參考。因此我們認為這個主題是兼顧趣味以及增加對課堂內容的理解以及實際生活實用性的主題，非常適合選來實作。

三、 整體架構

我們以下圖說明這次 project 的整體架構：



首先，我們利用 STM32 上的 accelro-sensor 跟 gyro-sensor 測量加速度與角速度並利用 user button 偵測玩家是否射擊。於 STM32 的主程式中對這三種 input 分別進行適當的處理之後，經由 wifi 傳送所需要的資料到筆電，並在遊戲本體中利用這些資料做出反應。過程中每一步雖然看似簡單，但也有不少需要注意的地方。因此我們將目標放在將每一步個別都做

到最好，期待整合起來能夠帶來舒適的遊戲體驗。

四、 目標

我們整體的目標是製作出一款具有美觀圖形化介面，同時操作上簡單又流暢的坦克大戰遊戲。為了達到這個總目標，我們設定整體架構中每個部份的目標如下：

i. STM32：

由於 STM32 的部分和玩家操控的流暢度息息相關，因此我們認為這部分必須能夠將訊號做最適當與最快速的處理，讓我們可以在不失靈敏度的前提下正確地解讀玩家的操作。另一方面，如果我們的感測太過靈敏，在玩家握搖桿的時候有些許晃動都會讓遊戲做出反應的話反而對玩家是個困擾。因此如何處理這個問題也是重要的課題。

最後，因為如果 STM32 太過頻繁地傳輸資料，且每筆資料又過大的話，對於遊戲主程式而言會變成負擔，更可能轉化成玩家的延遲，降低遊戲體驗。因此我們希望能讓 STM32 傳輸資料的次數與大小的越小越好。

ii. Python 主程式：

作為遊戲的本體，我們希望他能呈現出美觀的介面。並且如同 STM32，處理資料的速度也要盡可能快一點才行。

五、 作法

在此我們會統整我們各個部分的最終作法，並且會在之後的環節討論每個做法和某些其他可能作法的比較以及實驗上的觀察和討論。

i. 解決玩家握搖桿時有輕微搖晃的狀況：

(1) Problem modeling

如上所說，如果我們的偵測靈敏到連玩家自然的輕微晃動都會做出反應，反倒變得難以操控。對此我們先對這個場景做點簡單的數學表示：令$Y(t)$是我們的 sensor 於時間t偵測到的資料；$X(t)$為玩家在手部不會自然搖晃的前提下，做到某個操作時 sensor 應接收到的資料；$h(t)$則是當下因為輕微搖晃造成的背景雜訊。則我們能得到：

$$Y(t) = X(t) + h(t)$$

$$E[Y(t)] = E[X(t)] + E[h(t)]$$

在未知$h(t)$的分布時，我們沒辦法做出 zero-mean 的假設，但是若假設在校正過程與實際遊玩時的雜訊來自於同一個分布的話，可以藉由校正過程中$h_{cal}(t)$的平均值作為上述期望值的估計(因為校正時$X_{cal}(t) = 0$)

(2) Method

承上所述，我們首先會先校正 sensor。校正的方法是我們會在遊戲開始前請玩家握著搖桿，這段期間我們會蒐集 3000 筆的 accelero-sensor 和 gyro-sensor 的資料，並計算這 3000 筆 data 的平均值作為 offset 儲存起來，之後會先將加速度以及角速度的資料扣掉此 offset 後再做後續處理。本文往後所提到的「角速度」和「加速度」都是意指處理 offset 後的結果。

ii. 玩家射擊之處理：

如整體架構所說，我們會以 STM32 上的 user button 作為讓玩家發射子彈的按鈕。我們以學期初教過的 Interrupt handling 處理這個，作法如下：首先宣告一個 volatile global variable 代表玩家是否射擊。宣告為 volatile 的用意是為了避免一些上課提到過的可能問題。接著我們利用 callback function 讓按鈕被按下的時候便將該變數設為 1 (True)代表玩家想要發射子彈；當按鈕放開時改回為 0 (False)。透過這樣，我們可以在後續取得玩家動作的時候利用這個變數得知玩家是否要射擊。

iii. 玩家左右移動的偵測與處理：

我們在此先描述左右移動的作法。處理方法如下：由於玩家在向左或向右移動的時候，必須將 STM32 板子向左翻或向右翻。而要感測這樣的動作的話，使用加速度是最為直覺的。因此我們決定設定判斷左右移動的標準如下：

$$Action = left \text{ if } (acceleration_x) \times \alpha > threshold$$
$$= right \text{ if } (acceleration_x) \times \alpha < -threshold$$

其中α是一個 scale factor，用來將蒐集到的 data (加速度的 x 分量)轉換到與預設的 threshold 接近的尺度上。適當的α和 threshold 皆以無數次的實驗中不斷調整與嘗試，最後使用遊戲體驗上最好的一組參數。

iv. 玩家上下移動的偵測與處理：

對於上下移動的偵測與處理，我們除了使用和左右移動(改為使用加速度的 z 分量)類似的方法以外，額外使用了角速度的資料作為額外的判斷基準。想法上，我們想要知道玩家是往上(內)翻還是往下(外)翻的話，可以透過翻動的角位移作為判斷基準。理論上角位移可以由角速度的積分取得，但因為我們接受的訊號並非連續的，因此改為黎曼和求解：

$$\Delta\theta(t) = \int_0^t \omega(t')dt' \approx \sum_{i=0}^{n} \omega(t_i)\Delta t_i$$

求出翻動的距離後，我們便可類比先前加速度的做法，設定一個 threshold 去判斷是往上還是往下。

v. 動作偵測與資料傳輸：

上方三點描述了我們如何偵測上下左右的移動以及發射子彈的指令之後，我們接著描述整體程式如何偵測動作與資料傳輸。

動作偵測的部分分為兩個 events。其中一個 event 是為了 update 加速度以及角速度的 data，以及即時計算當下翻轉距離的黎曼和，以供未來判斷動作使用。

另外一個 event 則是要判斷當下玩家採取的動作為何並且將結果透過 Wifi socket 傳輸過去。判斷的步驟基本上如同上述，主要特別的點在於傳輸資料的步驟。如前述，我們希望資料傳輸的次數可以同時兼顧正確性，又能減少傳輸的資料量以及傳輸的次數。因為我們會採取的動作只有 5 個，因此傳輸資料只需要 3 個 bit 即可正確傳達。利用這個想法，我們可以大幅減少傳輸的資料大小。另外，每次傳輸後會紀錄該次傳輸的資料為何，往後只需要在當下需要傳輸的動作指令和上一次不同的時候才傳，至於遊戲主程式的部分只需要一直重複上一次接收到的指令直到接收到不同的指令。透過這種方式我們便可以大幅減少需要傳輸的次數了。

最後，我們使用 event queue 進行 event scheduling。讓 update event 每 2ms 便進行一次更新，而每 10ms 便判斷指令以及進行傳輸(如果需要的話)。

vi. 遊戲主程式框架：

Tank battle 遊戲部分是以 Pygame 套件提供的 API 作為基本架構而寫成，此部分的目標為能夠呈現專題其他部分之運作結果，所以在遊戲設計並沒有使用較複雜的演算法。遊戲部分比較重要的地方是 FPS (frame per second，幀數) 的大小，我們將其數值訂在 30，目的是能夠以適當的頻率處理 STM32 傳送來的資料，並且讓玩家可以流暢地進行遊戲，如果數值設的太小，會在遊戲本身造成有影響的延遲，但如果數值設的太大，STM32 資料傳送的速度反而跟不上，會造成遊戲運行上的 error。

i. 遊戲與 STM32 連接：

我們原先是計畫以 Bluetooth 來進行資料傳輸，但後來考量到

資料傳輸的 latency 對於遊戲遊玩的重要性，才改用 Wi-Fi 進行資料傳輸。

實際運作方式如下：我們利用 Wi-Fi 進行資料傳輸，運行主要程式的筆電作為 server，STM32 作為 client 來主動傳出資料。而主程式的以 multi-threading 的方式來同時處理 server 資料的接收與處理，以及遊戲部分的運行。在這兩個 thread 之間是以 Event 的架構來進行互動，每種操作都有一個對應的 Event，server 接收到資料後會根據資料改變 Event 的設定，將 Event set()或是 clear()，而遊戲部分則是在每一幀都去檢查每個 Event 的狀態，若 Event 處於 is_set()的狀態，就會讓 player 進行對應的動作。

值得一提的是原先在主程式的部分，是將 server 與遊戲部分在同一個 thread 運行，在每一幀會去讀取 client 傳輸的資料，然後再讓 player 去作出相對應的動作。但後來我們發現這整個過程的處理時間較長，會使遊戲在運行上產生較多延遲，同時 STM32 傳輸資料的頻率還必須要對應到遊戲的幀數，不然可能會接收不到及時的資料。考慮到以上幾點之後，我們決定採用 multi-threading 的方式，將 server 資料的接收與處理，以及遊戲部分的運行分開，來降低資料接收的 latency。

另外，我們為了避免分工上的時間浪費。負責 game program 的組員會建立一個 simulating player (經過一些邏輯判斷來生成正常玩家會生成的操作序列)來測試 game program 是否能夠正常運作。

六、 成果

i. Demo 影片連結

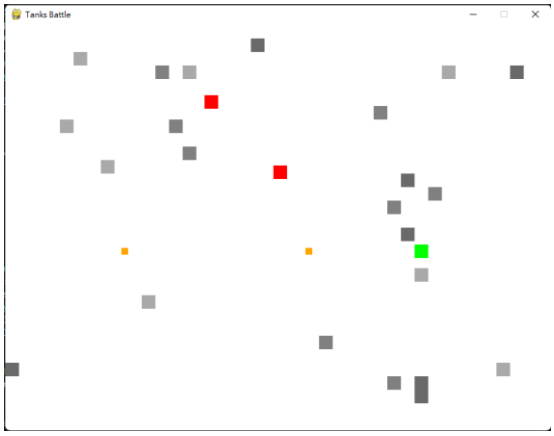
由於我們的成果較為動態，只靠幾張截圖也無法表達實際呈現的樣子為何，因此我們選擇在此放上 Demo 影片的連結以供參考：

https://www.youtube.com/watch?v=Tiydr7d9E0w

ii. 遊戲介面

1. 開始畫面：

2. 遊戲中畫面：



其中，綠色方塊為玩家操控的坦克，紅色方塊為敵方坦克，橘色方塊則是發射出來的子彈

iii. 整體遊戲性

根據我們自身的感覺，以及上述 Demo 影片的內容，我們認為遊玩起來還挺流暢的。整體來說雖然仍有一些問題尚待克服，但對於遊戲順暢度並不會有太大影響，因此我們認為已經算是達到我們原先的目標了。

七、 觀察與討論

在這個部分，我們想要討論實作上述作法時觀察到的現象，以及針對一些做法以及其他我們嘗試過的可能方法之選擇做討論：

i. Sensor 校正的替代方法

上述我們已經介紹過我們最終在 sensor 校正方面使用的方法，但其實我們也有做一些其他的嘗試，我們在此對嘗試內容、結果與不採用的原因做點說明如下。

我們曾經嘗試使用 DSP 的技巧來處理資料，想法上是將因為玩家手部輕微晃動造成的加速度與角速度當作 low-frequency background noise 處理。我們原先對輕微晃動的假設是將任何時刻的輕微晃動都當作同一種分布，且以某個時間段的平均來近似其期望值。對於一個未知的隨機變數而言，這樣的假設雖然很符合直覺，但也可能太過強烈。反之，新的方法只假設其為 low-frequency noise，相對來說比較合理，因此我們預期這個方法會有比原本方法更好的結果。

實作上，我們減少 STM32 傳送的頻率，而多出來的這段時間則是用來儲存足夠多的資料，接著我們對這些儲存起來的資料進行 DSP 的處理，試圖將低頻的背景噪音去除掉。最後我們將通過濾波器後的資料取平均作為代表，接下來就可以用先前定義的標準來判

斷動作了。至於 DSP 的部份我們則是仿照作業，利用 Mbed OS 的相關 library 實踐。

實驗結果上，我們發現這樣的方法綜合考量上並沒有比原本的方法好。更精確地說，在感測正確度方面，新的方法確實有進步，然而在靈敏度方面，卻是大大地不如原先的方法。我們分析理由如下：原先的方法好處在於需要取得當下動作狀態時，我們只需要讀取當下的 data 瞬時值並做一些簡單的計算和比較即可。然而使用濾波器的方法，在判斷動作之前，卻需要花一些時間蒐集資料並進行 DSP 運算，不論哪一步都是費時的，且沒有甚麼技巧能將這些時間省下來。當然，減少儲存並進行 filtering 的資料數量是可以減低運算時間的(因為蒐集的時間變少，且做 filtering 的時間也會隨著資料點變少而減少)，然而經過我們實驗後發現，減少需蒐集的資料點數量卻導致判斷的準確度也跟著變低了。考量到這次的 project 仍是以遊戲作為主題，靈敏度與準確度理當是著重考量的重點，因此我們捨棄了這個做法。

ii. 移動偵測之加速度 threshold 值

先前提到，我們會給四個方向的移動設定 threshold，並以此作為標準判斷是否觸發以及移動方向。在此我們必須說明四個方向的 threshold 未必是相同的，而是可以根據實驗觀察調整的。

舉例而言，我們發現如果要讓坦克往上轉向的同時射擊很容易造成誤判，其中誤判結果絕大多數都是會將坦克向下轉向。我們認為造成這樣的誤差並不是偶然，而是因為當我們往上轉的同時按下 STM32 的 user button 的話，會自然地對板子施加一個和移動方向相反的作用力。而正是因為這個作用力的加速度導致我們的演算法判定成往下移動，導致上述的錯誤。要解決這個情境造成的錯誤，改變判定往下移動的 threshold 是最為直覺的想法。直觀來看，如果判定向下移動的 threshold 較大，那麼按下按鈕造成的微微加速度和轉動都將不會觸發判定，因此可以產生正確的結果。

以上例子只是用來說明各個方向的 threshold 未必要相等，兩個相反方向(如左和右、上和下)的 threshold 也未必要對稱，完全可以基於使用者方便做點調整與更動。

iii. 解決 Recoiling 問題的可能方法

我們所說的 recoiling 問題是指當玩家向希望的方向轉動板子後並且成功觸發判定後，因為手部有意或無意的回位而導致程式額外判定一個往原先方向之反方向的 operation，並進而導致操作上不順利的問題。我們最終是以和前一點一樣的概念，透過調整

threshold 來減緩這個問題的發生率。但我們也嘗試過其他可能的方法，並在此提出以供參考。

第一個方法中，我們會使用一個 counter 來記錄當前 operation 連續發生的次數(其中 counter 會在計算 action 時同時更新)。而此 counter 的用途是為了評估當程式突然產生和前一個 operation 相反方向的 operation 時，其為 recoiling 的可能性有多高。這個方法的基本概念如下：一般來說一次有效操作會產生 10 到數十個 operation，且這些 operation 大多都會是同一個(例如都是往右)，但如果在產生 3 個 up (往上)之後突然插進一個 down (往下)，那很有可能是因為 recoiling 造成的誤判；反之如果是在 20 個 up 之後才有一個 down 的話，則有可能是玩家真的選擇採取 down 這個 action。因此我們才利用 counter 來輔助判定。然而我們設定 counter 大於 15 之後出現的反方向操作是有效的之後進行實驗，卻發現遊玩起來並沒有比較順手。這個原因是因為有時有可能是想要進行「微操作」，在往上移動一小段距離之後就馬上轉向下。然而前述的 counter constraint 卻會將這種操作視為誤判，因而不會產生反應。這也反映了這個方法的致命缺點：無法建立對任何情境都適用的標準。因此最後我們並未採用此法。

第二個方法是試著給每個操作(除了射擊外)一個分數，並以此分數來判定其是不是 recoiling。賦予分數的方法有很多種，舉例來說，我們可以把分數設定成

$$(current\ operation\ acc - current\ threshold) - (last\ operation\ acc - last\ threshold)$$

透過這樣的分數設定(acc 為加速度)，我們可以知道當前 operation 時板子運動的幅度和上一個 operation 時板子運動幅度相差多少。實作上我們可以給這個分數也設定一個 threshold。假設分數大於該 threshold，代表兩次 operation 時板子運動幅度相差甚大，那就很有可能是由玩家自主操控的結果而不是 recoiling。然而這個方法的困難點也和前一個方法一樣，是我們無法給出一個放諸四海皆準的 scoring policy。以前面提到的評分方式為例，可以看出其給的分數會和加速度的 scale 有關。也就是說同一套評分方式可能對於平常轉動板子幅度就比較大、比較用力的使用者有用，但是對於平常轉動幅度比較小的使用者就沒那麼有幫助了。也就是因為這一點我們最後才沒有採取這個做法。

最後一個做法則是利用一些比較先進的演算法給數據做 labelling。譬如我們可以取連續 30 個時間點的資料(其中包含一個 operation 轉換的時間點)，將這 30 個時間點的資料合併成一個 feature 讓像 KNN 或是 machine learning 的技術自動幫我們給 feature

做標記。利用一些人為標註的資料作為訓練集，期待演算法可以學出甚麼是合法的 operation 而不是 recoiling。不過這部分由於我們時間不足，因此沒有太深入探究，但是我們相信這個方法或許能夠奏效，因此在此提出供未來研究類似主題的學弟妹們參考。

iv. 使用 z 軸加速度而非 x、y 軸加速度判斷移動

最後我們提出這個主題是因為助教先前在進度報告時有和我們提到以往做類似主題的作法都是用 z 軸加速度分量的變化量作為判斷移動方向的標準。然而在我們實作之後卻並沒有觀察到明顯的改善。

對此，我們認為：首先，我們的移動方向有四種，理論上需要四個自由度才能夠完整描述。然而由於四種移動方向實際上只在兩個維度移動(遊戲畫面上來說是鉛直、水平；板子轉動方向來說是左右翻、內外翻)，因此我們可以將需要的自由度壓縮成兩個。但也因為如此，我們不可能僅用一個自由度(z 軸加速度)就完整區分出四種移動方向，勢必得要額外的資訊(其他分量或是角速度)才可以達成。這也是我們認為不可能僅靠 z 軸加速度就完成這項任務的原因，也是我們一開始導入角位移或是 x 軸加速度等作為判斷基準的原因。而在實驗中也確實看到這些輔助條件是有幫助的。

八、 結論

我們在這次 project 中，成功利用 STM32 的 onboard sensor 以及 Wi-Fi socket 和 pygame 套件實作出一款坦克大戰遊戲。其中，我們利用一套結合加速度資訊以及角速度資訊的資料處理的方法完成 sensor 的校正，以及以 STM32 做為搖桿的操作辨識，並且達到很好的靈敏度與準確度。雖然我們的主題可能較為樸素，但是相對的我們也將目光放在更微小但重要的細節，並將其做得更好。最後也成功達成當初所設下的目標。

九、 使用到的課堂教授技巧總結

以下列出我們在這次 project 中，使用到的課堂技巧：volatile variable、event queue、on-board sensor、interrupt handling、wifi socket programming、code optimization、multi-thread programming、DSP 等。

十、 心得

1. 楊智凱：

我這次主要負責 STM32 板子的處理。坦白講在過程中學到挺多的，從原本的只求東西能跑就好，到慢慢地可以自己設計新的想法並且實作出來，這整個過程中的思考是我過去修實驗課很少經歷的。以前大多都是照著

助教簡報的步驟做一做，報告寫一寫就好了。但是這次 final project 卻要自己實作出一個產品，相當新奇。而且由於要自己解決問題，因此也需要額外上網查許多資料，不論是套件的使用還是一些其他先進已經研究過的做法等等，不僅加強我資料蒐集的能力，也讓我慢慢培養出自己的思考能力。另外，這次 project 也讓我知道，即使一件事情看似很簡單，但背後的細節卻可能不那麼簡單了；或是一件事情剛接觸的時候很簡單，但是要把簡單的事情做好，甚至做到更好、最好卻是困難的。因此我學到了不要憑自己的第一印象去評估某一件事情的難度，因為還沒做之前一切都是未知，太過大意或小看一個表面上不難的任務是有可能吃盡苦頭的。總而言之，我認為這次 final project 讓我收益良多，也很感謝助教們每次實驗課的幫忙！

2. 李杰銘：
   這次project我們實做了不少東西，也有很多東西是沒人教過的要自己查。不知不覺自己都慢慢學會那些以前自學學不會的東西了。像是這次使用的 pygame 我也是照著網路上的教學先一步一步做出個範例，後來再慢慢舉一反三弄出現在的遊戲主程式。雖然過程中遇到不少麻煩，做出來的畫面也沒有到很好看，但我已經對自己和這份作品很滿意了。

十一、 參考資料

1. https://os.mbed.com/handbook/DSP
2. https://www.pygame.org/news
3. https://github.com/NTUEE-ESLab/2021-pikachu_volleyball
4. https://os.mbed.com/docs/mbed-os/v6.15/apis/wi-fi.html
5. https://github.com/NTUEE-ESLab/2021Spring-Fruit-Ninja