東海大學資訊工程學系

專題報告書

搭載動態物體追蹤之發射裝置與頭戴裝置聯動

Helmet-Linking Launcher with Target-Moving Tracer

指導老師：楊朝棟

S07353042 施品言

S07353028 陳昱瑋

中華民國 110 年 12 月 25 日

致謝

首先要感謝我們的指導老師，在我們提出此專題想法時，雖然此專題所運用到的知識與目前熱門之人工智慧、Edge AI、AIoT 較無關聯，但指導老師仍盡可能協助支援我們器材以及測試方向上的建議等等，也必須提到陳世澤助教，在我們尚未開始專題時，就時常提供我們關於專題相關的想法，並且在長達一年製作此專題時，也時常協助提供我們在測試方向上的建議。

摘要

在現今影像辨識以及物聯網技術逐漸發展的同時，已經有許多企業將自家產品融入部分影像辨識或是物聯網技術，而在看過阿帕契[1]頭盔相關文章後，發覺阿帕契[1]頭盔相關技術不應該只用於阿帕契[1]直升機上，可以利用影像辨識以及物聯網技術簡易還原出阿帕契[1]頭盔的部分功能，再利用那些功能融合發射裝置，以利於部分軍事載具使用。

因無法實際取得相關發射裝置，故在實作時是以模擬出的軍事載具上測試，我們使用了物聯網技術將搭載感測器的頭盔與模擬出的發射裝置聯動，當鎖定目標後，發射裝置會將該目標當作物體追蹤之目標進行追蹤並且旋轉自身方向。

關鍵詞：物聯網、物體追蹤、阿帕契[1]頭盔、Re^3

目錄

圖目錄

表目錄

第一章 緒論

1.1 研究背景與動機

近幾年影像辨識以及物聯網技術不斷精進，再加上體感互動的遊戲、商品也愈來愈多，想嘗試是否能夠利用我們所知道的物聯網技術製作出一套擁有體感互動的裝置，在構想時想到前些年的阿帕契[1]頭盔新聞，因此我們想嘗試模擬阿帕契[1]頭盔功能，操控發射裝置之人員只需透過擺動頭部就能同步旋轉發射裝置，再搭配上能夠追蹤物體位置之發射裝置，希望有機會能夠利用我們所學去改進目前的軍事載具。

1.2 研究目的

在目前的軍事載具上大部分皆為由載具中的人員去手動操控各個部件，以觀看附近狀況、鎖定目標、發射砲彈等，但在鎖定目標時極度仰賴該人員之肉眼辨識能力，若能夠將發射裝置搭載物體追蹤技術及頭盔聯動旋轉發射裝置，如此就能降低人員失誤機會，甚至有機會實現遠程控制載具進攻敵軍，以減少人員傷亡。

1.3 研究流程與架構

本系統因為無法位於實際載具上實作，所以在此次系統開發上皆以模擬的方式進行實作，本系統分為兩部分：第一部分為模擬的砲台旋轉與頭盔聯動、第二部分為發射裝置追蹤物體。

第一部分，由頭盔上的 MPU9250 九軸姿態感測器偵測出使用者頭部往左或右旋轉多少角度，再由 ESP32-S 透過呼叫 API 的方式將數據傳送至模擬載具上搭載的 Nvidia Xavier NX，如此就能夠在 NX 上進行簡單運算後，由 NX 傳送資料給伺服馬達，實現頭盔與砲台旋轉聯動。

第二部分，透過使用 Re^3 來實作物體追蹤，若物體移動或是模擬載具移動時，物體在影像中的 x 軸、y 軸數據會產生變化，再將此變化數據進行簡單運算後，傳送資料給發射裝置上的伺服馬達，以實現搭載物體追蹤之發射裝置。

第二章 相關文獻回顧

2.1 ESP32-S[2]

ESP32-S 是一款通用型 WiFi-BT-BLE MCU 模組，功能相當強大，用途也相當廣泛，此款模組的核心為 ESP32 晶片，具有可擴展性、自適應的特點，兩個 CPU 核心也可以被單獨控制或通電，CLOCK 頻率為 80 MHz 到 240 MHz，使用者可以切斷 CPU 的電源，利用低功耗處理器不斷監測個介面的狀態變化或者某些數值是否超出臨界值，ESP32 晶片的睡眠電流更是小於 5uA，使其適用於電池供電的可穿戴電子設備。

ESP-WROOM-32 集成了傳統藍芽、低功耗藍芽和 Wi-Fi，具有廣泛的用途，Wi-Fi 支援極大範圍的通訊連接，也支援路由器直接連接乙太網路；而藍芽可以讓用戶連接手機或者廣播 BLE Beacon 以便於訊號檢測。

2.2 步進馬達[3]

步進馬達是無刷直流馬達的一種，為具有如齒輪狀突起相鍥合的定子和轉子，可藉由切換流向定子線圈中的電流，以一定角度逐步轉動，且只需要通過脈波信號的操作，即可簡單實現高精度的定位，並使工作物在目標位置高精度的停止。

2.3 Nvidia Xavier NX[4]

Nvidia Xavier NX 為小尺寸模組系統，體積僅 70mm x 45mm，並擁有卓越的性能和能源優勢，以及一系列豐富的 IO，從高速 CSI 和 PCIe 到低速 I2C 和 GPIO，不僅如此，Nvidia Xavier NX 還具備 384 個 Nvidia CUDA 核心、48 個 Tensor 核心、6 個 Carmel ARM CPU 與兩個 Nviida 深度學習加速器引擎的效能，結合超過每秒 59.7 GB 的記憶體頻寬、影片編碼與解碼功能。

2.4 Re^3[5]

Re^3 為即時深度物體追蹤器，是一份開源於 Github 上的專案，此專案執行需求為 Python 2.7+或 Python 3.5+、Tensorflow、Numpy、OpenCV 2 與 CUDA，而 cuDNN 則是看使用需求。

此即時深度物體追蹤器會不斷將時間資訊傳入模型裡，而不是專注於一組特定物件或訓練，且 Re^3 會同時追蹤物件與更新外觀模型，還能夠以 150 FPS 的速度追蹤物件。

2.5 ToF[6]

飛行時間 (ToF) 是對物體、粒子或波在介質中傳播一段距離所花費的時間的測量，可以用此訊息來建立時間標準，最為測量速度或路徑長度的一種方始，或作為了解粒子或介質屬性的一種方式。可以直接或間接地檢測行進物體。

2.6 Flask[7]

Flask 是一個使用 Python 撰寫的輕量級 Web 應用程式框架，由於其清量特性，也稱為 micro-framework。Flask 核心主要是由 Werkzeug WSGI 工具箱和 Jinja2 模板引擎組成，Flask 和 Django 不同的地方在於 Flask 給予開發者非常大的彈性，可以選用不同的 extension 來增加其功能。

2.7 伺服馬達[8]

伺服馬達是以回饋訊號控制，採用閉迴路系統，將感測器裝在馬達與控制對象機器上。

伺服馬達分為直流和交流兩種，而直流馬達因為操作容易，也就是旋轉方向由電流決定，並且旋轉速度由改變施加的電壓來控制，控制簡單所以廣泛使用，因此現在直流伺服馬達是使用最多的馬達。

第三章 專題研究與開發

3.1 系統架構

整個系統主要分成兩個部分，第一部分為頭盔上利用 MPU9250 姿態感測器偵測使用者頭部擺動的資料，由 ESP32-S 透過 WebSocket 的方式呼叫 API，將擺動資料傳送至載具上的 Nvidia Xavier NX，再傳送訊號給模擬砲塔的伺服馬達以及旋轉平台之伺服馬達，以達到頭盔與砲塔聯動，第二部分為載具上的 Nvidia Xavier NX 作業系統為 Jetson NX 官方之 Linux，利用 Flask 建立 API Server，透過呼叫 API 以實現使用者能夠觀看到模擬載具的視角與砲塔的視角，當按下按鍵鎖定目標後，呼叫 API 以執行 Re^3 物體追蹤，再透過 X 軸、Y 軸的數據變化轉動伺服馬達，以實現搭載物體追蹤之發射裝置，如圖 3.1。

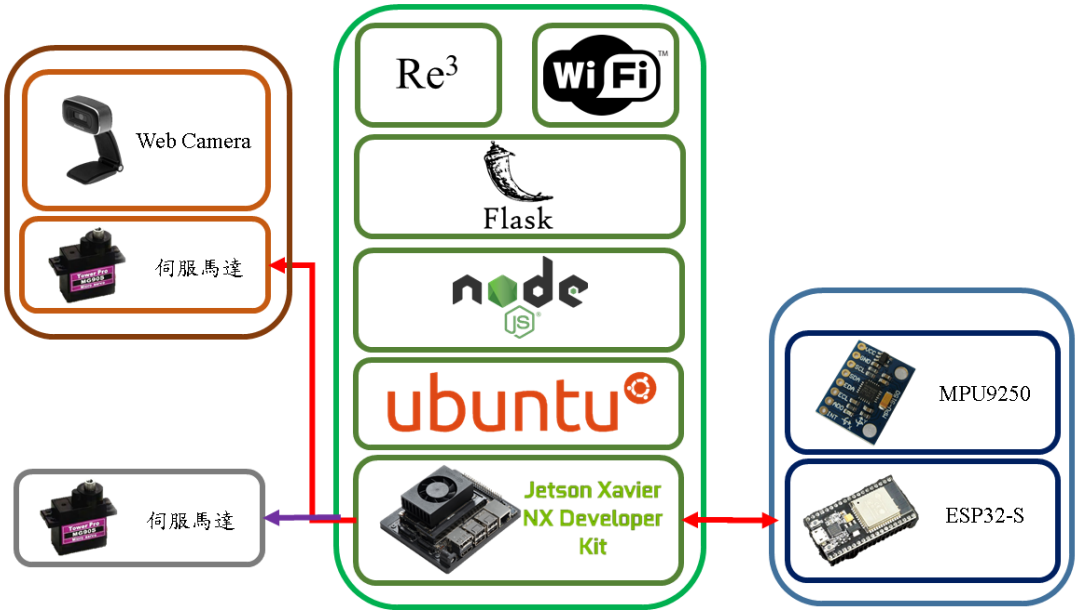

圖 3.1 系統架構圖

3.2 研究方法

3.2.1 OpenCV[9]影像辨識實作

1. cv2.absdiff[9]

該函式字面的意思是「差異的絕對值」，運算方式是將兩張照片相減後取絕對值，實作上就是抓取影像前一幀與後一幀的圖片，並用來得到其中的差異，物體移動時，由於移動是漸進的，因此差異會發生在「物體邊緣」，這使得移動中的物體看起來像是由「線條」組成的輪廓。此方法較常用於固定視角時，辨識物體所用，結果如圖 3.2。

圖 3.2 absdiff 函式結果

2. cv2.createBackgroundSubtractorKNN[9]

該函式是使用前背景分離技術，是以 K 近鄰 (K-nearest neigbours) 為基礎的前景/背景分離技術，來自於 Zoran Zivkovic 與 Ferdinand van der Heijden 的論文「Efficient adaptive density estimation per image pixel for the task of background subtraction」。

主要是利用 KNN 來實現影像的背景分割演算法，並且在後續每一幀的影像做微調，實作之程式碼如圖 3.3。

由於在進行辨識時，此方法會將一些微小的動作也一併標記，所以我們利用輪廓點的大小來判斷是否要將有移動的物體標記出，這種方法可以避免會有

太多標記框同時出現，導致畫面變得非常混亂，但也會因此犧牲掉一些小片段(如：一根手指)。

```
from cv2 import cv2
import numpy as np
import sys
camera = cv2.VideoCapture(0)
bs = cv2.createBackgroundSubtractorKNN(detectShadows=True) #建立前背景分離的演算法
es = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 2)) #用於形態學圖像處理
while True:
    ok, frame_lwpCV = camera.read()
    fgmask = bs.apply(frame_lwpCV)
    # 背景分割器，该函数计算了前景掩碼
    th = cv2.threshold(fgmask.copy(), 244, 255, cv2.THRESH_BINARY)[1]
    # 二值化處理，前景掩碼含有前景的白色值以及陰影的灰色值，在此圖像中，將非纯白色（244~255）的所有像素都設為0，而不是255
    dilated = cv2.dilate(th, es, iterations=2)
    # 形态学膨胀
    contours, hierarchy = cv2.findContours(dilated, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # 該函數計算圖像中，所有目標的輪廓點
    for c in contours:
            #利用輪廓點的數量判斷是否做框
        if cv2.contourArea(c) > 6000:
            r = cv2.boundingRect(c)
            (x, y, w, h) = cv2.boundingRect(c)
            cv.rectangle(frame_lwpCV, (x, y), (x + w, y + h), (255, 255, 0), 2)
    cv2.imshow('detection', frame_lwpCV)

    cv2.waitKey(1)
camera.release()
cv2.destroyAllWindows()
```

圖 3.3 createBackgroundSubtractorKNN 測試之程式碼

3. cv2.TrackerCSRT_create()[9]

該函式是使用 CSR-DCF (Discriminative Correlation Filter with Channel and Spatial Reliability) 追蹤演算法來進行物體的追蹤，與前兩種方法不同，前兩種偏向於物體偵測，而此種方法使用的是物體的追蹤，通常物體追蹤的演算法會比物體偵測的演算法還快，原因是，由於追蹤物體時，是追蹤一個範圍內的物體，並且在追蹤前一幀的影像時，對於要追蹤的物體的外觀可以有相當的了解，所以在下一幀中，可以使用前面所獲得的訊息來預測下一幀中物體的位置，並圍繞物體的預期位置進行小範圍搜尋，而物體偵測的演算法則是從頭開始，實作之程式碼如圖 3.4 。

雖然物體追蹤速度很快，但是在追蹤過程中出現阻礙 (如：被遮擋) 時，很有可能導致物體追蹤的誤判 (如：追蹤的物體被迫改變) 或是失敗。而此時可能就需要重新進行追蹤。

```
class VideoCamera(object):
    def __init__(self, flip = False):
        self.vc = cv2.VideoCapture(0)
        self.x = 0
        self.y = 0
        self.time_count = 0
        self.track_sign = 0
        self.tracker = cv2.TrackerCSRT_create()

    def Rectangle(self):
        rval,frame = self.vc.read()
        if self.track_sign==1:
            if self.time_count=="1":
                bbox = (self.x-50,self.y-50,100,100)
                ok = self.tracker.init(frame, bbox)
                self.time_count="0"
            ok, bbox = self.tracker.update(frame)
            cbox = (int(bbox[0]-100),int(bbox[1]-100),300,300)
            if ok:
                p1 = (int(bbox[0]), int(bbox[1]))
                p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
                cv2.rectangle(frame,p1,p2,(255,0,0),2)
                ret, jpeg = cv2.imencode('.jpg', frame)
        else:
            cv2.circle(frame,(self.x,self.y),1,(0,0,255),4)
        rect,jpeg = cv2.imencode('.jpg',frame)
        return jpeg.tobytes()
```

圖 3.4 使用 TrackerCSRT_create 之程式碼

3.2.2 Re^3 物體追蹤實作

Re^3 全名為 Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects[5]，追蹤方法是依照我們對於鏡頭裡部份被選擇的區塊，讓該區塊影像經由常短期記憶模型（LSTM）[10]的訓練後再輸出，如圖 3.7。

LSTM 是一種時間循環神經網路，與 RNN 不同的地方在於，單純的 RNN 是無法很有效的處理影像與時間的關係，因此才會使用 LSTM，而 Re^3 為了提高其精準度，它判斷的不只有影像前一幀與後一幀的關係，並利用 LSTM 來回顧前幾個訓練的結果，進而做出更加合理的預測，如圖 3.6，因為 Re^3 是利用了機器學習的模組去作訓練，因此它也擁有了其他先進的追蹤器的準確性和穩健性，如 OpenCV[9]底下的 CSRT_Tracker[9]。

```
def show_webcam(self):
    while True:
        self.StartFrameTime = time.time()
        ret_val, img = self.cam.read()
        if self.track_sign==1:
            if self.time_count=="1":
                self.boxToDraw = (self.x-50,self.y-50,self.x+50,self.y+50)
                self.outputBoxToDraw = self.tracker.track('webcam', img[:,:,::-1], self.boxToDraw)
                self.time_count = "0"
            else:
                self.outputBoxToDraw = self.tracker.track('webcam', img[:,:,::-1])
            cv2.rectangle(img,
                    (int(self.outputBoxToDraw[0]), int(self.outputBoxToDraw[1])),
                    (int(self.outputBoxToDraw[2]), int(self.outputBoxToDraw[3])),
                    [0,0,255], PADDING)
        else:
            cv2.circle(img,(self.x,self.y),1,(0,0,255),4)
        fps = 1/(self.StartFrameTime - self.PrevFrameTime)
        self.PrevFrameTime = self.StartFrameTime
        fps = int(fps)
        fps = str(fps)
        cv2.putText(img, fps, (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 1, cv2.LINE_AA)
        rect,jpeg = cv2.imencode('.jpg',img)
        return jpeg.tobytes()
```

圖 3.5 呼叫 Re^3 Tracker 之程式碼

```
class Re3Tracker(object):
    def __init__(self, gpu_id=GPU_ID):
        os.environ['CUDA_VISIBLE_DEVICES'] = str(gpu_id)
        basedir = os.path.dirname(__file__)
        tf.Graph().as_default()
        self.imagePlaceholder = tf.placeholder(tf.uint8, shape=[None, CROP_SIZE, CROP_SIZE, 3])
        self.prevLstmState = tuple([tf.placeholder(tf.float32, shape=(None, LSTM_SIZE)) for _ in range(4)])
        self.batch_size = tf.placeholder(tf.int32, shape=())
        self.outputs, self.state1, self.state2 = network.inference(
                self.imagePlaceholder, num_unrolls=1, batch_size=self.batch_size, train=False,
                prevLstmState=self.prevLstmState)
        self.sess = tf_util.Session()
        self.sess.run(tf.global_variables_initializer())
        ckpt = tf.train.get_checkpoint_state(os.path.join(basedir, '..', LOG_DIR, 'checkpoints'))
        if ckpt is None:
            raise IOError(
                    ('Checkpoint model could not be found. '
                    'Did you download the pretrained weights? '
                    'Download them here: http://bit.ly/2L5deYF and read the Model section of the Readme.'))
        tf_util.restore(self.sess, ckpt.model_checkpoint_path)

        self.tracked_data = {}

        self.time = 0
        self.total_forward_count = -1
```

圖 3.6 Re^3 內 Tracker 之程式碼

```python
import tensorflow as tf

class CaffeLSTMCell(tf.contrib.rnn.RNNCell):

  def __init__(self, num_units,
               initializer=None,
               activation=tf.nn.tanh):
    """Initialize the parameters for an LSTM cell.
    Args:
      num_units: int, The number of units in the LSTM cell
      initializer: (optional) The initializer to use for the weight and
        projection matrices.
      activation: Activation function of the inner states.
    """

    self._num_units = num_units
    self._initializer = initializer
    self._activation = activation

    self._state_size = tf.contrib.rnn.LSTMStateTuple(num_units, num_units)
    self._output_size = num_units
```

圖 3.7 Re[3] 訓練模型之程式碼

這次實做 Re[3] 是使用 Python 底下的 TensorFlow-GPU，版本為 1.15，會使用 TensorFlow 的原因主要是由於 CUDA[11]是支援機器學習相關函式，因而讓影像處理可以使用 GPU 進行計算，也使 NX 的 CPU 減輕負擔，執行時效能，如圖 3.8。

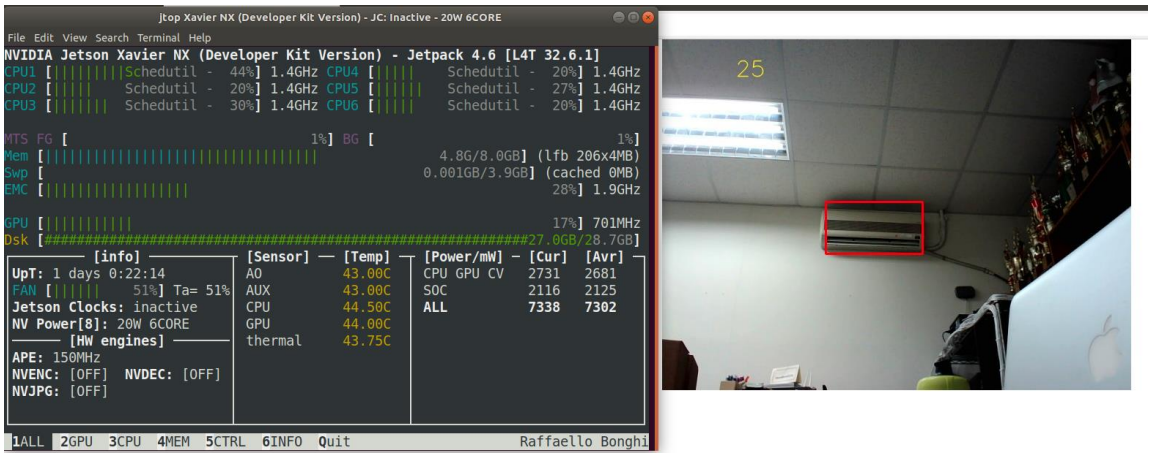

圖 3.8 Re[3] 實際執行之效能表現

是否開始進行追蹤是利用 JavaScript 與 Python 的溝通做判斷，利用 AJAX 傳輸資料至 Python 裡，而在網頁開啟時會先傳送一次資料至 Pyhton 當中，讓影

像進入等待追蹤的狀態，等待追蹤時中心會以紅點顯示，如圖 3.9，開始追蹤則是以紅框來代表開始追蹤框內物件。

```
$(document).ready(function () {
    getCoordinates(0);
});
function getCoordinates(track_sign) {
    sign = track_sign;
    var data = {
        data: JSON.stringify({
            time_count: "1",
            track_sign: sign
        }),
    }
    $.ajax({
        url: "image_roi",
        type: "post",
        data: data,
        dataType: 'html',
        success: function (data) {
            console.log("success");
        },
        error: function (xhr, ajaxOptions, thrownError) {
            console.log(thrownError);
            alert(thrownError);
            alert("data send fail")
        }
    })
}
```

圖 3.9 物體追蹤尚未執行時情況與 FPS

而因為 Re^3 對於物件追蹤的精準度需求比 TrackerCSRT 來的高，為了配合物件的大小，我們使初始追蹤框可以改變大小，程式碼如圖 3.12 與圖 3.13，呈現效果如圖 3.10 和圖 3.11，現在未追蹤時我們將以虛線框表示，並可利用鍵盤按鍵改變虛線框大小。

圖 3.10 未改變追蹤框大小

圖 3.11 增大虛線框時

```
function BoxAdjust(adjust_sign){
    sign = adjust_sign;
    var data = {
        data: JSON.stringify({
            sign : sign
        }),
    }
    $.ajax({
        url: "tracking_size",
        type: "post",
        data: data,
        dataType: 'html',
        success: function(){
            console.log("send success")
        },
        error: function (thrownError){
            console.log(thrownError);
            alert(thrownError);
        }
    })
}
```

圖 3.12 按鍵傳送調整追蹤框大小之 JavaScript 程式碼

```python
def BoxAdjust(self):
    if self.ChangeStatus == 1:
        self.size -= 2
        self.StartCoordinate = (self.x-self.size,self.y-self.size)
        self.EndCoordinate = (self.x+self.size,self.y+self.size)
        self.ChangeStatus = -1
    elif self.ChangeStatus == 0:
        self.size += 2
        self.StartCoordinate = (self.x-self.size,self.y-self.size)
        self.EndCoordinate = (self.x+self.size,self.y+self.size)
        self.ChangeStatus = -1
```

圖 3.13 Flask 端調整繪製追蹤框之 Python 程式碼

1 物體追蹤 FPS 比較

使用 Re[3] 之前，我們是使用 OpenCV 中的物件 TrackerCSRT，進行影像追蹤的處理，但由於 OpenCV 底下的 Tracker 物件並沒有支援使用 GPU 進行影像處理，因此我們先使用多執行緒的方法進行分工的作業，只可惜效果並沒有變好，程式碼如圖 3.14，執行效能如圖 3.15。

```python
def queryframe(self):
    while self.started:
        status , Frame = self.vc.read()
        return Frame
        time.sleep(0.1)
def start(self):
    frame_thread = threading.Thread(target=self.queryframe, args=())
    self.started = True
    tracker_thread = threading.Thread(target=self.tracking,args=())
    frame_thread.start()
    time.sleep(1)
    tracker_thread.start()
def tracking(self):
    while True:
        self.StartFrameTime = time.time()
        frame = self.queryframe()
        if self.track_sign==1:
            if self.time_count=="1":
                bbox = (self.x-50,self.y-50,100,100)
                ok = self.tracker.init(frame, bbox)
                self.time_count="0"
            ok, bbox = self.tracker.update(frame)
            cbox = (int(bbox[0]-100),int(bbox[1]-100),300,300)
            if ok:
                p1 = (int(bbox[0]), int(bbox[1]))
                p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
                cv2.rectangle(frame,p1,p2,(255,0,0),2)
        else:
            cv2.circle(frame,(self.x,self.y),1,(0,0,255),4)
        fps = 1/(self.StartFrameTime - self.PrevFrameTime)
        self.PrevFrameTime = self.StartFrameTime
        fps = int(fps)
        fps = str(fps)
        cv2.putText(frame, fps, (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 1, cv2.LINE_AA)
        rect,jpeg = cv2.imencode('.jpg',frame)
        return jpeg.tobytes()
```

圖 3.14 OpenCV Tracker 使用多執行緒之程式碼

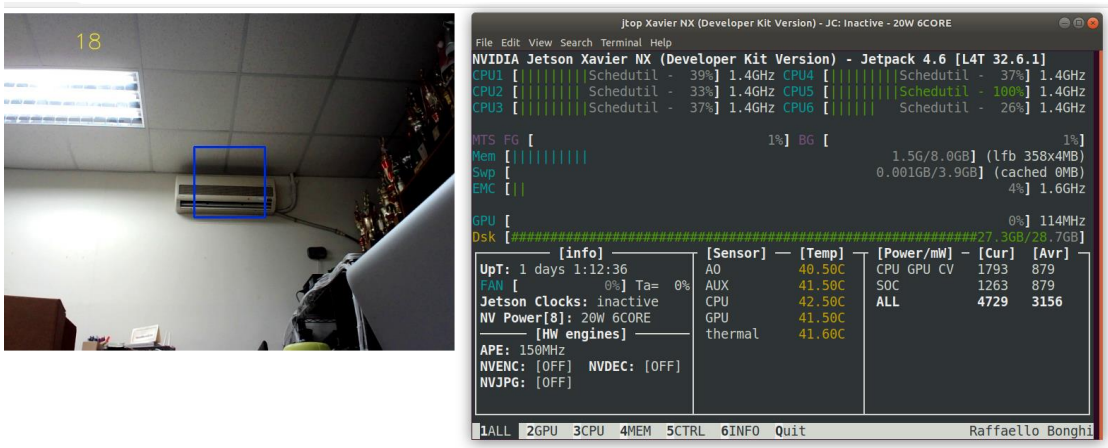

圖 3.15 OpenCV Tracker 使用多執行緒之效能

由圖 3.16 可觀察到，單個執行緒會執行至 100%，而整體的 CPU 則會跑到 175% 甚至更高，而幀數也從原本的 30 降到 15 以下，因此追蹤的效果有達到，但是效率並沒有達到理想值，且追蹤的效果也會因為幀數的降低導致追蹤速度較快的物體更容易失敗，而使用多執行緒並沒有出現改善的原因很有可能是因為 Python 的 GIL(Global Interpreter Lock) 限制，造成 Python 程式無法以多執行緒發揮多核心 CPU 的效能。而 OpenCV 的 Tracker 物件並不會因為追蹤物體的大小進而去改變框的形狀與大小。

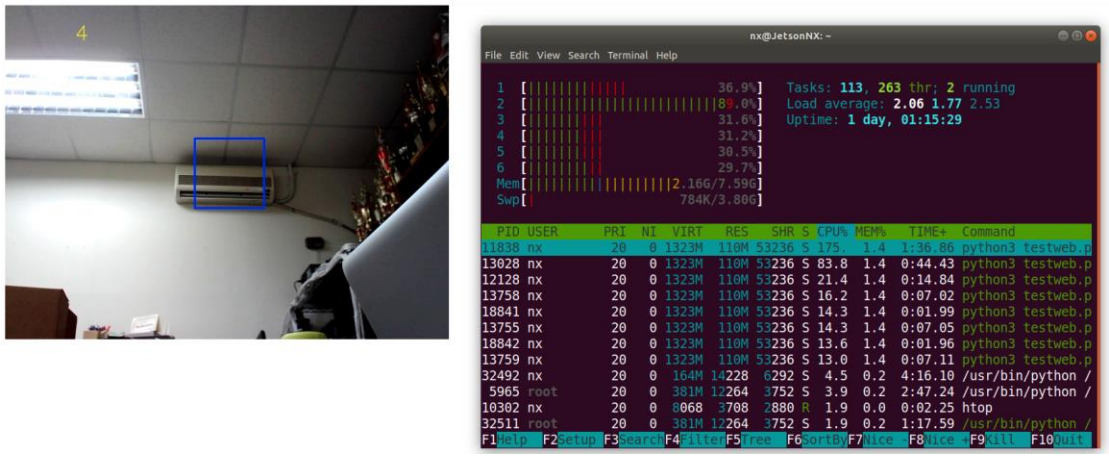

圖 3.16 OpenCV 物體追蹤效能

而使用 Re^3 搭配 TensorFlow-GPU 不只能夠利用 GPU 進行影像處理，讓 CPU 的負擔變輕，更能夠隨著追蹤物體的遠近、大小去改變追蹤框的大小與形狀，影像的幀數也能夠維持在 28 左右，與原本沒有追蹤時的幀數 30 不會差距過大，如圖 3.17 與圖 3.18。

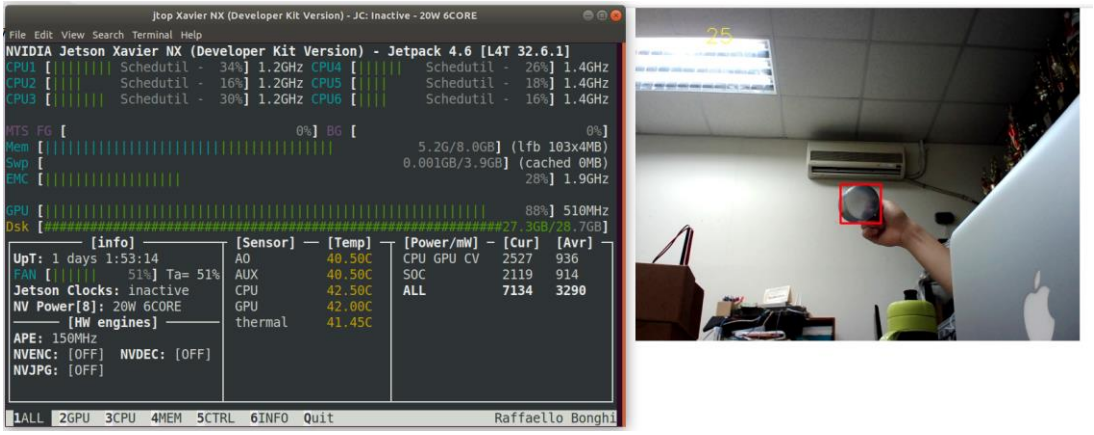

圖 3.17 Re^3 物體追蹤較遠之物體

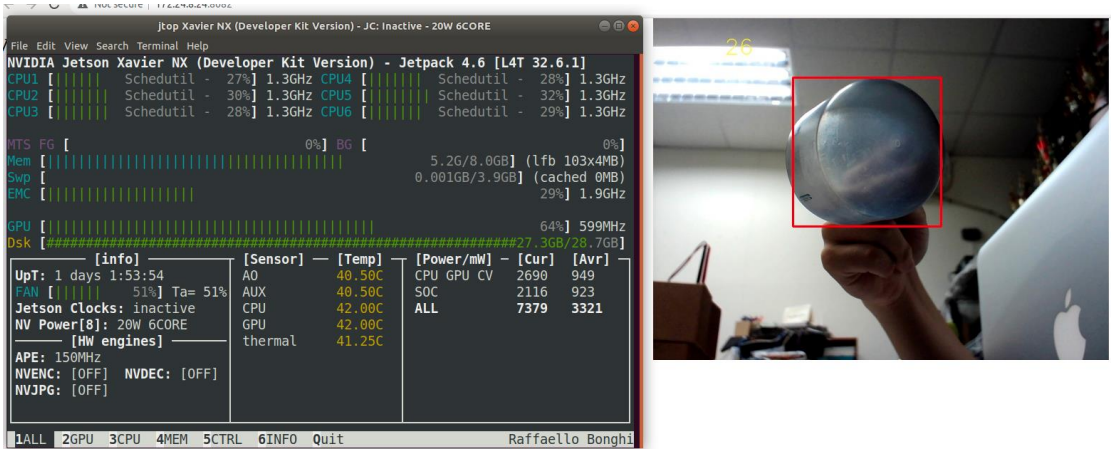

圖 3.18 Re^3 物體追蹤較近之物體

鏡頭的幀數設置在 30 的原因則是鏡頭本身使用的限制，如圖 3.19，鏡頭在不同的解析度會有不同的幀數限制，而我們是使用 OpenCV 中的 cv2.CAP_PROP_FPS 對其幀數去作改變，當設定的幀數不是鏡頭所規定的數字時，則鏡頭將無法讀取。

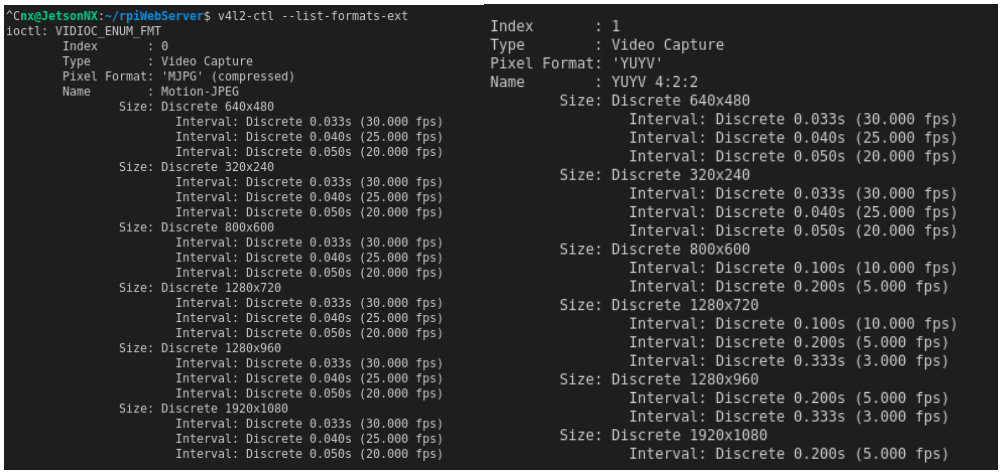

圖 3.19 鏡頭之 FPS 限制

2 各感測器取值與整合

需要讀取頭部旋轉角度資訊，剛開始時是先使用 GY-85 九軸感測器，但不斷測試後，發現 GY-85 在相同定點得出之資訊浮動相當高，若後續當作偵測頭部旋轉角度之感測器，可能會發生發射裝置不斷晃動等不穩定之情況，故後續採用 MPU9250 姿態九軸感測器，並使用開源於 Github 上之函式庫初始化 MPU9250 並讀取旋轉之資料，如圖 3.20 與圖 3.21。

MPU9250_Test

```
#include "MPU9250.h"
#include "Wire.h"

MPU9250 mpu;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.setup(0x68);
  Serial.println("Start!!");
  Init();
  delay(1000);
}

void loop() {
  if(mpu.update()) {
    static uint32_t prev_ms = millis();
    if (millis() > prev_ms + 100) {
      Serial.print("Yaw:");Serial.print(mpu.getYaw()); Serial.print(", ");
      Serial.print("Pitch:");Serial.print(mpu.getPitch()); Serial.print(", ");
      Serial.print("Roll:");Serial.println(mpu.getRoll());
      prev_ms = millis();
    }
  }
}

void Init() {
  Serial.println("Start Init Accel & Gyro");
  delay(1000);
  mpu.calibrateAccelGyro(); // Initialize Accel and Gyro
  delay(1000);
  Serial.println("End Init");
  delay(1000);
  Serial.println("Start Init Mag");
  Serial.println("round device around during mag calibration");
  delay(1000);
  mpu.calibrateMag(); // Initialize Mag
  delay(1000);
  Serial.println("Initialize Mag Finish");
}
```

圖 3.20 MPU9250 抓取數值及初始化之程式碼

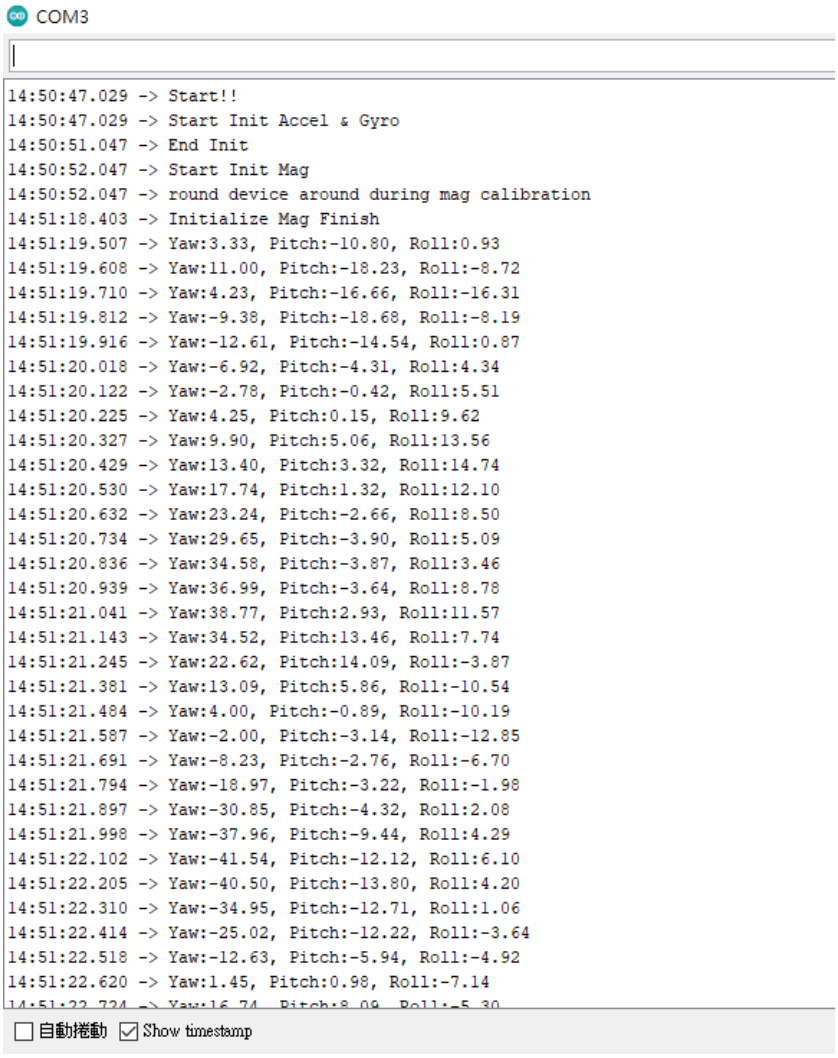

圖 3.21 初始化與讀取資料之執行結果

後續將 Yaw 值與 Roll 值傳入 API 中，再經由 API 轉換過後之數值，去控制各伺服馬達需如何旋轉，控制伺服馬達之程式碼，如圖 3.22、圖 3.23、圖 3.24。

```
class PCA9685:
    ######################
    # REG POSITION
    ######################

    __MODE1              = 0x00
    __MODE2              = 0x01
    __SUBADR1            = 0x02
    __SUBADR2            = 0x03
    __SUBADR3            = 0x04
    __PRESCALE           = 0xFE
    __LED0_ON_L          = 0x06
    __LED0_ON_H          = 0x07
    __LED0_OFF_L         = 0x08
    __LED0_OFF_H         = 0x09
    __ALL_LED_ON_L       = 0xFA
    __ALL_LED_ON_H       = 0xFB
    __ALL_LED_OFF_L      = 0xFC
    __ALL_LED_OFF_H      = 0xFD

    ######################
    # MODE1 FUNCTION Bits
    ######################

    __RESTART            = 0x80
    __SLEEP              = 0x10
    __AI                 = 0x20
    __ALLCALL            = 0x01
    __INVRT              = 0x10
    __OUTDRV             = 0x04

    def __init__(self):
        self.Horzionch = 3
        self.Verticalch = 1
        self.addr = 0x40
        self.bus = smbus.SMBus(1)  # 1 : /dev/i2c-1
        self.setFreq(50)
        self.setAutoIncrement(1)
        self.HStartAngle = 85
        self.VStartAngle = 80
        self.StepDir = 0

    def getRegChOffH(self,channel):
        return self.bus.read_byte_data(self.addr, 4 * channel + self.__LED0_OFF_H)

    def setRegChOffL(self,channel,val):
        self.bus.write_byte_data(self.addr, channel * 4 + self.__LED0_OFF_L , val )

    def setRegChOffH(self,channel,val):
        self.bus.write_byte_data(self.addr, channel * 4 + self
.__LED0_OFF_H  , val )
    def getRegMODE1(self):
        return self.bus.read_byte_data(self.addr, self.__MODE1)

    def setRegMODE1(self,val):
        self.bus.write_byte_data(self.addr, self.__MODE1, val)

    def setRegPrescale(self, _prescale):
        return self.bus.write_byte_data(self.addr, self.__PRESCALE, _prescale )
```

圖 3.22 伺服馬達之程式碼之一

```
def setAutoIncrement(self, val):
    if(val == 1):
        self.setRegMODE1(self.getRegMODE1() | self.__AI)
    elif(val == 0):
        self.setRegMODE1(self.getRegMODE1() &~ (self.__AI))

def sleep(self):
    self.setRegMODE1( self.getRegMODE1() | self.__SLEEP)
def wakeup(self):
    self.setRegMODE1( self.getRegMODE1() &~ self.__SLEEP )

def setValChOff(self, chan, off):
    if((off & 0xf000) > 0):
        print("error : too large value of off")
        return
    _off_l =     off & 0x00ff
    _off_h = ( ( off & 0xff00 ) >> 8 ) | ( self.getRegChOffH(chan) & 0x10)
    self.setRegChOffL(chan , _off_l)
    self.setRegChOffH(chan , _off_h)

def setFreq(self, freq):
    prescale_val = int(25e6/4096/float(freq) -1.0)
    self.sleep()
    self.setRegPrescale(prescale_val)
    time.sleep(0.005)
    self.wakeup()

def reset(self):
    self.bus.write_byte_data(self.addr, self.__ALL_LED_ON_L, 0)
    self.bus.write_byte_data(self.addr, self.__ALL_LED_ON_H, 0)
    self.bus.write_byte_data(self.addr, self.__ALL_LED_OFF_L, 0)
    self.bus.write_byte_data(self.addr, self.__ALL_LED_OFF_H, 0)
    self.bus.write_byte_data(self.addr, self.__MODE2, 0x04)
    self.bus.write_byte_data(self.addr, self.__MODE1, self.__ALLCALL | self.__AI)
    time.sleep(0.005)

def chDuty(self, chan, duty):
    if((duty > 100.0)or (duty < 0.0)):
        return
    _off_count = int(4096.0/100.0 * duty)
    self.setValChOff(chan,_off_count)
```

圖 3.23 伺服馬達之程式碼之二

```
def StartHorzion(self,centerdata,conflag,Horziontal):
    if conflag == 1:
        if centerdata - 320 < 0:
            self.HStartAngle = 80 - (320-centerdata)/4
        if centerdata - 320 > 0:
            self.HStartAngle = 90 + (centerdata-320)/4
        if centerdata - 320 == 0:
            self.HStartAngle = 85
    if conflag == 0:
        if Horziontal == 1:
            self.HStartAngle = 57
        if Horziontal == -1:
            self.HStartAngle = 113
        if Horziontal == 0:
            self.HStartAngle = 85
    val = 180-self.HStartAngle
    angle = (0.05 * 50) + (0.19 * 50 * val / 180)
    self.chDuty(self.Horzionch,angle)

def StartVertical(self,centerdata, conflag, Vertical):
    if conflag == 1:
        if centerdata - 240 < 0:
            StepDir = 0.2
        elif centerdata - 240 > 0:
            StepDir = -0.2
        elif centerdata - 240 == 0:
            StepDir = 0
        self.VStartAngle += StepDir
    if conflag == 0:
        self.VStartAngle += Vertical*0.2
    if self.VStartAngle < 60:
        self.VStartAngle = 60
    elif self.VStartAngle > 130:
        self.VStartAngle = 130
    val = 180-self.VStartAngle
    angle = (0.05 * 50) + (0.19 * 50 * val / 180)
    self.chDuty(self.Verticalch,angle)
```

圖 3.24 伺服馬達之程式碼之三

第四章 研究成果

4.1 研究方法

本專題將 Re^3 物體追蹤與網頁相關之操作放置 Nvidia Xavier NX 上執行，透過鏡頭獲取影像後，將影像呈現於網頁上，當使用者旋轉頭部時，就會透過伺服馬達同時旋轉發射裝置與鏡頭之方向，進而讓使用者得以不停觀察是否有需瞄準之物體，若使用者須鎖定物體後，則不再依使用者頭部旋轉而調整發射裝置與鏡頭之方向，而以物體為主。

4.2 實作展示

由於此專題皆建立於有模擬載具之情況，故與他組專題合作，將物體追蹤功能與頭盔聯動功能搭載於他組之載具與發射裝置上。

4.2.1物體追蹤成果

原本使用 OpenCV 中的 TrackerCSRT 追蹤時，除了幀率不佳外，也時常無法確切追蹤目標，換作使用 Re^3 追蹤物體時，明顯較為穩定，幀率與執行效能之差異如表 4.1 與表 4.2，而圖 4.1 與圖 4.2 為執行 Re^3 時呈現效果。

表 4.1 幀率之差異

物體追蹤技術	幀率
TrackerCSRT	3~15
Re^3	28~31

表 4.2 執行效能之差異

物體追蹤技術	單執行緒之最高使用率
TrackerCSRT	85~100
Re^3	30~40

圖 4.1 使用 Re[3] 追蹤距離較遠之物體時

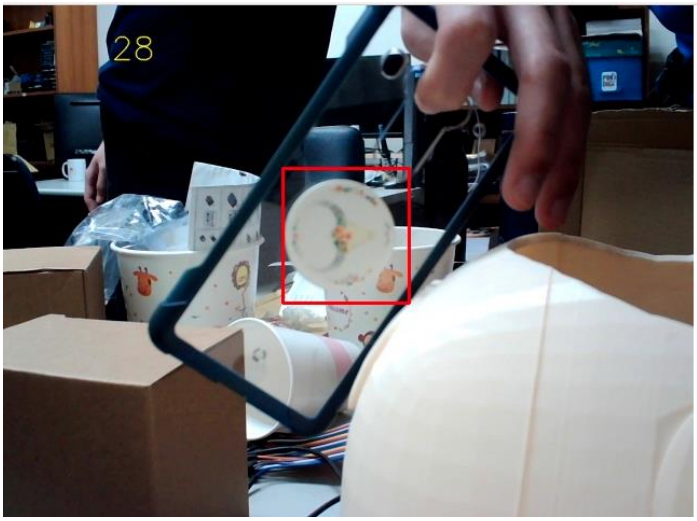

圖 4.2 使用 Re[3] 追蹤距離較近物體時

4.2.2頭盔聯動成果

透過頭盔上的 MPU9250 讀取使用者轉動頭部時的 Yaw 數值與 Roll 值，再各自判斷為 1、0 或 -1，透過 WebSocket 的方式傳送至 NX 上之 API Server，再控制各伺服馬達之旋轉，進而達成頭盔與發射裝置聯動的效果，轉動效果如圖 4.3 與圖 4.4。

圖 4.3 發射裝置原始位置

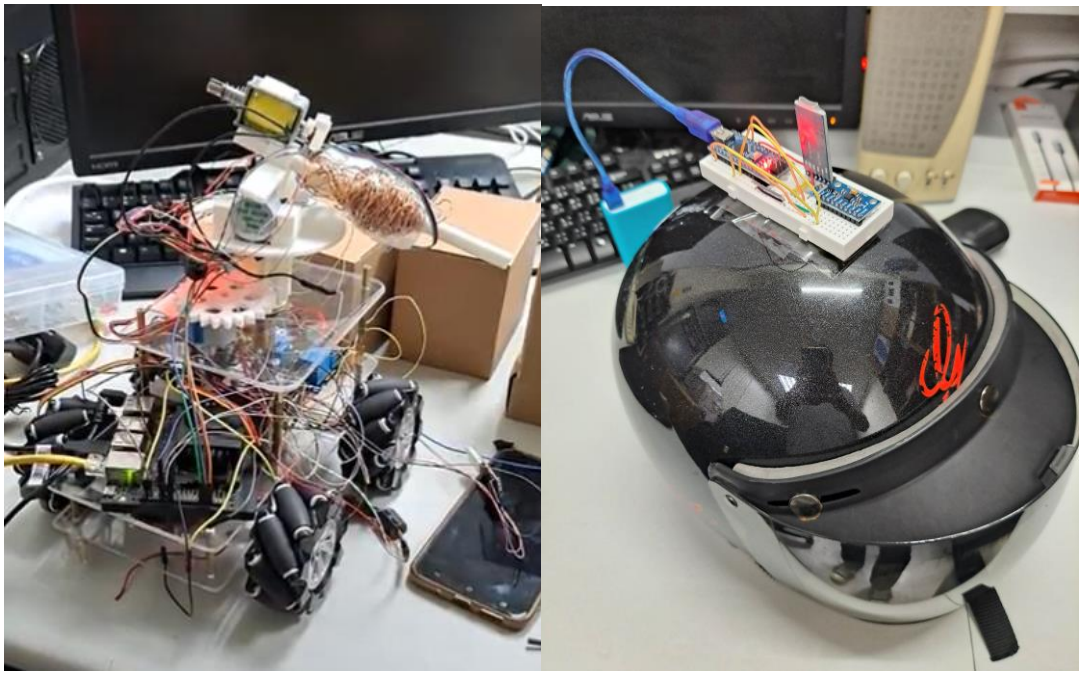

圖 4.4 頭部擺動後，發射裝置隨著旋轉

4.2.3實測幀率

在物體追蹤前，幀率約為 31~34 FPS，當開始物體追蹤時，幀率略為下降至 28~31 FPS，可得知，雖然因為後端不斷在抓取物體數據與運算，而造成幀率略為下降，如圖 4.5、圖 4.6、圖 4.7 與圖 4.8。

圖 4.5 尚未執行物體追蹤，較高之 FPS

圖 4.6 尚未執行物體追蹤，較低之 FPS

圖 4.7 執行物體追蹤時，較高之 FPS

圖 4.8 執行物體追蹤時，較低之 FPS

第五章 結論

5.1 結論

本專題主要是為了降低在軍事載具中操作人員可能發生的失誤，而利用 MPU9250 和 ESP32-S 與 NX 溝通後操控伺服馬達旋轉，達到頭盔與發射裝置聯動，並加上物體追蹤時物體移動的座標，利用垂直與平行之伺服馬達旋轉，達到在鎖定物體時能夠不斷自動瞄準目標，雖然無法實際建設於真正的軍事載具上，但透過與他組合作，達到模擬操控軍事載具之情況。

5.2 研究限制

在不斷傳送感測器資料給 NX 時，因為是透過 Wi-Fi 溝通，所以在距離上限制比較嚴重，在目前許多協定不斷實作出成品與物聯網不斷發展的現在，未來也有可能可以利用 5G 網卡來傳送資料，就能打破距離的限制。

5.3 未來展望

現在都是以模擬為前提作為實作，若未來能夠實際了解載具中發射裝置與車體之旋轉關係，改進我們頭盔與發射裝置旋轉之關係，進而實現更貼近現實情況之模擬，或是更加了解軍中載具實際操作情況，加以改進我們在實際操作時應如何改進，使得減少操控載具上之發射裝置人員壓力。

參考文獻

[1] 阿帕契直升機 - wiki

https://zh.wikipedia.org/wiki/AH-64%E9%98%BF%E5%B8%95%E5%A5%91%E7%9B%B4%E5%8D%87%E6%A9%9F

[2] ESP32S 數據手冊

https://docs.ai-thinker.com/esp32/spec/esp32s

[3] 步進馬達的動作原理

http://km.emotors.ncku.edu.tw/emotor/worklog/EMTRC/motor_learn/elearning/motor/3/lecture_3.2.pdf

[4] Nvidia Xavier NX

https://blogs.nvidia.com.tw/2019/11/06/nvidia-introducing-jetson-xavier-nx/

[5] Re^3-tensorflow - github

https://github.com/danielgordon10/re3-tensorflow

[6] 3D 感測技術：什麼是飛時測距

https://www.stockfeel.com.tw/3d%E6%84%9F%E6%B8%AC-%E9%A3%9B%E6%99%82%E6%B8%AC%E8%B7%9D-tof/

[7] Flask

https://blog.techbridge.cc/2017/06/03/python-web-flask101-tutorial-introduction-and-environment-setup/

[8] 步進馬達與伺服馬達區分

http://romeofan.synology.me/mainhome.files/motor/servo&stepper.html

[9] OpenCV

https://opencv.org/

[10] OpecnCV 的前景/背景分離技術

https://chtseng.wordpress.com/2018/11/03/opencv%E7%9A%84%E5%89%8D%E6%99%AF%EF%BC%8F%E8%83%8C%E6%99%AF%E5%88%86%E9%9B%A2%E6%8A%80%E8%A1%93/

[11] 淺談遞歸神經網路與長短期記憶模型

https://tengyuanchang.medium.com/%E6%B7%BA%E8%AB%87%E9%81%9E%E6%AD%B8%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-rnn-%E8%88%87%E9%95%B7%E7%9F%AD%E6%9C%9F%E8%A8%98%E6%86%B6%E6%A8%A1%E5%9E%8B-lstm-300cbe5efcc3

[12] 什麼是 CUDA？

https://blogs.nvidia.com.tw/2012/09/10/what-is-cuda-2/

附錄

附錄一 前端呼叫追蹤之程式碼

```
$(document).ready(function () {
  getCoordinates(0);
});
function getCoordinates(track_sign) {
  sign = track_sign;
  var data = {
    data: JSON.stringify({
      time_count: "1",
      track_sign: sign
    }),
  }
  $.ajax({
    url: "image_roi",
    type: "post",
    data: data,
    dataType: 'html',
    success: function (data) {
      console.log("success");
    },
    error: function (xhr, ajaxOptions, thrownError) {
      console.log(thrownError);
      alert(thrownError);
      alert("data send fail")
```

```
        }
    })
}
```

附錄二 調整鎖定框大小之程式碼

```
function BoxAdjust(adjust_sign){
  sign = adjust_sign;
  var data = {
     data: JSON.stringify({
        sign : sign
     }),
  }
  $.ajax({
     url: "tracking_size",
     type: "post",
     data: data,
     dataType: 'html',
     success: function(){
        console.log("send success")
     },
     error: function (thrownError){
        console.log(thrownError);
        alert(thrownError);
     }
  })
}
```

附錄三 執行物體追蹤

```
import cv2
import argparse
import glob
import numpy as np
import os
import time
import sys

basedir = os.path.dirname(__file__)
sys.path.append(os.path.abspath(os.path.join(basedir, os.path.pardir)))
from tracker import re3_tracker
from re3_utils.util import drawing
from re3_utils.util import bb_util
from re3_utils.util import im_util
import DottedFrame
from FinalCode import helmet
from constants import OUTPUT_WIDTH
from constants import OUTPUT_HEIGHT
from constants import PADDING


class VideoCamera(object):
  def __init__(self, flip = False):
      self.cam = cv2.VideoCapture(0)
```

```
    self.cam.set(cv2.CAP_PROP_FPS, 30)
    self.outputBoxToDraw = None
    self.boxToDraw = np.zeros(4)
    self.x = 320
    self.y = 240
    self.time_count = 0
    self.track_sign = 0
    self.Scale = 0
    self.mousedown = False
    self.mouseupdown = False
    self.initialize = False
    self.tracker = re3_tracker.Re3Tracker()
    self.StartFrameTime = 0
    self.PrevFrameTime = 0
    self.size = 50
    self.StartCoordinate = (self.x-self.size,self.y-self.size)
    self.EndCoordinate = (self.x+self.size,self.y+self.size)
    self.ChangeStatus = -1

def show_webcam(self):
    while True:
        self.StartFrameTime = time.time()
        ret_val, img = self.cam.read()
        if not self.ChangeStatus == -1:
            self.BoxAdjust()
        if self.track_sign==1:
```

```
            if self.time_count=="1":
                self.boxToDraw = (self.x-self.size,self.y-
self.size,self.x+self.size,self.y+self.size)
                self.outputBoxToDraw = self.tracker.track('webcam', img[:,:,::-1],
self.boxToDraw)
                self.time_count = "0"
            else:
                self.outputBoxToDraw = self.tracker.track('webcam', img[:,:,::-1])
            cv2.rectangle(img,
                  (int(self.outputBoxToDraw[0]), int(self.outputBoxToDraw[1])),
                  (int(self.outputBoxToDraw[2]), int(self.outputBoxToDraw[3])),
                  [0,0,255], PADDING)
            boxcenter = (
                  (int(self.outputBoxToDraw[0])+int(self.outputBoxToDraw[2]))/2 ,
                  (int(self.outputBoxToDraw[1])+int(self.outputBoxToDraw[3]))/2
               )

        else:

DottedFrame.drawrect(img,self.StartCoordinate,self.EndCoordinate,(0,0,255),1,'
dotted')
        fps = 1/(self.StartFrameTime - self.PrevFrameTime)
        self.PrevFrameTime = self.StartFrameTime
        fps = int(fps)
        fps = str(fps)
```

```
            cv2.putText(img, fps, (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 1, cv2.LINE_AA)
            rect,jpeg = cv2.imencode('.jpg',img)
            return jpeg.tobytes()
    def BoxAdjust(self):
        if self.ChangeStatus == 1:
            self.size -= 2
            self.StartCoordinate = (self.x-self.size,self.y-self.size)
            self.EndCoordinate = (self.x+self.size,self.y+self.size)
            self.ChangeStatus = -1
        elif self.ChangeStatus == 0:
            self.size += 2
            self.StartCoordinate = (self.x-self.size,self.y-self.size)
            self.EndCoordinate = (self.x+self.size,self.y+self.size)
          self.ChangeStatus = -1
```