**WNFA lab1 - CamCom Team03**

**組長** B00902094 資工系三 白惠婷

**組員** B00902092 資工系三 陳璽凉

B00902117 資工系三 張奕嫻

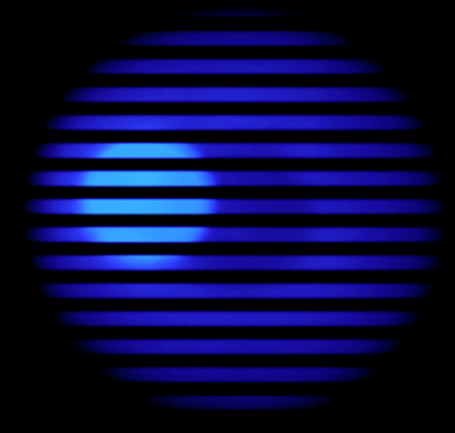
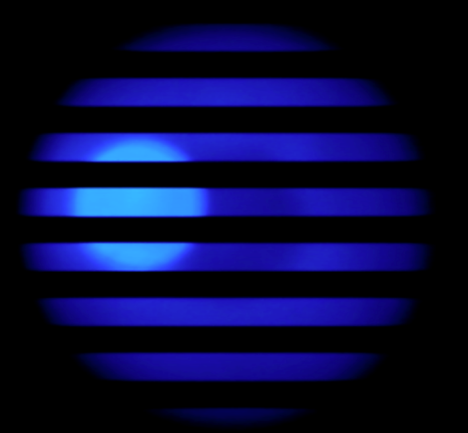
B00902123 資工系三 楊筑媛

R01922110 資工碩二 彭菘瑋

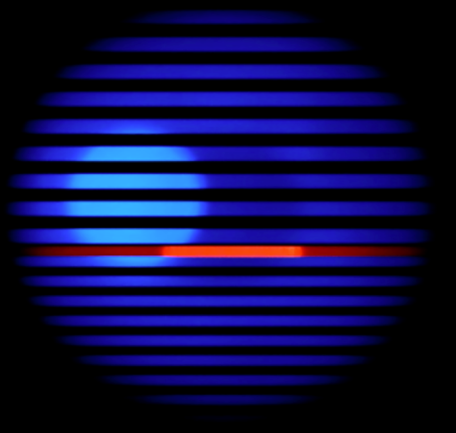
# **Tx - 傳輸的實作（tx.c）**

#### 傳輸方法

* 我們是用FSK的方式來傳data，每一個frame都以不同的頻率作為傳送的方法，固定為藍色；以下兩張圖即為不同頻率的照片。



* 使用的是CTC mode with interrupt handler，每1/30s切換一次頻率。
* 為了避免錄影的時候一個 frame照到兩種頻率，我們在每1/30s的瞬間加上瞬間的紅光，作為分隔兩個frame的線條。當畫面中出現紅線即代表一個frame出現兩種頻率，在decode的時候即會checksum error。我們的程式還無法自動依紅線分隔兩種頻率，而是要改以手動調整webcam高低的位置來改變紅線的位置，只要調整讓藍光出現，紅線消失即不會發生frame中有兩種頻率的情形。



#### 

#### Input

* 把要傳送的字串直接存在Arduino的code中，以char array的方式儲存為tx\_word。
* tx\_data\_pos是傳到第幾個data。
* tx\_data才是真正要傳的data，存的值是OCR1A的值。（另外還要傳checksum和結尾byte，會在下面提到）
* 我們是把一個字元（一個byte）分成兩部分來傳，每個frame傳4個bit，即4\*30=120bps。總共由16種頻率來代表不同的4個bit。

tx\_data[i\*2] = (int(tx\_word[i])/16+1)\*5;

tx\_data[i\*2+1] = (int(tx\_word[i])%16+1)\*5;

* PIN8是藍燈。
* PIN9是紅燈。

計算checksum

* 計算checksum是為了在解碼時能夠分辨解讀出來的字串有沒有錯。
* 以modular sum的方式計算，公式為

checksum[i] = (checksum[i-1]\*256 + tx\_word[i]) % 255

* 由第一個字計算至最後一個字，並將checksum存在字串最尾端，最後加上一個結尾byte，即0x00
* 以Hello為例，即應該傳送含checksum及結尾共7個byte。

#### timer3

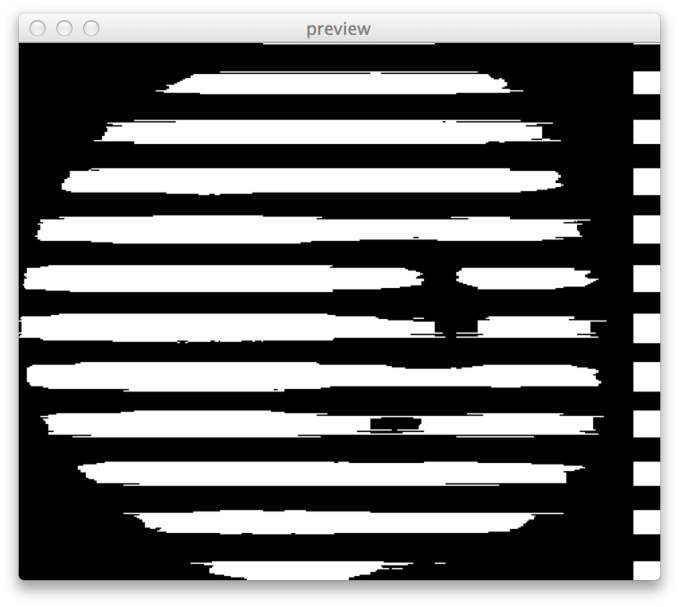
* OCR3A設成每1/30s就做一次timer3的interrupt。
* Interrupt里做的事情：
* 亮紅燈，以區分上一個頻率和下一個頻率。
* 切換OCR1A，即準備傳下4個bit。
* 把TCNT歸零，這樣兩個timer比較同步。

#### timer1

* OCR1A用來調藍燈閃爍的頻率，一共有16種。
* 在timer1的interrupt里要讓紅燈暗掉，這樣才不會出現一片紅和藍，紅光只會出現一下子。

**Rx - 接收的實作（rx.py）**

* 軟體：Python 2.7，OpenCV 2.4.7
* 硬體：Logitech C310 @ mac
* 控制讓快門固定的程式：CameraControl Demo App (opensource)  
  http://phoboslab.org/log/2009/07/uvc-camera-control-for-mac-os-x
* 先把周圍不需要的黑色部份裁掉。
* 將畫面進行取threshold，得到黑白的結果。



* 每個橫線上對每個pixel取or。即同一個橫線上有任一個pixel為白色，即視為白，否則為黑。得到的結果在上圖的右邊。
* 由上而下，找出所有白條紋及黑條紋的中點，並取所有中點間距的中位數。最上面及最下面的的條紋（不論黑白）因為不完整不列入考慮。
* 最後依測出來的對應表找出原本傳送的值。

**數據對應表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit | OCR1A | 對應 的頻率 | 亮紋或暗紋 的持續時間 | 影像中的亮紋 或暗紋高度 |
| Sequence |  | (Hz) | (ms) | (pixel) |
| 0000 | 5 | 2604 | 0.4 | 3.00 |
| 0001 | 10 | 1420 | 0.7 | 5.53 |
| 0010 | 15 | 977 | 1.0 | 8.06 |
| 0011 | 20 | 744 | 1.3 | 10.59 |
| 0100 | 25 | 601 | 1.7 | 13.12 |
| 0101 | 30 | 504 | 2.0 | 15.65 |
| 0110 | 35 | 434 | 2.3 | 18.18 |
| 0111 | 40 | 381 | 2.6 | 20.71 |
| 1000 | 45 | 340 | 2.9 | 23.24 |
| 1001 | 50 | 306 | 3.3 | 25.77 |
| 1010 | 55 | 279 | 3.6 | 28.30 |
| 1011 | 60 | 256 | 3.9 | 30.83 |
| 1100 | 65 | 237 | 4.2 | 33.36 |
| 1101 | 70 | 220 | 4.5 | 35.89 |
| 1110 | 75 | 206 | 4.9 | 38.42 |
| 1111 | 80 | 193 | 5.2 | 40.95 |

#### 

# 

# 

# **問題與解決**

#### **儀器**

一開始我們拍不出條紋，以為一般手機都可以照到條紋，但後來發現只有iphone拍出來的效果較好，其他手機比較不穩定。因為距離太近，沒有微距鏡頭的相機會無法對焦，所以也無法拍出條紋。最後因為我們想要以realtime做decode，但很多webcam無法控制固定快門，因此後來選擇採用有支援Universal Video Camera協定的webcam，並且找到程式可以固定快門。

#### **輸入input**

一開始使用serial monitor，但是當使用serial monitor將一個字串寫進array裡，發現每讀一個byte要跑一次loop，較難控制。詢問助教之後，我們後來決定直接把input寫進code裡。

#### **頻率的上下限**

一開始不知道可以用哪一種頻率比較好。頻率越低，拍出的條紋越粗；頻率越高，條紋就越細。條紋太粗的話可能會超過frame的范圍；條紋太細，會解讀不出來。所以我們先用很高頻或很低頻來測試可用的頻率範圍的上下限，再決定用哪幾種頻率來代表字元。

#### **可用頻率的間距**

在影像解碼的時候一開始打算頻率可分成128甚至256種（7~8bit/frame），後來發現可能誤差太大難以解碼，最後改用4bit/frame，即分成16種。

#### **延遲**

為了傳遞訊息，需要讓一種頻率持續1/30s，我們剛開始時是用currenmillis() - previousmilis() > interval來讓一種頻率持續1/30s，再切換下一個頻率。但interval用的是整數，所以無法維持準確的1/30s。因此決定多加一個timer3來當作clock，可以和timer1同步執行，比較精準。因為不是每一種錄影工具都是準準30fps，還是要根據錄影工具來調timer3的頻率，用這個方法可調頻調到比較精確。

#### **誤差問題**

因為誤差的關系，一個frame可能包含兩種頻率，最後的解法是使用一條線當作分隔頻率的標示。過程中還發現藍色跟綠色很容易混在一起，眼睛看的時候不明顯，最後選擇藍色當作頻率改變的訊號，及紅色當作不同頻率的分隔線。照到紅光時解碼就會出錯，所以盡量避免照到紅光。如果有error，會顯示有checksum error。要傳的字串會一直進行解碼，所以可以看出找到紅光時有checksum error，如果是對的話，就沒有checksum error。