

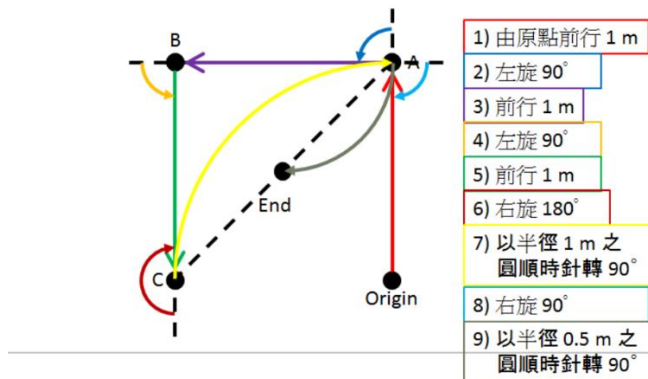
期末成果競賽

組員:楊翰祥 0510379、宋其諭 0510888

實驗目的:期末專題成果競賽主要分成指定項目和創意發想兩大主思想前提下，設計出位置控制、速度控制和影像辨識三大控制的對抗賽

競賽項目一:指定路徑行走(排名賽)

給定依指定路徑，開始進行比賽後不能以任何外加刑事進行控制，以起點集中點之直線距離當作牌為基準，實際與理想距離差越小者牌位越高，若距離差越小，則以所花費時間長度為勝負依據。



*註:

1. 由 origin 至 end 需 3 分鐘內完成
2. 可不搭載 rpi 層
3. 中間每一點都必須車體覆蓋到，少碰一個扣一分

成果:練習時有完整成功過，初賽時分組排名第二

設計理念:

我們之前設計建模方式想嘗試以兩種平衡方式來達到位置控制，但遇到上下坡時會遇到無法上坡或下坡衝刺速度過快，衝出場外、效果不彰等的問題，因此我們後來緊急改變策略採用改變車傾角搭配 Cascade-Form 的方式達到位置控制。

實現

直線行走的部份，我們讓車傾角傾斜連動車輪轉動，當達到目標地固定距離範圍內時將 Reference 設為零度，使車子利用先前產生之慣性來移動至目標地，若有 overshoot 則依靠位置控制 Cascade-Form 的 PID 讓車身回到制定位置。

而身體旋轉的部分，我們將 PID 回授要輸出之 PWM 左右倫分別做一加一減的動作，讓左右產生輪差，達到轉彎的效果

設計問題討論

Q1:馬達硬體本身存在些微差異，無法行走直線如何解決?

A: 兩輪馬達本身輸出有些微的誤差，因此在兩輪有寫一個校正的函數，

```
float OutputCalibration(float Calibration){ //微調左右馬達
    float outputA = output + TurnL ;
    float outputB = output + Calibration+TurnR ;
    // move the car
    motorA.Rotate( outputA );
    motorB.Rotate( outputB );
}
```

讓馬達可以達到直線行走的目的

Q2: 當在行走上波無法前進時該如何解決此問題？

A: 當車身無法向前移動連續達三秒鐘則會將車傾角擺最大擺幅加大 0.0005rad，讓車身獲得更大前進的驅動力

Q3: 當車速過大導致在做位置控制時會有過多的 overshoot，甚至會發生不受控的情況發生，請問如何解決？

A: 由於我們首要目標是要讓車子加速時車傾角在一定範圍內以確保維持車身的基本平衡，因此我們會**最大車傾角的限制**，但有最大車傾角的限制，會讓車子在下波路段發生暴衝抑或過大 overshoot 的問題，所以我們會特別針對下波路段的 **reference(校準零度值)** 做些微的調整。

```
} else if (flag == 2) {
    for (int i = 0; i < 1200; i++) {
        posController.SetSaturation(0.025, -0.025);
        ReferencePsi = 0.004; //Reference 微調
        delay(5);
    }
}
```

```
// pos Controller //最大車傾角
posController.SetSaturation(0.01, -0.01); // 0.02 ~ -0.02rad
posController.SetToleratedError(5); // 5cm
posController.SetPID(0.5, 0.2, 0.01); // KP, KI, KD 0.001, 0.2 ,0
posController.SetReference(Reference); // Reference
```

Q4: 如何走曲線路徑？

A: 我們將路徑切成十段，旋轉和直線交錯，採用採用黎曼和「分割、求和」的理念組成一段圓滑的曲線。

```
case 7:
    //Calibration=-0.1;
    Addpsi(-15,3000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    turn(0,5,1);
    Addpsi(-20,4000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    turn(0,15,1);

    Addpsi(-20,4000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    turn(0,5,1);
    Addpsi(-20,3000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    turn(0,5,1);
    Addpsi(-20,2000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    //delay(10000);
    break;

case 8:
    turn(0,90,1);
    Addpsi(-15,2000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    turn(0,15,1);
    Addpsi(-15,3000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag

    delay(10000);
    turn(0,5,1);
    Addpsi(-20,2000,3); //克服最大靜摩擦，增加車傾角，//ms, //flag
    delay(10000);
    default:
        break;
```

程式碼

比賽項目一：Arduino：main.ino

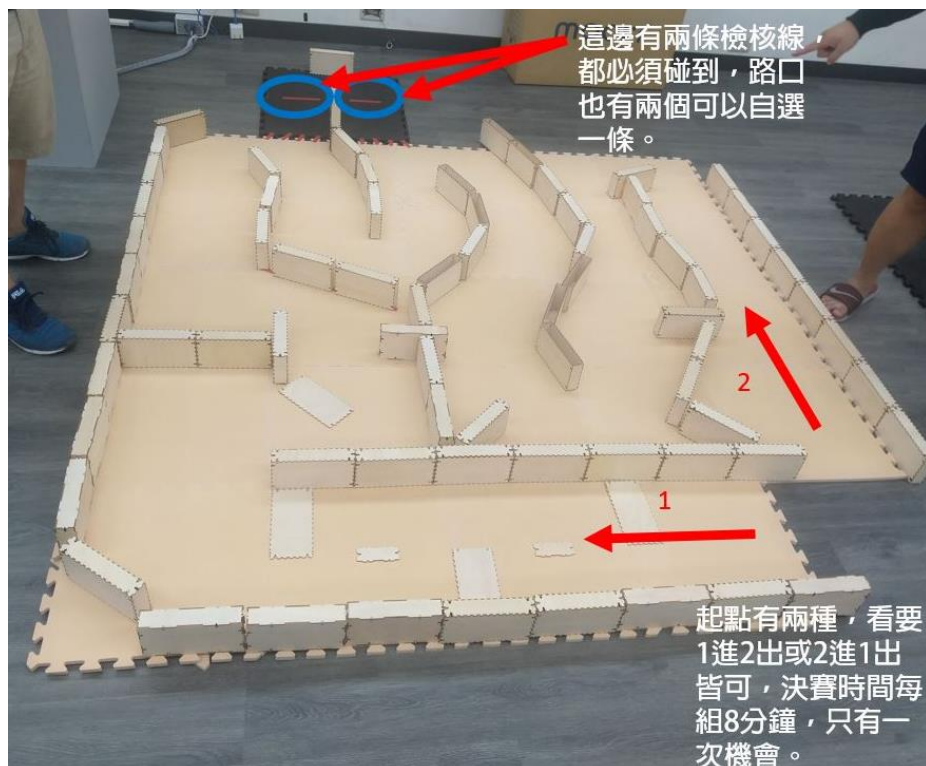
：影片：demo1.mp4(之前沒拍到成功通過的影片)

因為是走固定路徑因此我們在 loop 迴圈中 switch 不斷跑不同的 case，

```
switch(level){
  case 0:
    delay(2000);
    break;
  case 1:
    y=0;timercount=0;
    Calibration=-0.1;
    Addpsi(-60,8000,1);//克服最大靜摩擦，增加車傾角
    break;
  case 2:
    turn(1,90,1);
    delay(1000);
    break;
  case 3:
    y=0;timercount=0;
    // Calibration=-0.15;
    Addpsi(-85,10000,2);//克服最大靜摩擦，增加車傾角
    break;
  case 4:
    turn(1,90,2);//向轉90度 為(0,90)
    delay(2000);
    break;
  case 5:
    Calibration=-0.15;
    Addpsi(-60,15000,3);//克服最大靜摩擦，增加車傾角，//ms, //flag
    break;
  case 6:
    turn(0,180,1);//向右轉90度 為(0,90)
    delay(3000);
    break;
}
```

競賽項目二：賽道競走(排名賽)

給定依指定賽道(包還直線 & 轉彎)，從固定起始點開此計時，可以外加控制方式進行控制(手機或電腦)，已抵達之時間做為牌位之基準



*註：

1. 過程中可以任何遠端裝置進行遙控。
2. 賽道邊框部分底下會貼膠帶，上方則為可移動之物體作為護欄，移動過程中造成護欄移動每個護欄加時 5 秒。
3. 可不搭載 rpi 層。

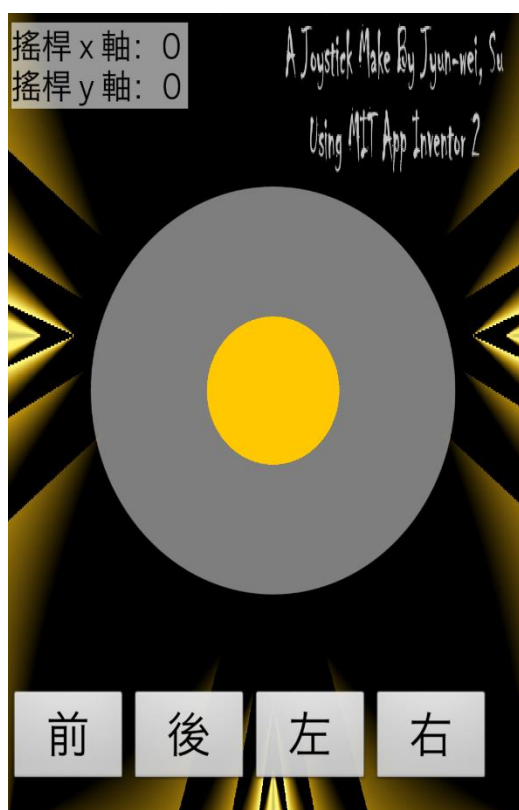
成果:複賽冠軍，花費 82 秒，西堤牛排券到手 XD

設計理念:

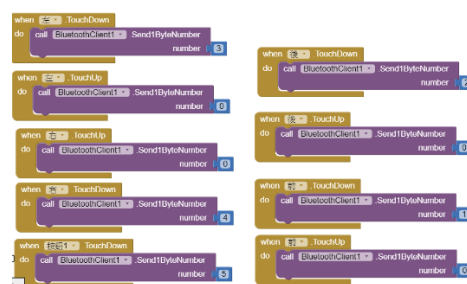
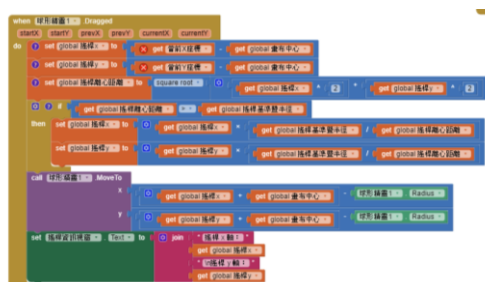
此項賽道競走為了縮短我們比賽耗時，我們加入了速度控制的機制讓車子行走得以保持理想的速度狀態，我們控制速度的方式依然以車傾角作為前進速度的依據，一方面我們有設計兩組PID來做更細膩的平衡，當車身向前移動和車身旋轉、靜止分別會提供不同的PID確保車子本身的穩定性，另一方面實測出在平衡狀況下車身可承受的最大傾角，作為速度控制的依據。

#手機程式介面

一開始我們利用時間中斷的方式將讀取道的位置和方向透過藍芽傳輸，但要讓操控者隨心所欲準確的用搖桿需要一位熟練的玩家，在幾次試驗後，決定將介面設置為具備調速功能的按鍵式介面，此種策略能有效的降低玩家手控失誤的情況發生。



App inventor 傳輸判斷式



設計問題討論

Q1:如何調出優秀的PID?

A: 我們參考 <https://www.youtube.com/watch?v=cSiycDHssxY&t=105s>

調出屬於速度控制穩定的系統

Q2:當遙控者失誤發生，如何在程式設計上將扣分降到最低?

A:當車子失控會發生暴衝的狀況，因此我們當車傾角大於 45 度時，我們便判定車子已失去平衡控制，立即將馬達轉速設為零，避免造成大量失分

程式碼

比賽項目二：Arduino：main.ino

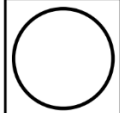
：AppInventor：pid_control_1_final.aia

：影片：demo2.mp4


競賽項目三:影像辨識迷宮自走(對抗賽)

參賽者須結合雙輪車的平衡控制、位置控制和影像辨識之空能，完成具有變態圖形指令的雙輪平衡車，並以此通過對手所設計的迷宮題目。


1



2




3





1 → 停止 & 等待 & 倒地


2 → 左旋 90 度 & 前進

3 → 右旋 90 度 & 前進

底色

 (1)

 (2)

 (3)

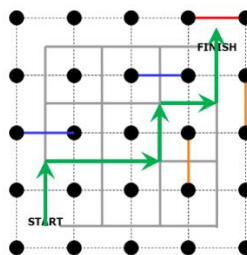
動作命令\單位	(1)	(2)	(3)
1. 靜止平衡 & 等待 & 倒地	1 秒	2 秒	3 秒
2. 左旋 90 度 & 前進格數	1 格	2 格	3 格
3. 右旋 90 度 & 前進格數	1 格	2 格	3 格

→ 黑點：立柱，用來區分每一個格子的範圍以及做為指令板的支撐。

→ 灰線：標示在地板上的線，定義車子移動之狀態 & 可協助車子移動。

→ 藍線、橘線、紅線：不同的指令牌代表符號，假設分別為右轉，左轉以及停止。

→ 以地圖為例，若參賽者依照指示成功到達終點，則全部可獲得： $10 \times 6 \text{次到達格子} + 30 \times 4 \text{次轉彎} + 100 = 280 \text{分}$

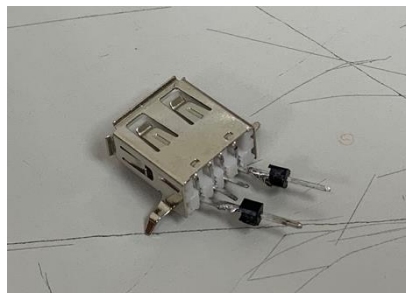


*註：

1. 參賽者事先使用多種指令卡為對手設計從起點到終點的路徑，對方則需**限時三分鐘內**盡量完成關卡。最後比較兩方的分數判斷勝負。
 2. 設計至少經過七格，最多不得超過 15 格。
 3. 每經過一格得 10 分，轉彎得 30 分，達終點得 100 分，以分數高者為勝
- 成果:分組晉級一次，敗給無平衡功能(會亂數斜向暴衝)的車體。

硬體設置：

將 Rpi 裝上去的樣子如圖，因為整個 Rpi 立著會造成重心變高，所以我們採用橫著的方式固定。並且因為下方板子摔太多次，已經用熱融膠固定，無法拿下做電源的焊接，所以我們將轉接頭的正負極焊上排針公頭，再用杜邦線接到主板的正負端做供電。(如下圖)



軟體設置：

在 Rpi 上編譯 python3 的 opencv，花了大概 4 小時

```
prophet — 777@prophet: ~ — ssh 777@prophet.local — 80x24
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[ wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
8 inet 172.20.10.2 netmask 255.255.255.240 broadcast 172.20.10.15
[ inet6 2402:7500:50a:32bd:4f89:600d:3645:fb81 prefixlen 64 scopeid 0x0<
global>
[ inet6 fe80::b7aa:7919:fca5:bbca prefixlen 64 scopeid 0x20<link>
ether b8:27:eb:79:91:18 txqueuelen 1000 (Ethernet)
RX packets 156 bytes 27054 (26.4 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 114 bytes 19289 (18.8 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

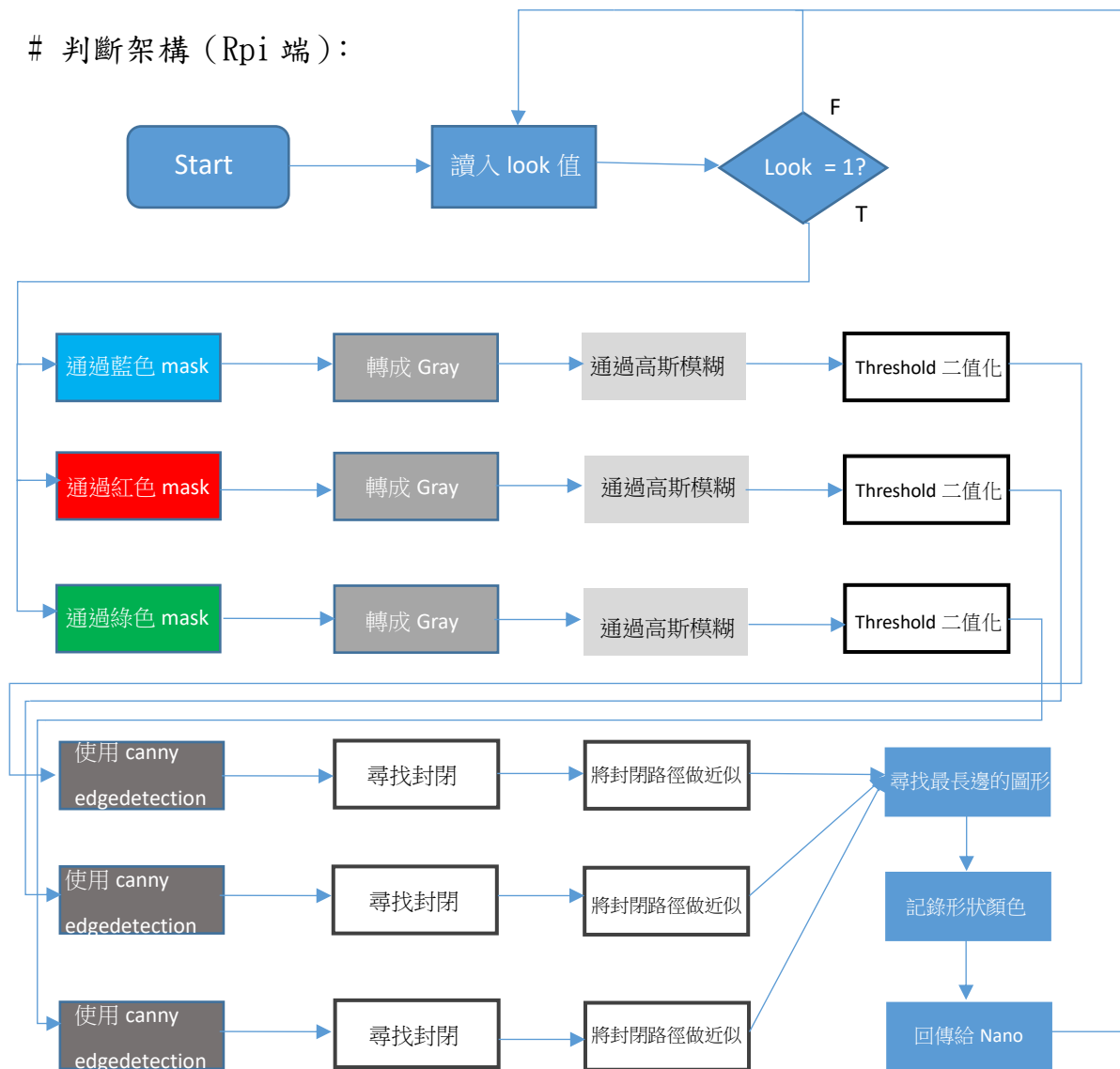
6l
[ 777@prophet:~ $ python3
9 Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[ [GCC 6.3.0 20170516] on linux
4: Type "help", "copyright", "credits" or "license" for more information.
[ >>> import cv2
d >>> cv2.__version__
[ '3.4.1'
c >>>
[ 16:49:05.006 NotebookApp] Adapting to protocol v5.1 for kernel 0f853ect-r0ad-
4b14_0ebh_87ebdc72027d
```

比較麻煩的是網上有些教學範例用的 python 版本不一樣，有 error 還要去查看是不是版本不一樣。

在做影像辨識的基本思路是將原圖通過三個顏色的 mask 再做近似以及計算

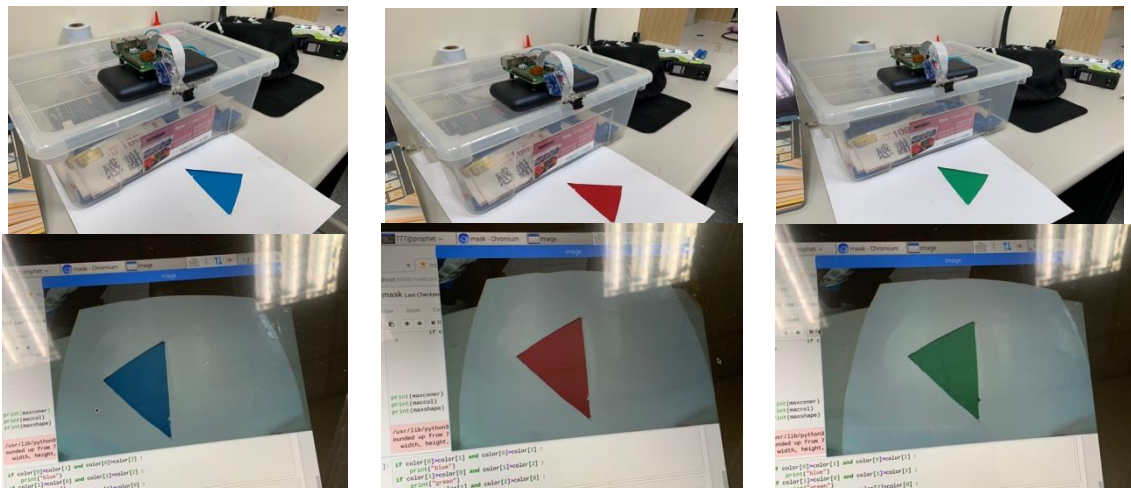
頂點數量做圖形分辨。

判斷架構 (Rpi 端):

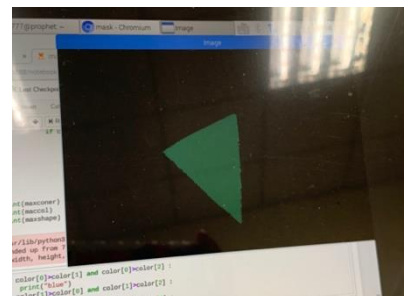
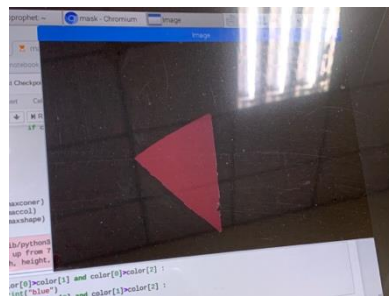
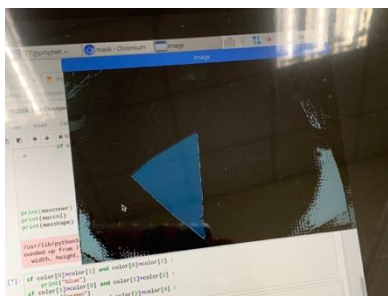


測試輸出：

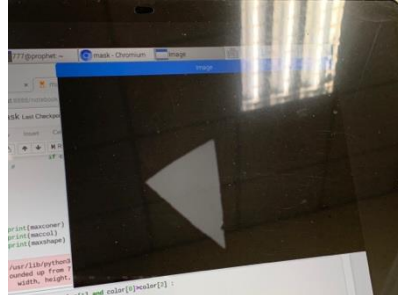
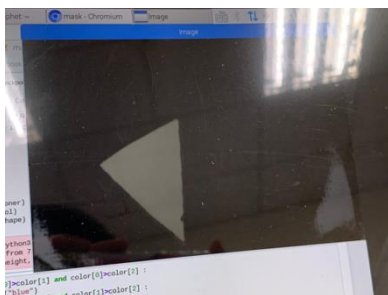
- 以下為三角形：



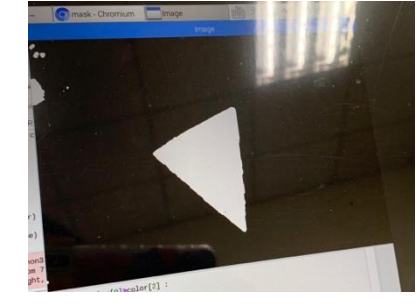
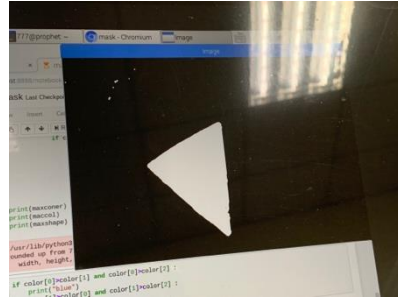
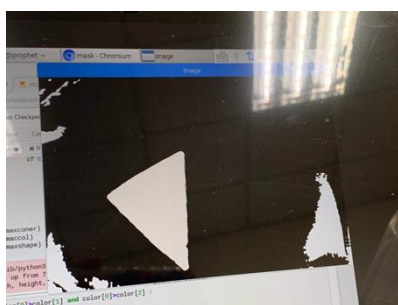
通過三個 mask



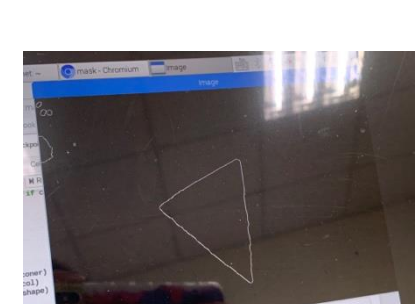
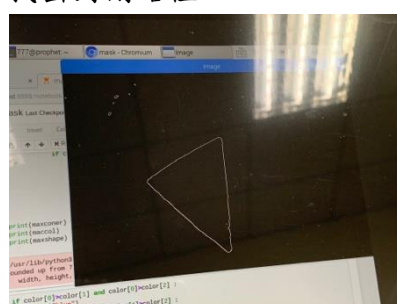
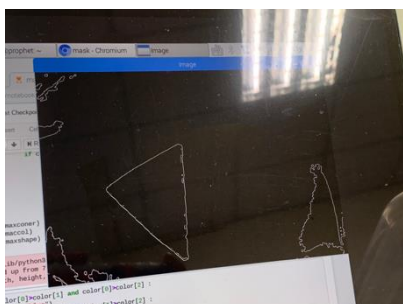
轉成灰階



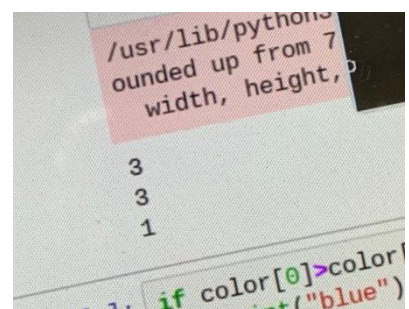
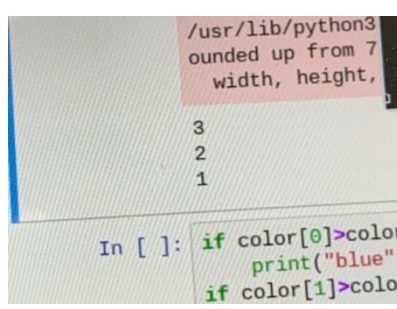
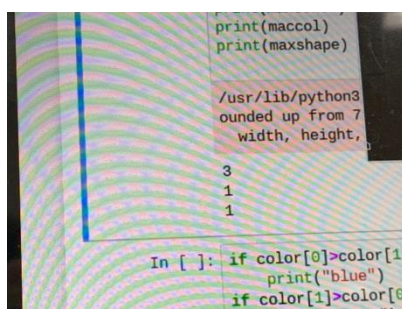
threshold 二值化



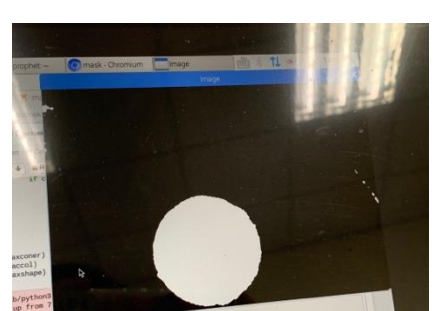
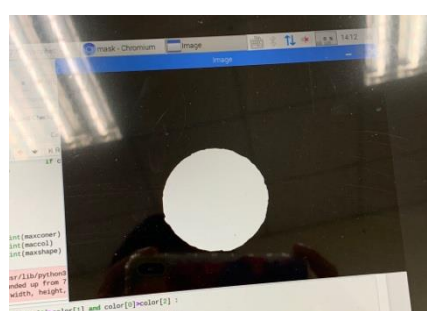
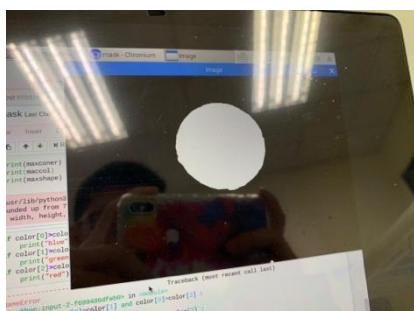
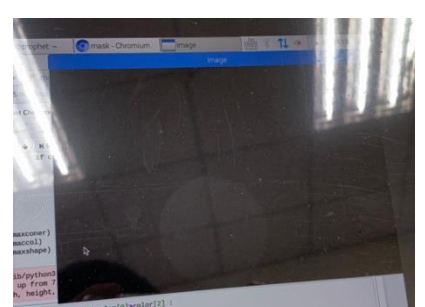
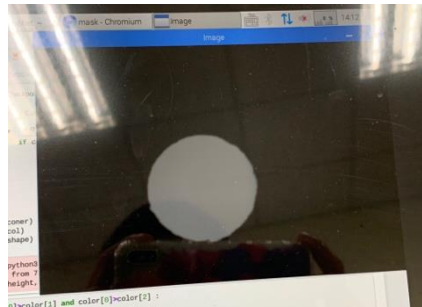
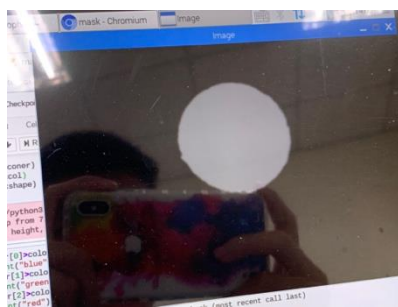
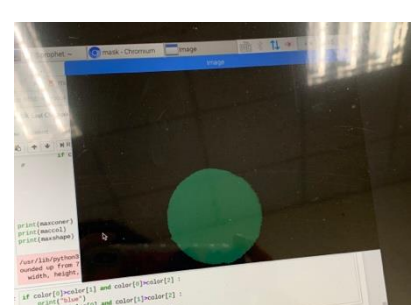
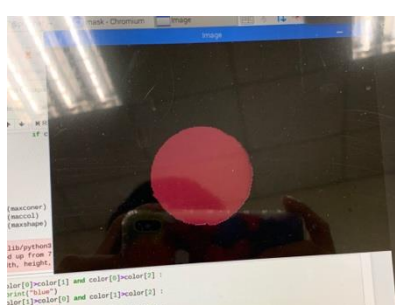
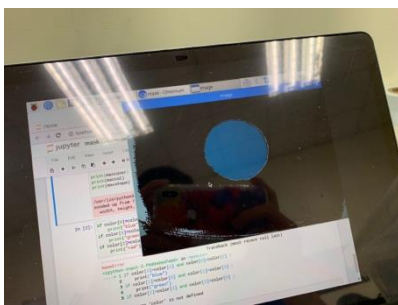
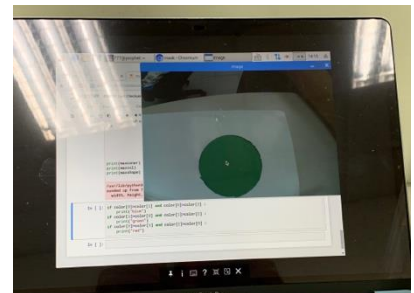
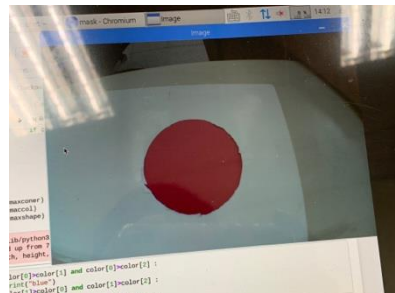
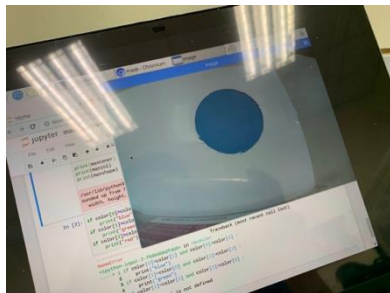
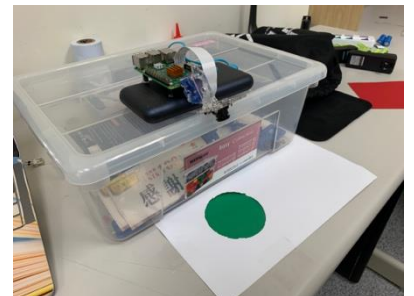
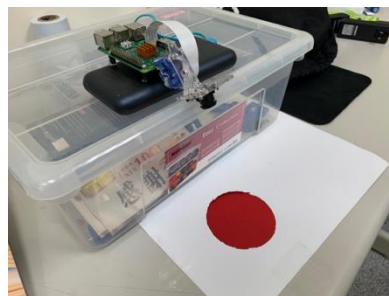
找出封閉路徑

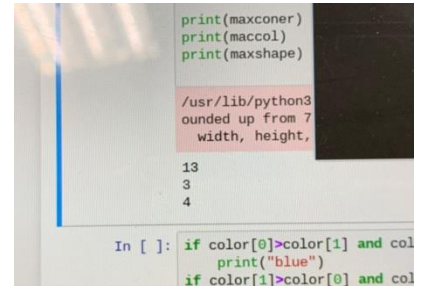
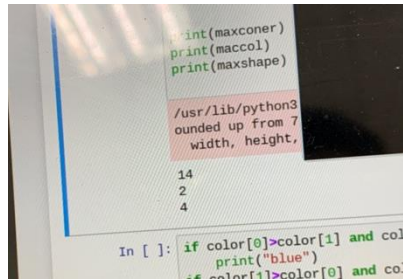
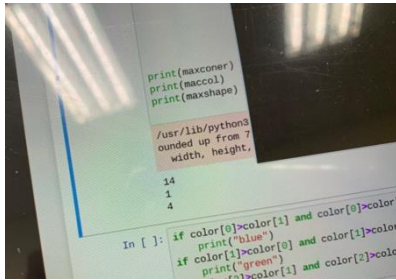
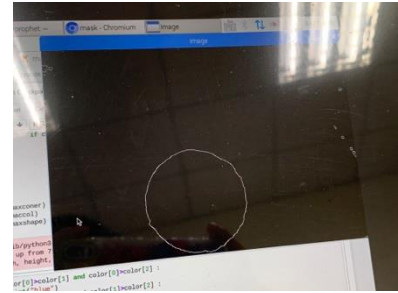
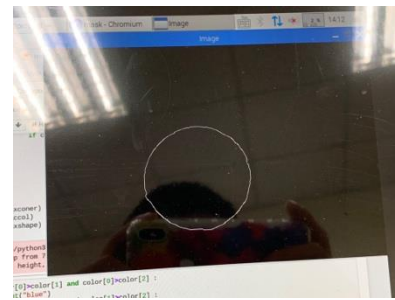
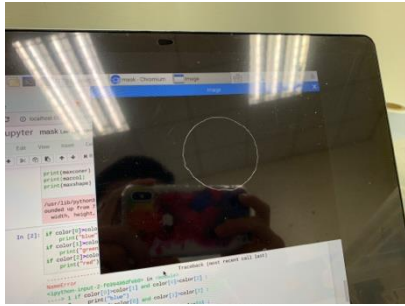


找取邊長最長的，並輸出（圖形頂點數/顏色/形狀）

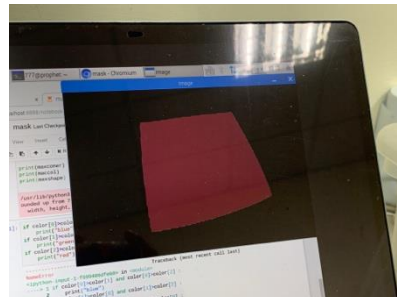
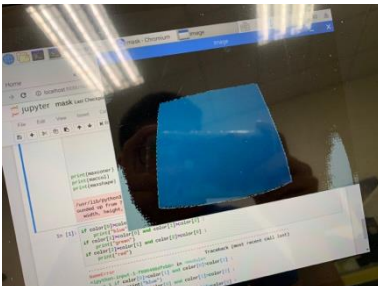
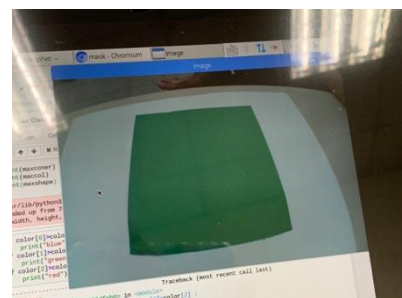
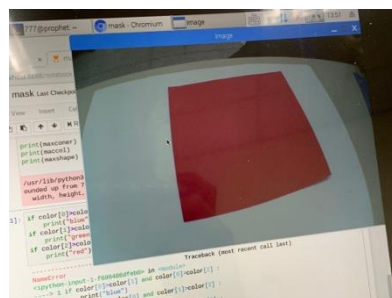
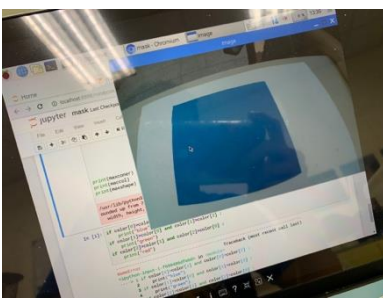
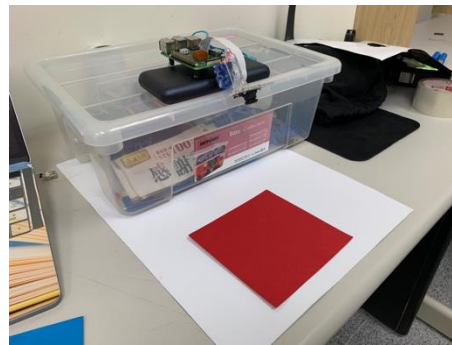


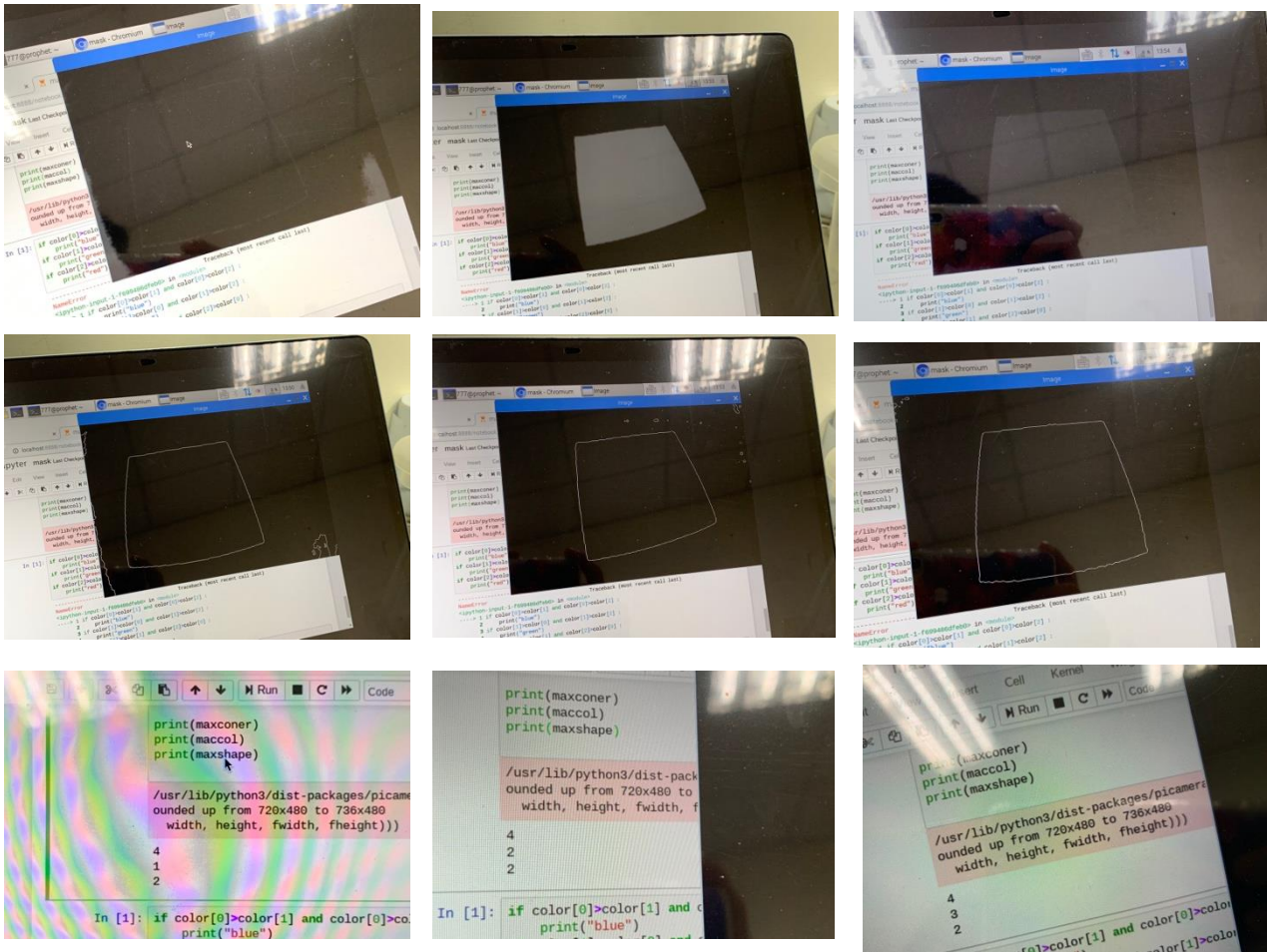
● 以下為圖形試驗結果





● 以下為矩形結果：





而 Arduino 那邊要經由 Uart 接收圖形跟顏色，並在平衡車完成 task 後再次讀取現在最新看到的圖形以及顏色。

程式碼

比賽項目三：rpi : Tesk3OnRpi.py

demo 影片：辨識 demo.mov（顯示辨識一塊紅色矩形的過程）

Arduino: main.ino

實際狀況：

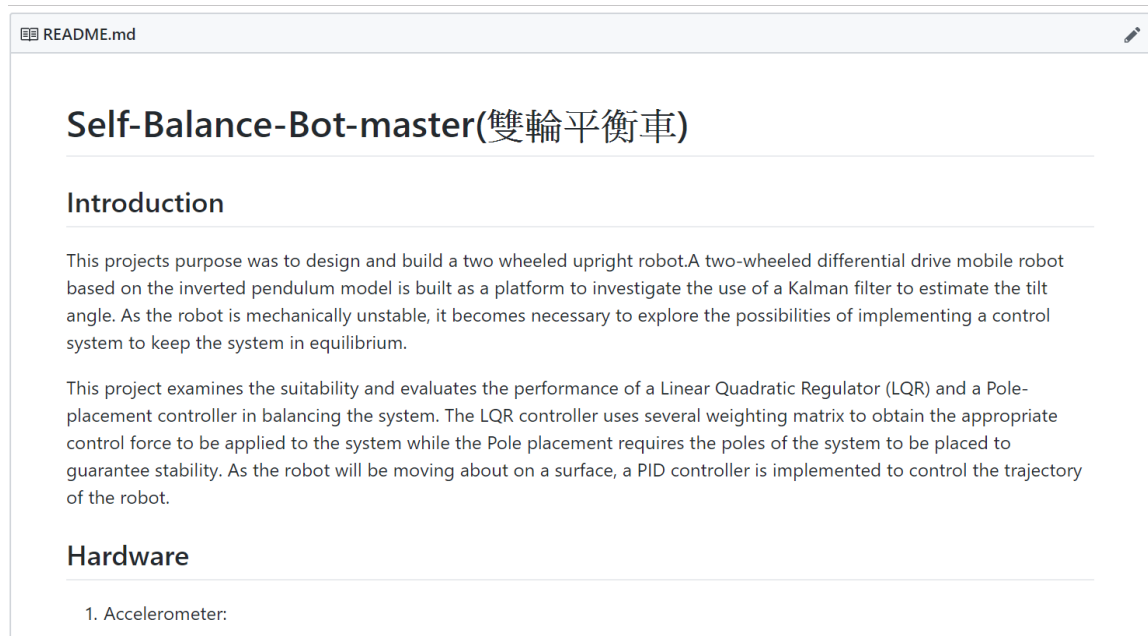
因為多加了一塊板子，所以平衡車的 PID 值需要再做調整才能穩定的平衡。而因為 Rpi 非常耗電，所以我們又多準備了很多電池做備用，可惜的是車子在第一次比賽就把供電的部分跟上面的支撐材摔斷了，手邊沒有焊槍無法立即修復因而沒有成功晉級到複賽（只有贏第一場）。

結論：

這次自控雙輪平衡車的實驗我們接投入了相當大量的時間，甚至到了比賽前幾天皆是熬夜到天明，超怕如果比賽當天車子發生甚麼問題或是做不出來，一切心血皆付諸流水，還好能做出一點屬於我們的小作品心中確實有小小的欣慰。此外，這次前面的進度有點拖到，所以後面花了大量時間研究安裝系統以及設定環境，了解 Python 以及 openCV 的基礎顏色圖形處理，邊緣偵測的參數等等，短時間學習了大量的新東西，雖然最後結果可惜。

然後在團體的專題中為了更正式的將作品呈現，我們也開始學習如何使用 GitHub 平台，包括撰寫 README.md，未來我們仍會繼續在這塊上去著墨。

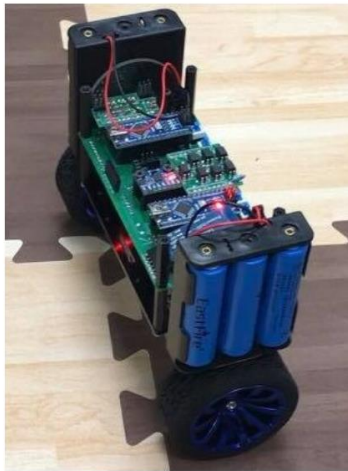
GitHub 連結：<https://github.com/CHI-YU-SUNG/Self-Balance-Bot-master>



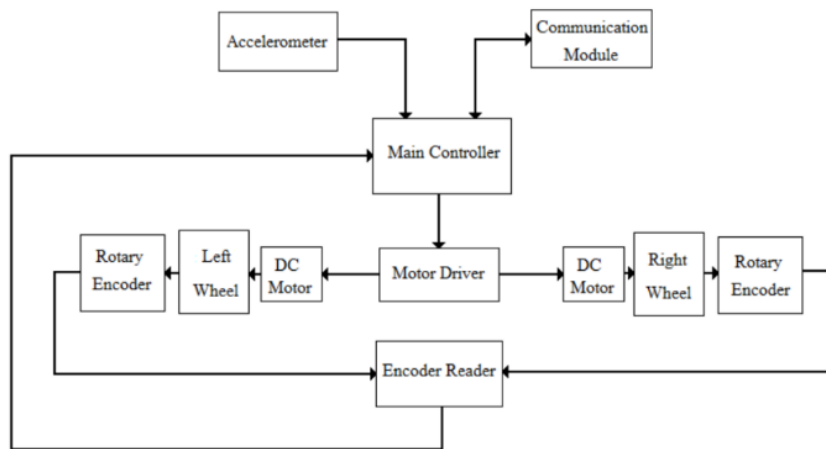
Hardware

1. Accelerometer:
2. Main Controller: Arduino nano
3. Motor Driver:
4. DC Motors:
5. Wheels:
6. Rotary Encoders:

System Mechanical Structure



System Structure



Function

- Position control
- Speed control
- Image recognition

Simulation

matlab

- ID:
- Simulink :

Product

Your support will let me be better

video

You will see in my youtube channel later.