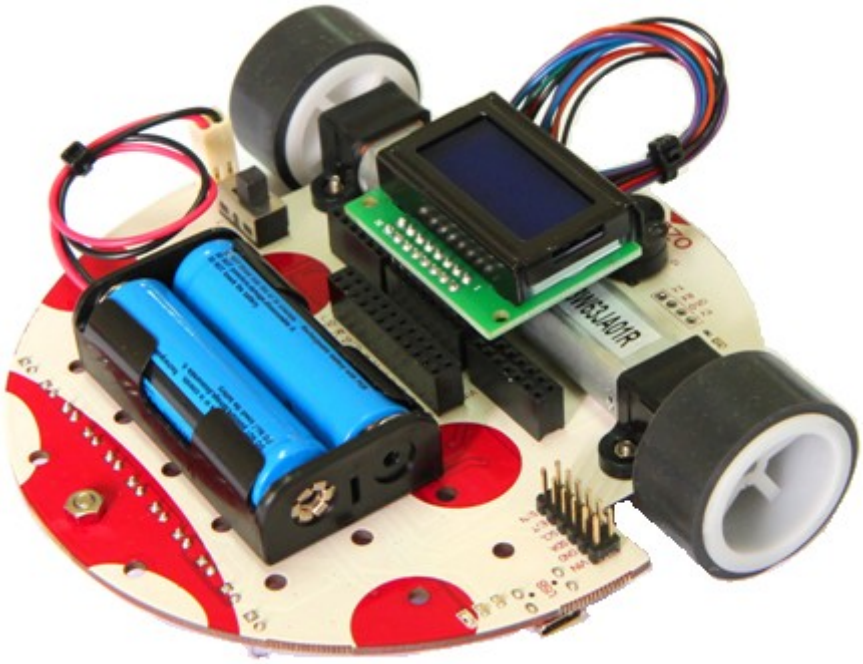
OZBOT™
循跡自走車
使用手冊

版本: 1.0



Passion for innovation

簡介

Ozbot 為利基應用科技開發的小型輪型機器人，採用 ATmega32u4 微控制器為控制核心，具有 32KB flash、2.5KB RAM 與 1KB EEPOM，系統頻率 16MHz。核心部份是與 Ozone (Arduino Leonardo)控制板相容。為提升整體效率，另有搭配 32 位元微控制器作為底層協同控制器。DC 馬達、紅外線循跡感測器、2 x 8 液晶顯示器、LED、蜂鳴器等配件，配合學習範例程式，是一台適合初學者的機器人學習套件。配合不同演算法，作為各種循跡競賽，例如線迷宮電腦鼠競賽。搭配擴充導航板(NAVI Board)可以學習機器人導航功能，可參與機器人自主導航避障競賽。

特性	說明
輸入電壓	7.4V (3.7V 鋰電池 x 2)
尺寸	長 12 cm 寬 13 cm
淨重	540 克 (不含電池重量與電池座)
DIO (5V)	16
循跡 IR 感測器	IR 發射 x6, IR 接收 x5
按鈕	1 reset, 2 使用者按鈕
程式下載	Micro USB
蜂鳴器	1
LED	電源 x1, CmdBUS x3, 使用者 LED x 2
輪直徑／輪寬	30mm/15mm
馬達	18,000 RPM 馬達 x2
減速比	1:21
編碼器	Yes (Hall Sensor)
UART 介面	Yes
CMDBUS 介面	Yes
LCD 顯示器	2x8 (Brightness VR on BZBOT)

OZBOT 同時提供進階擴充導航板(NAVI Board)，內含三軸磁力計、三軸加速度計、三軸陀螺儀，導航板可再外接藍牙模組、Zigbee、可做為進階機器人教學與研發使用或是群體機器人研究平台。

Module OZBot: 必要輸入的指令名稱與參數，以粗底或粗斜體表示，粗體的文字在輸入時請不要更改，粗斜體的文字請自行定義適當格式的參數填入。
在執行 OZBot 指令前，請先於程式開頭定義對應參數，例:

#include <SPI.h>

#include "ozbot.h"

指令格式	指令功能
馬達控制相關指令	
void ForwardL(uint16_t *Speed*);	以 ***Speed*** 輸入的速度值，設定馬達向前轉的速度。可以輸入 0 ~ 1024 間的整數值。其中 L 為左輪馬達，R 為右輪馬達。
void ForwardR(uint16_t *Speed*);	
void ForwardLR(uint16_t *SpeedL,* *uint16_t SpeedR*);	
void BackwardL(uint16_t *Speed*);	以 ***Speed*** 輸入的速度值，設定馬達向後轉的速度。***Speed*** 可以輸入 0 ~ 1024 間的整數值。
void BackwardR(uint16_t *Speed*);	
void BackwardLR(uint16_t *SpeedL,* *uint16_t SpeedR*);	
void StopL(void);	停止指定馬達轉動。
void StopR(void);	
void StopDual(void);	
void BrakeL(void);	快速停止指定馬達的轉動。
void BrakeR(void);	
viod BrakeDual(int16_t void);	
void SetVelL(int16_t *Vel*);	以 ***Vel*** 輸入的速度值，設定馬達轉動的速度。並且以正負值設定馬達的轉向。***Vel*** 可以輸入-1024 ~ 1024 間的整數值。
void SetVelR(int16_t *Vel*);	
void SetVelLR(int16_t *VelL,* int16_t *VelR*);	
void SetVelDual(int16_t *Vel*);	
紅外線感測相關指令	
void SetIrMode(unsigned char *Mode*);	以 Mode 輸入的設定值，設定速度控制時，所要使用的紅外線感測模式。預設值為 0。 Mode 可以輸入 0 或 1。 0: 以數位 IR1~IR3 做 PID 控制。 1: 以類比 IR1~IR3 做 PID 控制。 2: 以數位 IR0~IR4 做 PID 控制。 3: 以類比 IR0~IR4 做 PID 控制。
void GetIr(unsigned char &*IR*);	取得紅外線數位的感測值，儲存於 ***IR*** 參數中。***IR***

	回傳的 bit0~bit4，對應到各個紅外線感測的數位感測值。 ***IR*** 回傳值為 0 ~ 31 間的整數值。
void GetAnalogIr(unsigned char *ID, uint16_t& IR*);	根據 ***ID*** 設定的感測器編號，取得對應紅外線感測器的類比感測值，儲存於 ***IR*** 參數中。 ***IR*** 回傳值為 0 ~ 1023 間的整數值。
void IrCal(unsigned char *Mode*);	以 ***Mode*** 輸入的設定值，啟動紅外線感測器的校正模式。***Mode*** 可以輸入 0 ~ 4 間的整數值。 0: 校正模式開始 10 秒後結束。 1: 校正模式開始 10 秒後結束。 2: 校正模式開始 20 秒後結束。 3: 校正模式開始 30 秒後結束。 4: 校正模式開始 60 秒後結束。
void GetIrCal (unsigned char *ID,* uint16_t& *Min,* uint16_t& *Max*);	取得 ***ID*** 指定的紅外線感測器，校正所得的最大與最小值，存放於 ***Min*** 與 ***Max*** 中。 ***ID*** 可以設定 0 ~ 4 間的整數值。 ***Min*** 與 ***Max*** 會回傳 0 ~ 1023 間的整數值。
void SetIrThreshold(*Rate*);	以 ***Rate*** 輸入的設定值，設定該比例為紅外線感測器，將偵測值設定為高準位的最小值。 ***Rate*** 可以輸入 0 ~ 100 間的整數值。 預設值為 50。
PID 設定與讀取相關指令	
void SetP(uint8_t *Val*);	以 ***Val*** 輸入的設定值，設定 PID 相關參數的數值。
void SetI(uint8_t *Val*);	
void SetD(uint8_t *Val*);	
void SetScalar(uint8_t *Val*);	以 ***Val*** 輸入的設定值，設定 PID Scalar 的數值。
void SetErrScale(*Err1, Err2, Err3, Err4*);	以 ***Err1 ~ Err4*** 輸入的設定值，分別設定各種紅外線感測狀況的誤差值。 預設為： ***Err1***：1 ***Err2***：2 ***Err3***：3 ***Err4***：4
速度控制相關指令	
void SetSpdCtrlL(int16_t *SpdMin,* uint16_t *SpdMax*);	以 ***SpdMin*** 與 ***SpdMax*** 輸入的設定值，分別設定速度控制啟動後的最大速度值與最小速度值。

void SetSpdCtrlR(**int16_t *SpdMin*, int16_t *SpdMax*);**	***SpdMin*** 與 ***SpdMax*** 都可以輸入-1024 ~ 1024 間的整數值，但 ***SpdMax*** 必須大於 ***SpdMin***。
void SetStraight(**int16_t *SpeedL*, int16_t *SpeedR*);**	以 ***SpeedL*** 與 ***SpeedR*** 輸入的設定值，分別設定速度控制啟動後，直線行走時的左右馬達速度值。 ***SpeedL*** 與 ***SpeedR*** 都可以輸入-1024 ~ 1024 間的整數值。
void SpdCtrlOn(uint8_t *Mode*);	以 ***Mode*** 設定的模式，啟動速度控制。 1: 自動搜尋模式。 2: 利用自動搜尋模式所產生的最佳路徑表來走迷宮。
void SpdCtrlOff(void);	關閉速度控制。
void SetCtrlFreq(uint8_t *Period*);	以 ***Period*** 設定的參數值，設定速度控制的間隔時間。單位為 1 ms。
void ArcTurnControlL(**uint8_t *TurnState*,** ***int16_t LeftSpeed*, int16_t *RightSpeed*,** ***uint16_t BlackCount*,** ***uint16_t DelayCount*);**	設定競速圈的左轉轉彎參數 ***TurnState*** 轉向值 0 ~ 20 表示線在 IR1~ IR3 範圍內, 255 則表示不在範圍內 0 則表示 IR1 在線中心 10 則表示 IR2 在線中心 20 則表示 IR3 在線中心 當 IR 偵測到小於 ***TurnState*** 值時, 就表示轉彎完成 ***LeftSpeed*** 左轉時左輪的速度-1024~1024 ***RightSpeed*** 左轉時右輪的速度-1024~1024 ***BlackCount*** *當看到黑線多少距離時, 就開始*左轉 ***DelayCount*** 開始左轉時, 在此距離內不檢查轉向值以防止誤判
void ArcTurnControlR(**uint8_t *TurnState*,** ***int16_t LeftSpeed*, int16_t *RightSpeed*,** ***uint16_t BlackCount*,** ***uint16_t DelayCount*);**	設定競速圈的右轉轉彎參數 ***TurnState*** 轉向值 0 ~ 20 表示線在 IR1~ IR3 範圍內, 255 則表示不在範圍內 0 則表示 IR1 在線中心 10 則表示 IR2 在線中心 20 則表示 IR3 在線中心

	當 IR 偵測到大於 ***TurnState*** 值時，就表示轉彎完成 ***LeftSpeed*** 右轉時左輪的速度-1024~1024 ***RightSpeed*** 右轉時右輪的速度-1024~1024 ***BlackCount*** 當看到黑線多少距離時，就開始右轉 ***DelayCount*** 開始右轉時，在此距離內不檢查轉向值以防止誤判
void RecTurnControlL(**uint8_t *TurnState*,** ***int16_t LeftSpeed*, int16_t *RightSpeed*,** ***uint16_t BlackCount*,** ***uint16_t DelayCount*);**	設定記錄圈的左轉轉彎參數 ***TurnState*** 轉向值 0 ~ 20 表示線在 IR1~ IR3 範圍內, 255 則表示不在範圍內 0 則表示 IR1 在線中心 10 則表示 IR2 在線中心 20 則表示 IR3 在線中心 當 IR 偵測到小於 ***TurnState*** 值時，就表示轉彎完成 ***LeftSpeed*** 左轉時左輪的速度-1024~1024 ***RightSpeed*** 左轉時右輪的速度-1024~1024 ***BlackCount*** 當看到黑線多少距離時，就開始左轉 ***DelayCount*** 開始左轉時，在此距離內不檢查轉向值以防止誤判
void RecTurnControlR(**uint8_t *TurnState*,** ***int16_t LeftSpeed*, int16_t *RightSpeed*,** ***uint16_t BlackCount*,** ***uint16_t DelayCount*);**	設定記錄圈的右轉轉彎參數 ***TurnState*** 轉向值 0 ~ 20 表示線在 IR1~ IR3 範圍內, 255 則表示不在範圍內 0 則表示 IR1 在線中心 10 則表示 IR2 在線中心 20 則表示 IR3 在線中心 當 IR 偵測到大於 ***TurnState*** 值時，就表示轉彎完成 ***LeftSpeed*** 右轉時左輪的速度 -1024~1024 ***RightSpeed*** 右轉時右輪的速度-1024~1024 ***BlackCount*** 當看到黑線多少距離時，就開始右轉 ***DelayCount*** 開始右轉時，在此距離內不檢查轉向值以防止誤判

void SetMotorDeadZone(**int16_t *SpeedL, int16_t SpeedR*);**	以 ***SpeedL, SpeedR*** 輸入的速度值，設定馬達轉動的最低初始速度。
各項設定相關指令	
void SetLineColor(uint8_t *Color*);	以 ***Color*** 設定的參數值，設定軌道的顏色。預設為 0。 ***Color*** 0: 軌道為白色。 1: 軌道為黑色。
馬達回傳脈波相關指令	
void SetTachInR(int16_t *TACH*);	依照 ***TACH***，設定脈波計數的起始數據。
void SetTachInL(int16_t *TACH*);	
void SetTachInLR(**int16_t *TACHL,* int16_t *TACHR*);**	
void TachInR(int16_t& *TACH*);	取得馬達脈波計數數值，儲存於 ***TACH*****。**
void TachInL(int16_t& *TACH*);	
void TachInDual(**int16_t& *TACHL,* int16_t& *TACHR*);**	
軌道紀錄相關指令	
void StartRec(uint8_t ***Mode*****);**	開始紀錄軌道資訊，並根據 ***Mode*** 設定，儲存資料到 FLASH 中。

<table>
<tr><td></td><td>Mode
0: 不儲存資料於 FLASH。
1: 儲存資料於 FLASH。</td></tr>
<tr><td>void StopRec(void);</td><td>停止軌道紀錄。</td></tr>
<tr><td>void GetRecStatus(uint8_t& Status);</td><td>取得軌道紀錄狀態，存於 Status 中。
Status
0: 沒有開始紀錄模式或紀錄結束。
1: 進入紀錄模式。</td></tr>
<tr><td>void GetSecLen(
uint8_t Num,
int16_t& LengthL, int16_t& LengthR);</td><td>取得 Num 指定路段左右輪行駛的距離，分別儲存於 LengthL 與 LengthR。
Num 可以設定 0 ~ 255 間的整數值。
LengthL 與 LengthR 回傳-32768 ~ 32767 間的整數值。</td></tr>
<tr><td>void GetSecDir(
uint8_t Num, uint8_t& Dir);</td><td>取得 Num 指定路段中, 車子所設定轉彎的方向
0: 迴轉，1: 左轉，2: 直行，3: 右轉。</td></tr>
<tr><td>void GetSecCrossroad
(uint8_t Num, uint8_t& Crossroad);</td><td>取得 Num 指定路段中, 車子所遇到的路口型態
Crossroad 路口型態。<table><tr><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>終點</td><td>┐</td><td>｜</td><td>┤</td></tr><tr><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>┌</td><td>┬</td><td>├</td><td>┼</td></tr></table></td></tr>
<tr><td>void GetSecCnt(uint8_t& Cnt);</td><td>取得經過的路口個數，儲存於 Cnt。
Cnt 回傳範圍為 0 ~ 255 間的整數值。</td></tr>
<tr><td>void GetCurSecTach(
int16_t& LengthL, int16_t& LengthR)</td><td>取得最近一次路口(含開始點), 到現在位置的左右輪行駛距離，儲存於 LengthL 與 LengthR 中。
LengthL 與 LengthR 回傳-32768 ~ 32767 間的整數值。
＊需要啟動紀錄模式才有效。</td></tr>
<tr><td colspan="2">迷宮指令</td></tr>
<tr><td>void SearchPath (uint8_t Mode);</td><td>Mode 設定自動搜尋模式所選用的方法。
0: 使用左手定則。
1: 使用右手定則。</td></tr>
<tr><td>void GetTotalPathCnt(uint8_t& Cnt);</td><td>回傳自動搜尋模式的路徑表格大小。
Cnt = 0~255。</td></tr>
<tr><td>void TotalPath(
uint8_t& Point,
uint8_t&Crossroad,
uint8_t& Dir,</td><td>取得自動搜尋模式中，Num 指定路段的資料。
其中
Point 回傳過程中經過且不重覆的路口數。
Crossroad 路口型態。</td></tr>
</table>

<table>
<tr><td>int16_t& LSteps, int16_t& RSteps) ;</td><td>
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>終點</td><td>┐</td><td>│</td><td>┤</td></tr>
<tr><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>┌</td><td>┬</td><td>├</td><td>┼</td></tr>
</table>
Dir 走至該點所選擇的轉向。
0: 迴轉，1: 左轉，2: 直行，3: 右轉。
LSteps(-32768 ~32767) 在指定的路段中，左輪所走的距離。
RSteps(-32768 ~32767) 在指定的路段中，右輪所走的距離。
</td></tr>
<tr><td>GetOptiPathCnt(uint8_t& Cnt);</td><td>回傳最佳路徑表格大小。
Cnt = 0~255</td></tr>
<tr><td>OptimumPath(uint8_t Num,
uint8_t& Point,
uint8_t&Crossroad,
uint8_t& Dir,
int16_t& LSteps, int16_t& RSteps) ;</td><td>取得最佳路徑模式中，Num 指定路段的資料。
其中
Point 回傳過程中經過且不重覆的路口數。
Crossroad 路口型態。
<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>終點</td><td>┐</td><td>│</td><td>┤</td></tr>
<tr><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>┌</td><td>┬</td><td>├</td><td>┼</td></tr>
</table>
Dir 走至該路口所選擇的轉向。
0: 迴轉，1: 左轉，2: 直行，3: 右轉。
LSteps(-32768 ~32767) 在指定的路段中，左輪所走的距離。
RSteps(-32768 ~32767) 在指定的路段中，右輪所走的距離。</td></tr>
<tr><td colspan="2">其他設定指令</td></tr>
<tr><td>void Beep(void);</td><td>啟動 Buzzer 播放 0.1 秒。</td></tr>
<tr><td>void AutoBeep(uint8_t Mode);</td><td>根據 Mode 設定，自動啟動或關閉 Buzzer 播放。
Mode
0: 關閉自動撥放功能。
1: 啟動自動撥放功能，經過路口就會啟動 Buzzer 播放 0.1 秒。
預設為 0。</td></tr>
<tr><td>void SetCrossCount(uint16_t Count);</td><td>以 Count 設定交叉軌道的判定距離。</td></tr>
<tr><td>void LowBatteryAlarmOn(void);</td><td>啟動自動電壓偵測，當電壓過低時啟動蜂鳴器。
預設啟動。</td></tr>
</table>

void LowBatteryAlarmOff (void);	關閉自動電壓偵測。
uint8_t *Status* = CheckLowBattery(void);	將電壓狀態回傳至 ***Status*** 中。 0：電壓狀態正常。 1：電壓狀態過低。
移動游標相關指令	
void CarriageReturn(void); void CR(void);	讓游標移動到下一列
void CursorColumn(uint8_t ***Col*****); void CursorCol(**uint8_t ***Col*****);**	將游標移動至 ***Col*** 指定的行，***Col*** 請輸入 1~8 之間的整數值
void CursorDown(void);	將游標下移一列
void CursorLeft(void);	將游標向左移一個字元
void CursorRC(uint8_t ***Row*****,** uint8_t ***Col*****);**	將游標移動到 ***Row*** 所指定的列與 ***Col*** 所指定的行，***Row*** 請輸入 1~2 之間的整數值，***Col*** 請輸入 1~8 之間的整數值
void CursorRight(void);	將游標向右移一個字元
void CursorRow(uint8_t ***Row*****);**	將游標移動至 ***Row*** 指定的列，***Row*** 請輸入 1 到 2 之間的整數值
void CursorUp(void);	將游標上移一列
void Home(void);	將游標移動到第一行第一列
清除顯示相關指令	
void Clear(void);	清除畫面上所有顯示的字元
void ClearEOL(void);	清除由游標所在位置開始，到列尾的所有字元
void ClearEOS(void);	清除由游標所在位置開始，到螢幕最後顯示的所有字元
顯示字元相關指令	
void Display(*Parameter*);	根據 ***Parameter*** 參數形式，如果是 **String** 會直接顯示字串，其它數值則以十進制顯示
各種設定相關指令	
void BacklightOff(void);	關閉背光
void BacklightOn(uint8_t ***Time*****);**	以 ***Time*** 值設定背光要點亮的時間，若設為 0 則恆亮，***Time*** 請輸入 0~255 之間的整數值
void CursorBlinkOff(void);	停止游標閃爍
void CursorBlinkOn(void);	讓游標開始閃爍
void CursorOff(void);	關閉游標顯示
void CursorOn(void);	讓游標顯示於螢幕
void DisplayOff(void);	關閉螢幕顯示
void DisplayOn(void);	開啟螢幕顯示

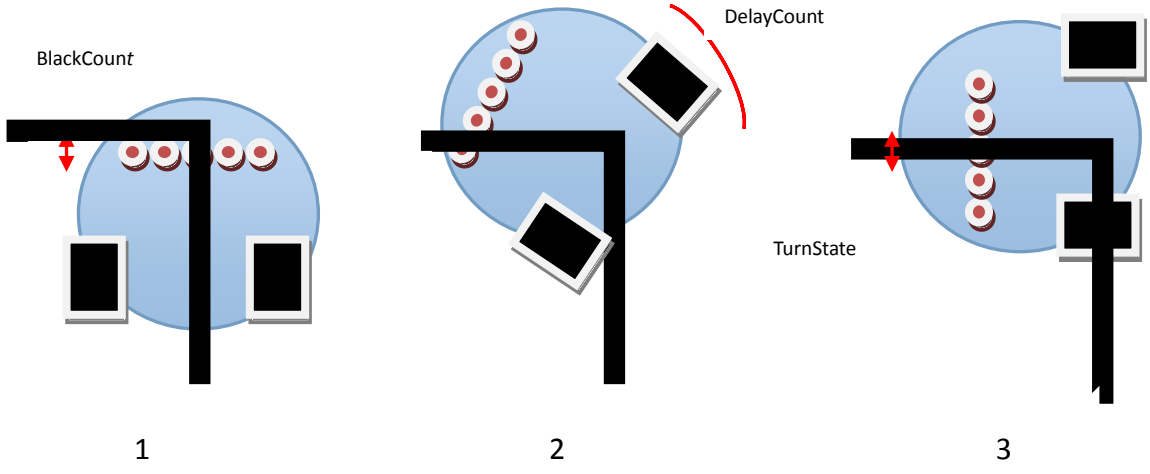
以下指令僅在安裝導航版(NAVI Board, 選配) 時才能使用	
Sonar 指令	
void Ranging(uint8_t *Num*);	執行指定的第 ***Num*** 個超音波發射與接收。 ***Num*** = 0 ~ 4 啟動相對應的超音波。 = 5 啟動所有超音波。
void RepeatRanging(void);	重複(週期性)執行所有的超音波偵測。
void StopRanging(void);	停止發射超音波。
void GetDistance(uint8_t&*Distance0*,uint8_t& *Distance1*, uint8_t&*Distance2*,uint8_t& *Distance3*, uint8_t& *Distance4*);	取得 5 個超音波的距離。 ***Distance0 ~ Distance4*** 範圍 0~255, 單位 cm ***255:*** 表示超過 255cm
Compass 指令	
void GetCompass(uint16_t& *Angle*);	取得由磁北與 X 軸向順時針方向的夾角，以度為單位，放於 Angle 中，Angle 的範圍在 0~359
void GetCompX(int16_t& *X*); void GetCompY(int16_t& *Y*); void GetCompZ(int16_t& *Z*);	取得 Compass 讀值
void GetCompDevAngle(uint16_t& *Angle*);	取得與設定為基準方位差異的偏向角。這個指令把 **SetCompCurrentTargetAngle/SetCompTargetAngle** 的角度值，設定為基準方位，並回傳現在量測到的角度，與基準方 向的偏向角，以度為單位，儲存在 ***Angle*** 中， 如果現在量測的方位，在基準方位逆時針方 向 180 度內，回傳值就是正的，如果現在角 度在基準角的順時針方向 179 度內，回傳值 就是負的，***Angle*** 回傳的範圍為 ***-179 ~ 180***
void SetCompCurrentTargetAngle(void);	設定目前角度為基準方位角
void SetCompTargetAngle(uint16_t Angle);	設定基準方位角
void CompassCal(uint8_t *Time*);	根據輸入的 Time 值，設定模組執行校正的時間。 Time 可以設定為 0~4 等五個不同的時間: Time = 0 執行校正 5 秒 Time = 1 執行校正 10 秒 Time = 2 執行校正 20 秒

	Time = 3　　執行校正 30 秒 Time = 4　　執行校正 60 秒
Accelerometer 指令	
void SetAccMode(*Mode*);	Mode = 0,　2G = 1,　4G = 2,　8G = 3,　16G 預設值 2
void GetAccX(int16_t& *X*); **void GetAccY(int16_t& *Y*);** **void GetAccZ(int16_t& *Z*);** **void GetAccXYZ(** **int16_t& *X,* int16_t& *Y,* int16_t& *Z*);**	取得 Accelerometer 讀值
void GetAccMaxMinX(**int16_t& *Max,* int16_t& *Min*);** **void GetAccMaxMinY(** **int16_t& *Max,* int16_t& *Min*);** **void GetAccMaxMinZ(** **int16_t& *Max,* int16_t& *Min*);**	取得期間內三軸 Accelerometer 最大值和最小值，且執行此命令後，最大值和最小值會自動清除為 0
void AccCal (void);	設定 Accelerometer 模組執行校正，校正時間 3 秒。
Gyro 指令	
void SetGyroMode(uint8_t *Mode*);	Mode = 0,　250 °/SEC = 1,　500 °/SEC = 2,　1000 ° SEC = 3,　2000 °/SEC 預設值 0
void GetGyroX(int16_t& *X*); **void GetGyroY(int16_t& *Y*);** **void GetGyroZ(int16_t& *Z*);** **void GetGyroXYZ(** **int16_t& *X,* int16_t& *Y,* int16_t& *Z*);**	取得目前的三軸 Gyro 值
void GetGyroIntegX(int32_t& *X*); **void GetGyroIntegY(int32_t& *Y*);** **void GetGyroIntegZ(int32_t& *Z*);**	取得期間內三軸 Gyro 積分值，且執行此命令後，積分值會自動清除為 0

void GetGyroMaxMinX(**int16_t& *Max,* int16_t& *Min*);** **void GetGyroMaxMinY(** **int16_t& *Max,* int16_t& *Min*);** **void GetGyroMaxMinZ(** **int16_t& *Max,* int16_t& *Min*);**	取得期間內三軸 Gyro 最大值和最小值，且執行此命令後，最大值和最小值會自動清除為 0
void GyroCal(void);	設定 gyro 執行校正，校正時間 3 秒。

紀錄圈

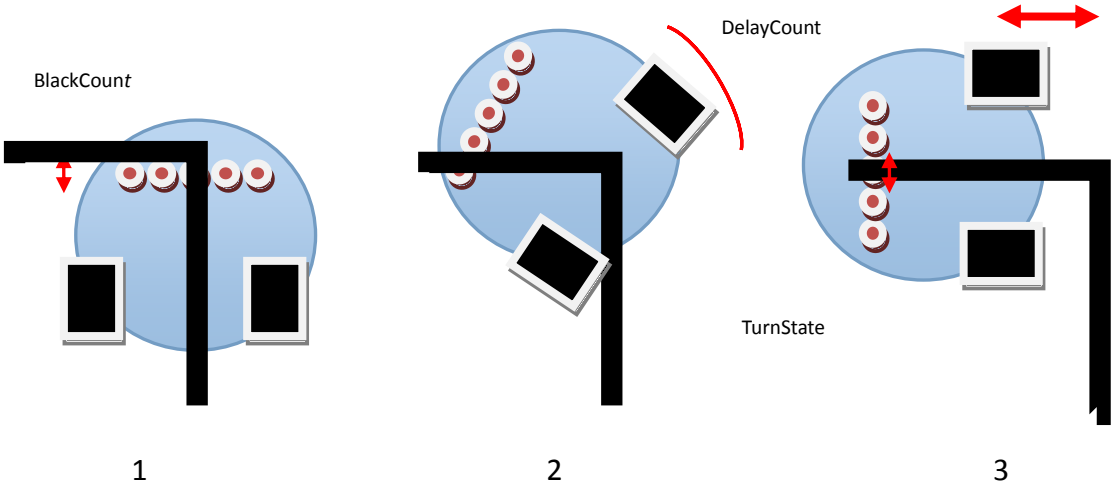
原地 90 度轉彎



RecTurnControlL/ RecTurnControlR

競速圈

以一輪為圓心轉彎，因此轉彎完成後，車子會較紀錄圈超前一小段距離. 而遇到轉彎時，也會比紀錄圈提前轉彎. 因此寫程式時，需考慮到這些差異.



ArcTurnControlL/ArcTurnControlR