Gestione avanzata dei motori

Ci sono un certo numero di comandi addizionali per la gestione dei motori che possiamo utilizzare per gestirli meglio. In questo capitolo parleremo dei comandi: **ResetTachoCount**, **Coast (Float)**, **OnFwdReg**, **OnRevReg**, **OnFwdSync**, **OnRevSync**, **RotateMotor**, **RotateMotorEx**, e dei concetti basilari del **PID**.

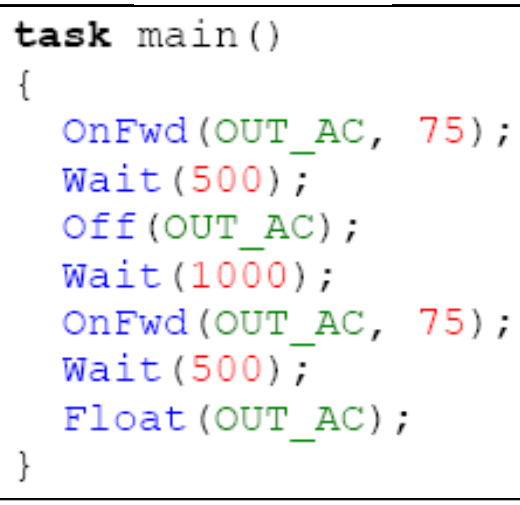
Arresto non brusco

Se utilizzate il comando **Off()** il servomotore si arresta di colpo, arrestando e bloccando il proprio albero. E' possibile fermare il motore più dolcemente, senza usare il freno. Per ottenere ciò, usate i comandi **Float()** oppure **Coast()** (indifferentemente).

L'uso di questi comandi toglie semplicemente la corrente ai motori. L'esempio seguente ferma i motori in entrambi i modi: prima bruscamente , col comando **Off()**, poi gentilmente.

Fate caso: anche se per un robot così piccolo la differnza è piccola, c'è e si nota. Nel caso di robot di maggiori dimensioni,la diferenza aumenta di molto.

Testo copiabile	Screenshot del BricxCC

task main()
{
OnFwd(OUT_AC, 75);
Wait(500);
Off(OUT_AC);
Wait(1000);
OnFwd(OUT_AC, 75);
Wait(500);
Float(OUT_AC);
}
	

Comandi avanzati

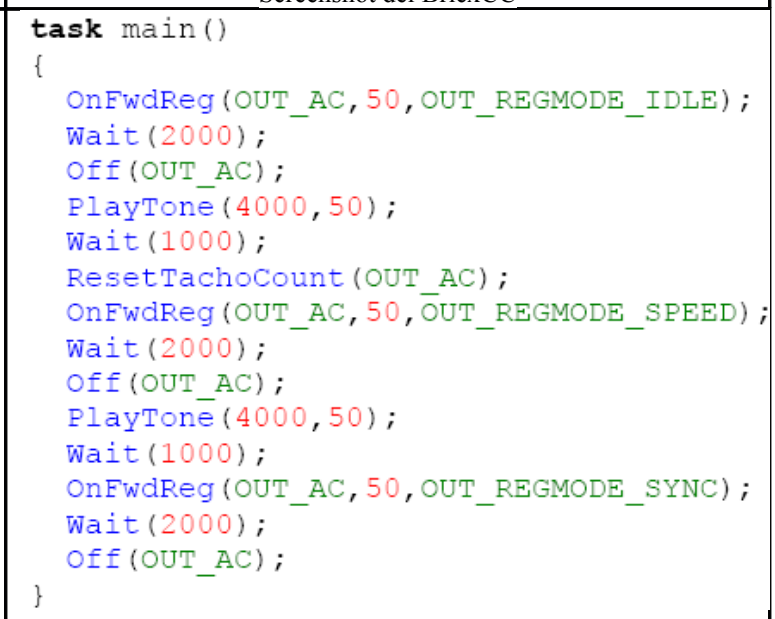
I comandi **OnFwd()** e **OnRev()** sono i comandi più semplici per muovere i motori. I servomotori dell' NXT contengono un encoder che permette di controllare con buona precisione la rotazione dell' albero e la velocità;

Il Firmware dell' NXT implementa un controller **PID (Proportional Integrative Derivative)** a ciclo chiuso per controllare i motori, utilizzando la lettura dei dati degli encoder. Se desiderate che il robot vada perfettamente diritto, potete utilizzare una sincronizzazione che fa muovere assieme una coppia di motori, e fa sì che uno dei due "aspetti" l'altro in caso uno dei due vanga rallentato oppure bloccato; in modo simile a questo, potete utilizzare una coppia di motori per far sì che il robot sterzi a destra, a sinistra, o routi sul posto, ma mantenendo i motori sincronizzati.

Ci sono un sacco di comandi per scatenare la potenza dei nostri servomotori!
OnFwdReg(‘ports',‘speed',‘regmode') fa ruotare i motori specificati alla porta **(ports)** alla velocità **(speed)** applicando il metodo di sincronia **(regmode)** che può essere **OUT_REGMODE_IDLE**, **OUT_REGMODE_SPEED**or **OUT_REGMODE_SYNC**.

Se scegliamo **IDLE**, non verrà utilizzata la regolazione **PID**; se scegliamo **SPEED**, l' NXT manterrà costante la velocità del motore in ogni condizione (ad esempio se aumentiamo il carico del robot o la pendnza che esso supera), infine, se scegliamo **SYNC**, la coppia di motori selezionata tramite **ports** si muoverà in sincronia come spiegato in precedenza. **OnRevReg()** si comporta nello stesso modo, ma facendo girare i motori al contrario.

Testo copiabile	Screenshot del BricxCC

task main()
{
OnFwdReg(OUT_AC,50,OUT_REGMODE_IDLE);
Wait(2000);
Off(OUT_AC);
PlayTone(4000,50);
Wait(1000);
ResetTachoCount(OUT_AC);
OnFwdReg(OUT_AC,50,OUT_REGMODE_SPEED);
Wait(2000);
Off(OUT_AC);
PlayTone(4000,50);
Wait(1000);
OnFwdReg(OUT_AC,50,OUT_REGMODE_SYNC);
Wait(2000);
Off(OUT_AC);
}
	
	task main()
	{
	OnFwdReg(OUT_AC,50,OUT_REGMODE_IDLE);
	Wait(2000);
	Off(OUT_AC);
	PlayTone(4000,50);
	Wait(1000);
	ResetTachoCount(OUT_AC);
	OnFwdReg(OUT_AC,50,OUT_REGMODE_SPEED);
	Wait(2000);
	Off(OUT_AC);
	PlayTone(4000,50);
	Wait(1000);
	OnFwdReg(OUT_AC,50,OUT_REGMODE_SYNC);
	Wait(2000);
	Off(OUT_AC);
	}

Questo programma mostra comportamenti differenti del robot in base alla modalità di gestione dei motori selezionata.

Se cercate di fermare le ruote tenendo il robot in mano:

- Il primo comportamento **(IDLE mode)**: fermando una ruota non accde assolutamente nulla.
- Nella seconda modalità **(SPEED mode)** cercando di rallentare una ruota il robot aumenta la potenza applicatale per neutralizzare l'interferenza esterna e mantenere la velocità invariata.
- Nell' ultima modalità **(SYNC mode)** arrestando una ruota si ferma anche l'altra, in attesa che quella

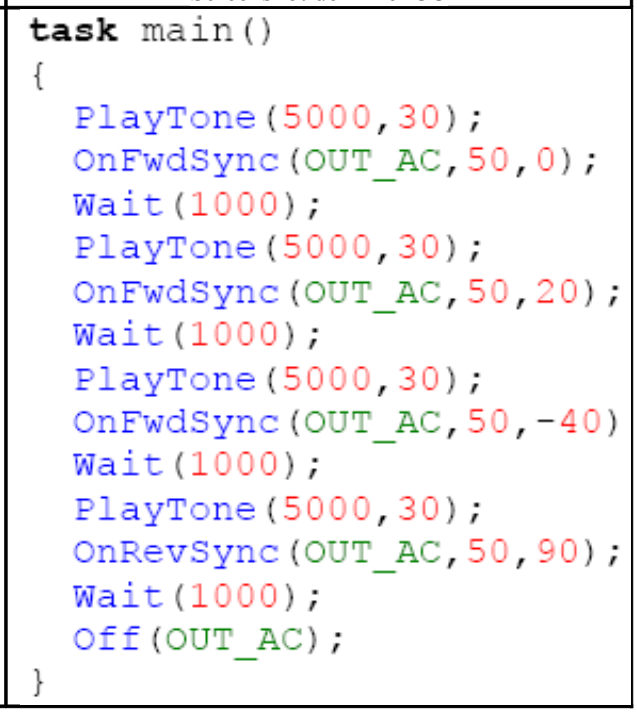
bloccata possa ricominciare a girare.

-

OnFwdSync(‘ports',‘speed',‘turnpct') è equivalente al comando **OnFwdReg()** in modalità **SYNC**, ma consente di specificare "la percentuale di sterzata" tramite il parametro "turnpct" (da 0 a 100). **OnRevSync()** è lo stesso di prima, ma ovviamente fa andare i motori al contrario.

Il programma seguente esemplifica il funzionamento dei comandi ora menzionati: provate a cambiare la "percentuale di sterzata" ed osservate che cosa accade.

Testo copiabile	Screenshot del BricxCC

task main()
{
PlayTone(5000,30);
OnFwdSync(OUT_AC,50,0);
Wait(1000);
PlayTone(5000,30);
OnFwdSync(OUT_AC,50,20);
Wait(1000);
PlayTone(5000,30);
OnFwdSync(OUT_AC,50,-40);
Wait(1000);
PlayTone(5000,30);
OnRevSync(OUT_AC,50,90);
Wait(1000);
Off(OUT_AC);
}
	

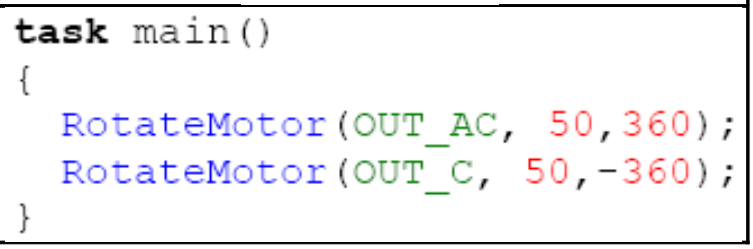
Per finire, i motori possono essere fatti ruotare per un limitato numero di gradi (ricordate che un giro completo equivale a 360 gradi).

In tutti i comandi seguenti, potete agire sulla direzione del motore variando sia il segno della velocità che il segno dell' angolazione.

Così: se l' angolo e la velocità hanno lo stesso segno, il motore girerà in avanti, e se hanno invece segno opposto il motore girerà indietro.

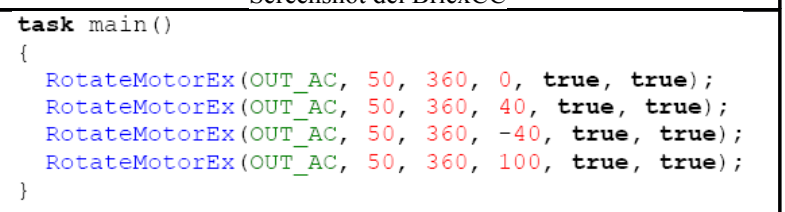
RotateMotor(‘ports',‘speed',‘degrees') ruota il perno del motore specificato da **(ports)** di un angolo specificato da **(degrees)** alla velocità specificata da **(speed)** (da 0 a 100).

Testo copiabile	Screenshot del BricxCC

task main()
{
RotateMotor(OUT_AC, 50,360);
RotateMotor(OUT_C, 50,-360);
}
	

RotateMotorEx(‘ports',‘speed',‘degrees',‘turnpct',‘sync', 'stop')è un' estensione del comando precedente, che permette di sincronizzare due motori (ad esempio OUT_AC) specificando una "percentuale di sterzata" **(turnpct)** (da 0 a 100) ed un flag booleano **(sync)** (che può essere vero o falso). Permete inoltre di specificere se il motore deve frenare il proprio asse di rotazione al termine della rotazione desiderata tramite il flag booleano **(stop)**.

Testo copiabile	Screenshot del BricxCC

task main()
{
RotateMotorEx(OUT_AC, 50, 360, 0, true, true);
RotateMotorEx(OUT_AC, 50, 360, 40, true, true);
RotateMotorEx(OUT_AC, 50, 360, -40, true, true);
RotateMotorEx(OUT_AC, 50, 360, 100, true, true);
}
	

PID control

Il firmware NXT implementa un controller **PID (proportional integrative derivative)** per regolare la posizione e la velocità dei servomotori con precisione. Questo tipo di controller è uno dei più efficaci e più semplici controller a ciclo chiuso esistenti, e viene utilzzato molto spesso. In parole povere funziona così (verifica traduzione):

Il programma dà al controller un punto da raggiungere. Mette in moto i motori col comando **U(t)**, misurando la sua posizione **Y(t)** con l'encoder integrato e calcolando l'errore **e(t)=R8(t) -Y(t)**: in questo senso lo definiamo "closed loop controller" (controller a ciclo chiuso),poichè la posizione di output**Y(t)** viene riportata all' input del controller per fare il calcolo dell' errore.

Il controller trasforma l'errore **E(t)** nel comando **U(t)** in questo modo: **U(t) = P(t) + I(t) + D(t)**, dove **P(t) = KP·E(t)**, **I(t) = KI·(I(t–1) + E(t))** e **D(t) = KD·(E(t) – E(t –1))**.

Può sembrare difficile a prima vista, ma lasciate che vi spieghi il meccanismo. Il comando è la somma di tre contributi, la componente proporzionale **P(t)**, la componente integrata **I(t)** e la componente derivata **D(t)**.

P(t) rende il controller veloce nella risposta, ma non assicura un errore nullo all' eqilibrio;

I(t) dà "memoria" al controller, nel senso che tiene traccia degli errori accumulati e tende a compensarli, garantendo all' equilibrio un errore nullo.

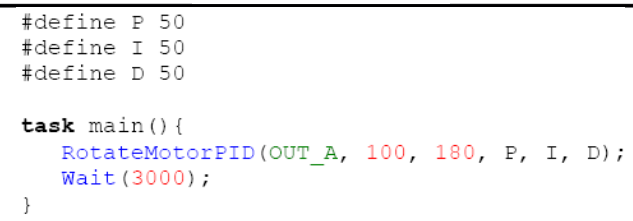
D(t) dà "previsione futura" al controller (così come la derivata in matematica), aumentando la velocità di risposta.

So bene che il concetto può risultare ancora confuso, considerate che sull' argomento sono stati scritti libri accademici!

Ma possiamo sperimentare dal vivo, con il nostro NXT.

Il più semplice programma per fissare il concetto in memoria è il seguente:

Testo copiabile	Screenshot del BricxCC

#define P 50
#define I 50
#define D 50
task main()
{
RotateMotorPID(OUT_A, 100, 180, P, I, D);
Wait(3000);
}
	

Il comando **RotateMotorPID(port,speed, angle, Pgain,Igain,Dgain)** fa muovere i motori con dei settaggi del **PID** differenti da quelli di default.

Proviamo coi settaggi seguenti:

- (50,0,0): il motore non ruota esattamente di 180°, visto che rimane un errore non compensato.
-
- (0,x,x): senza la parte proporzionale , l' errore è enorme.
-
- (40,40,0): con questo settaggio l'albero del motore si muove troppo in avanti, e si deve correggere l'errore facendolo tornare indietro.
-
- (40,40,90):buona precisione ma maggiore lentezza.
-
- (40,40,200): il perno di rotazione del motore oscilla: la componente derivata è troppo elevata.
-

Provate altri settggi, per scoprire che effetto hanno sul funzionamento del motore.