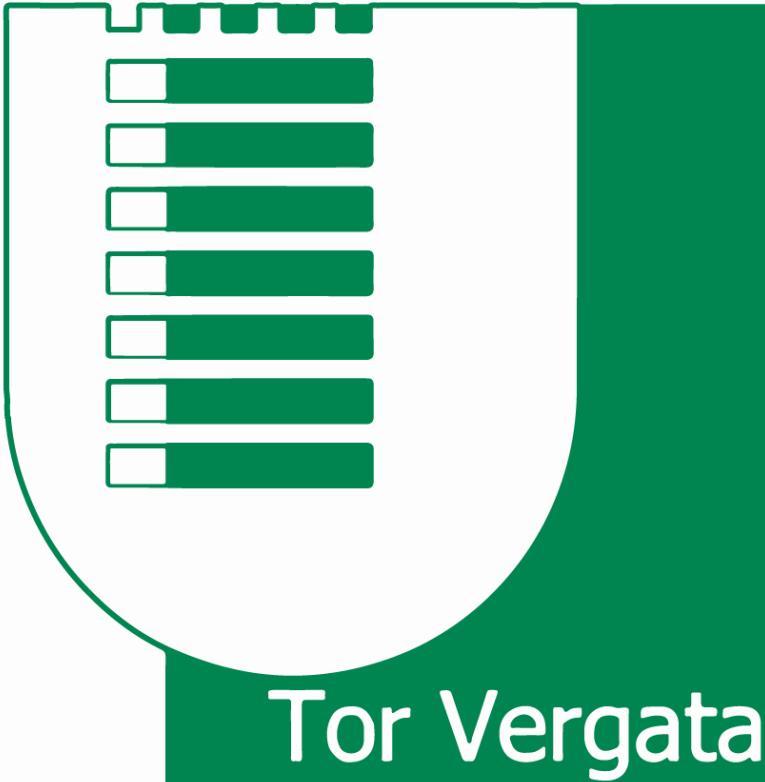


Università di Roma



Controlli Automatici

A.A. 2018-2019

Robot Antropomorfo

Alfano Emanuele

Angeloni Ilaria

Antonini Davide

Cosentini Alessandro

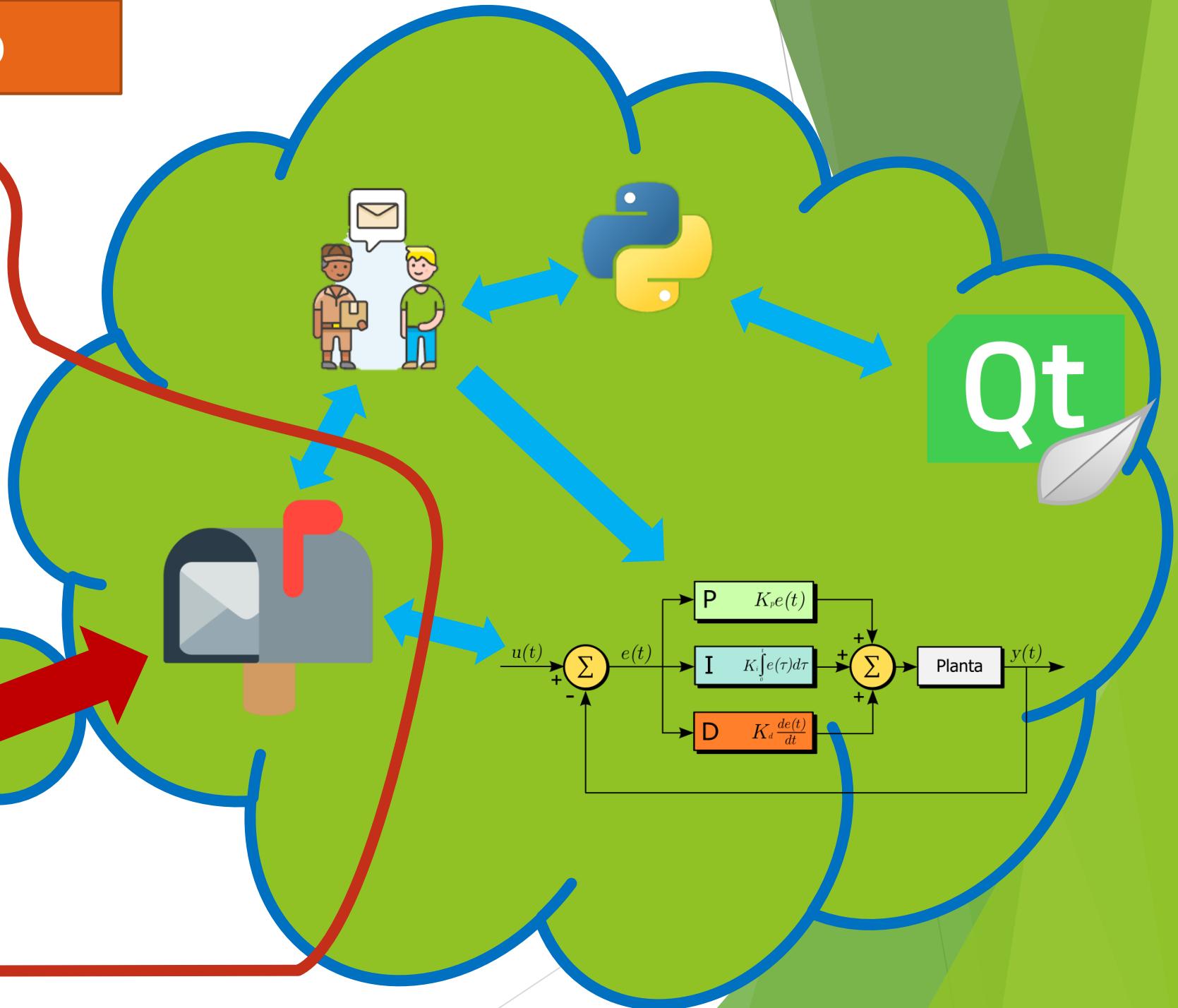
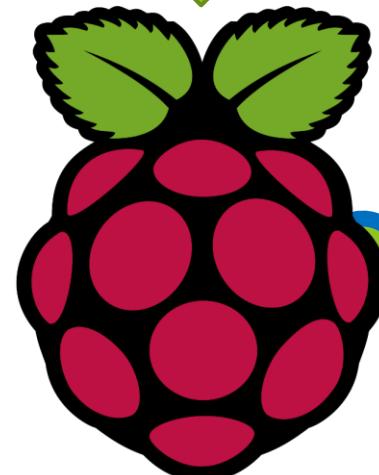
Del Papa Nicolò

Di Vincenzo Fabio

Ferrò Francesco

Menichelli Alberto

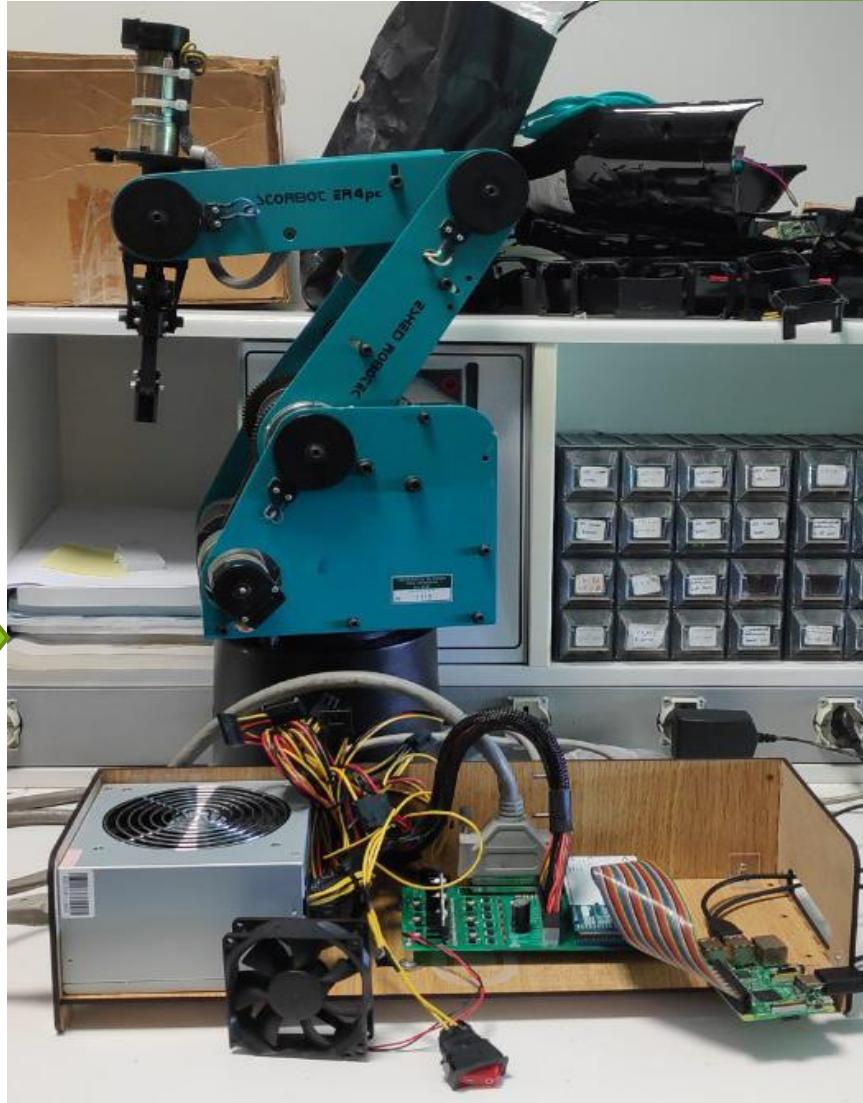
Struttura del progetto



Cosa abbiamo fatto:



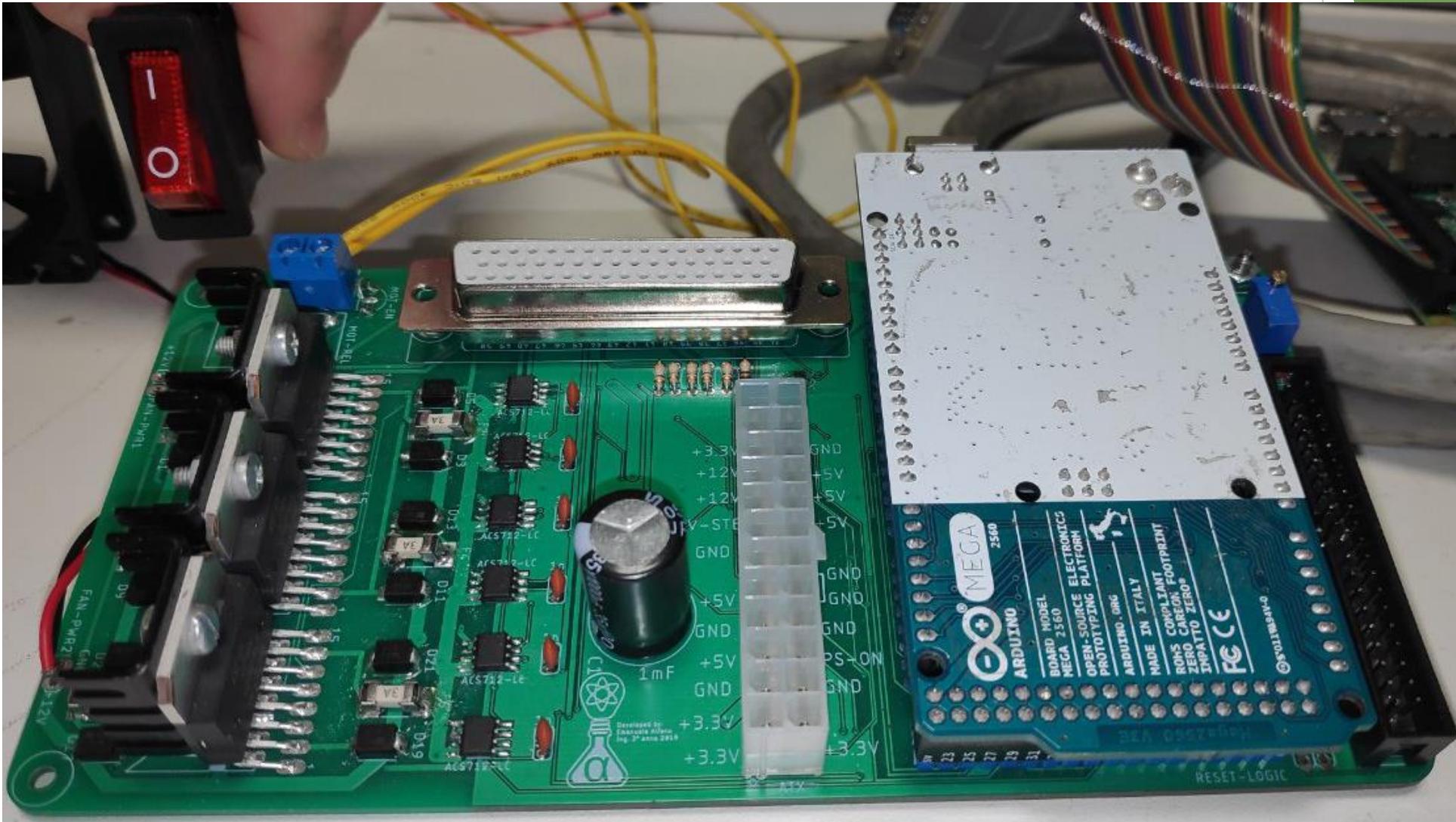
Abiamo
creato



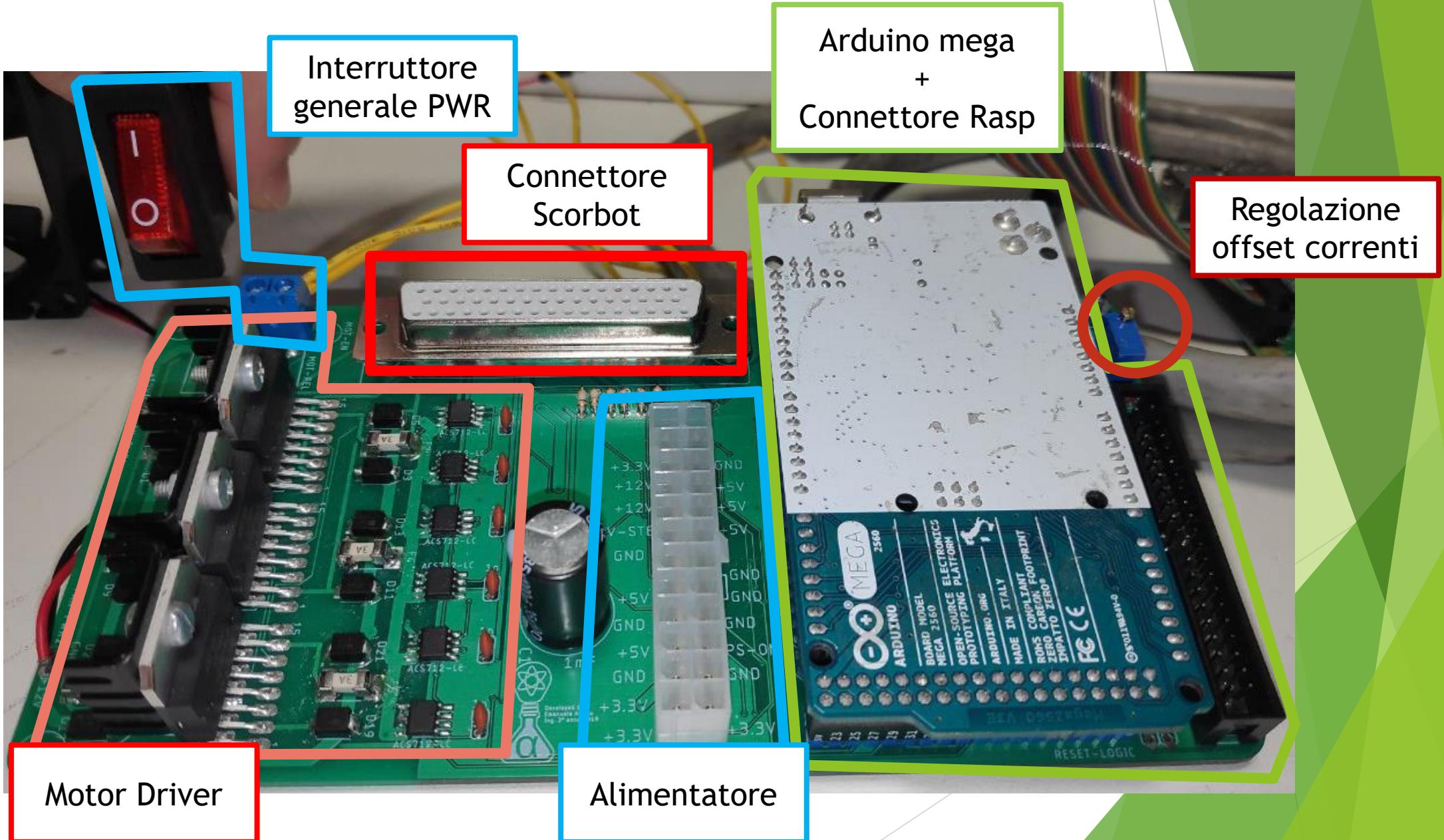
- Alimentatore ingombrante
- Controllo basico
- Lavoro non parallelo
- Potenze limitate

- Tutto il sistema grande quanto il vecchio alimentatore
- Controllo parallelo dei motori e sensori
- Encoder senza perdita dati
- Salvataggio impostazioni sul driver

Hardware & Firmware



Hardware & Firmware



Alimentatore



Quicksilver Pinout

GND Pin 11	[Color Key: Black]	Pin 22 +25V TRKL
+12V Pin 10	[Color Key: Yellow]	Pin 21 +5V
+25V Pin 9	[Color Key: Yellow]	Pin 20 +5V
+3.3V Pin 8	[Color Key: Orange]	Pin 19 GND
GND Pin 7	[Color Key: Black]	Pin 18 GND
+5V Pin 6	[Color Key: Red]	Pin 17 GND
GND Pin 5	[Color Key: Black]	Pin 16 GND
+5V Pin 4	[Color Key: Red]	Pin 15 [Power On]
GND Pin 3	[Color Key: Black]	Pin 14 GND
+3.3V Pin 2	[Color Key: Orange]	Pin 13 -12V
+3.3V Pin 1	[Color Key: Orange]	Pin 12 +3.3V
+12V Pin 2	[Color Key: Yellow]	Pin 4 GND
+12V Pin 1	[Color Key: Yellow]	Pin 3 GND

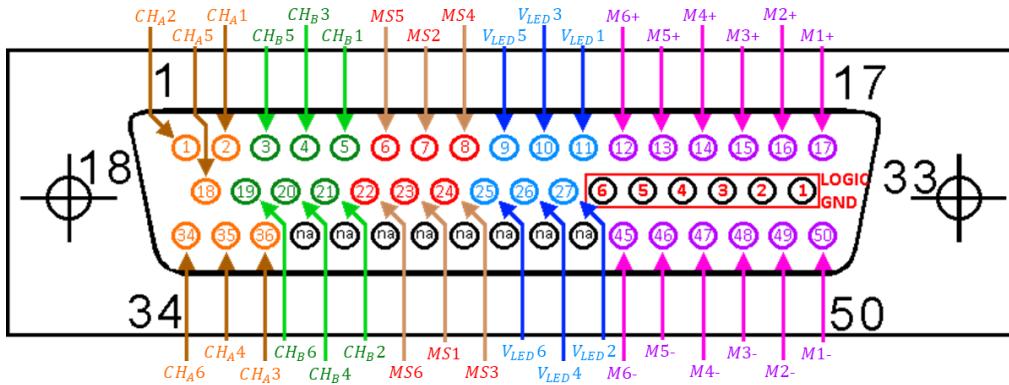
È in grado di alimentare da solo:

- Motori Scrobot
- Elettronica di controllo
- Raspberry Pi

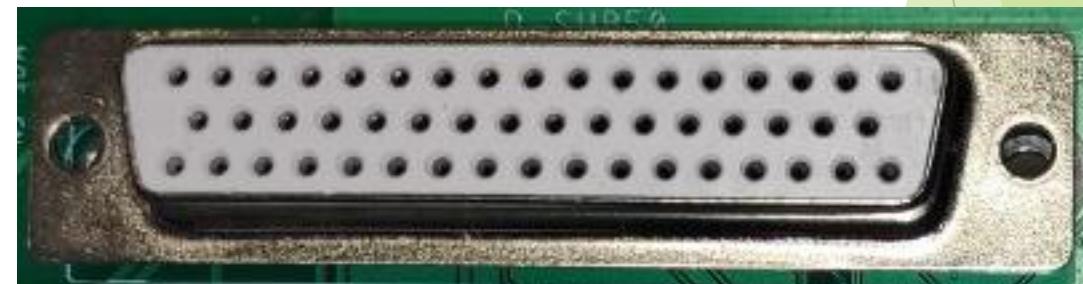
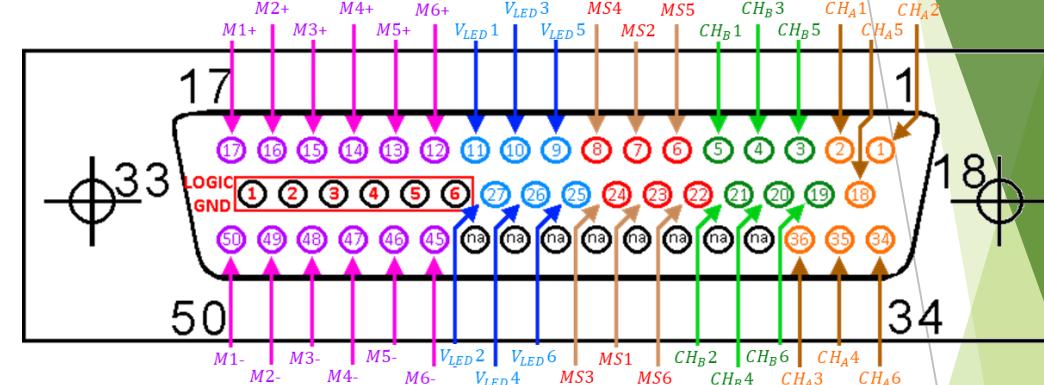
Un ALL-IN-ONE insomma

Connettore D-Sub50

Frontale Connettore Maschio

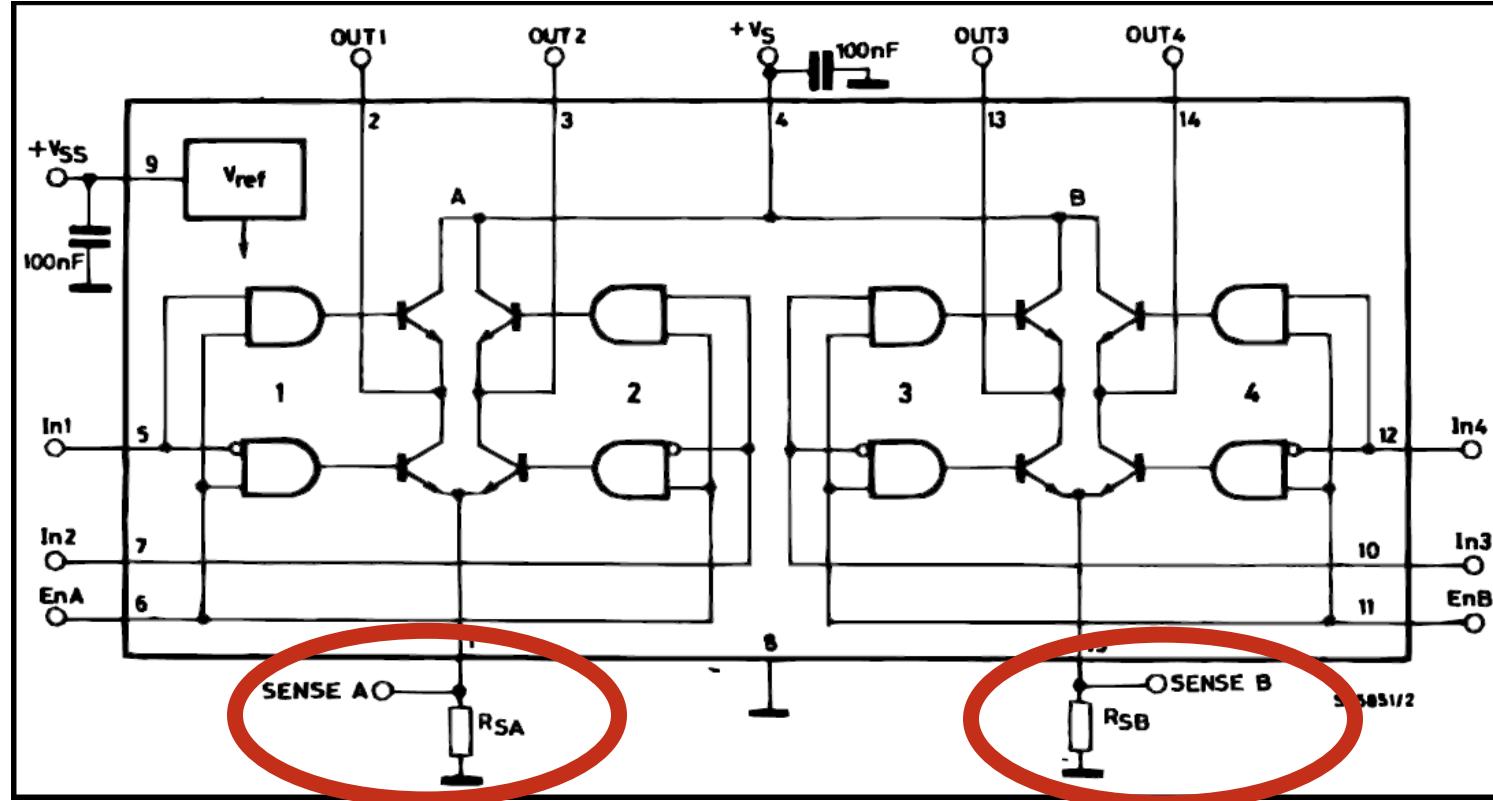


Top Board Pin Out



Motor Driver

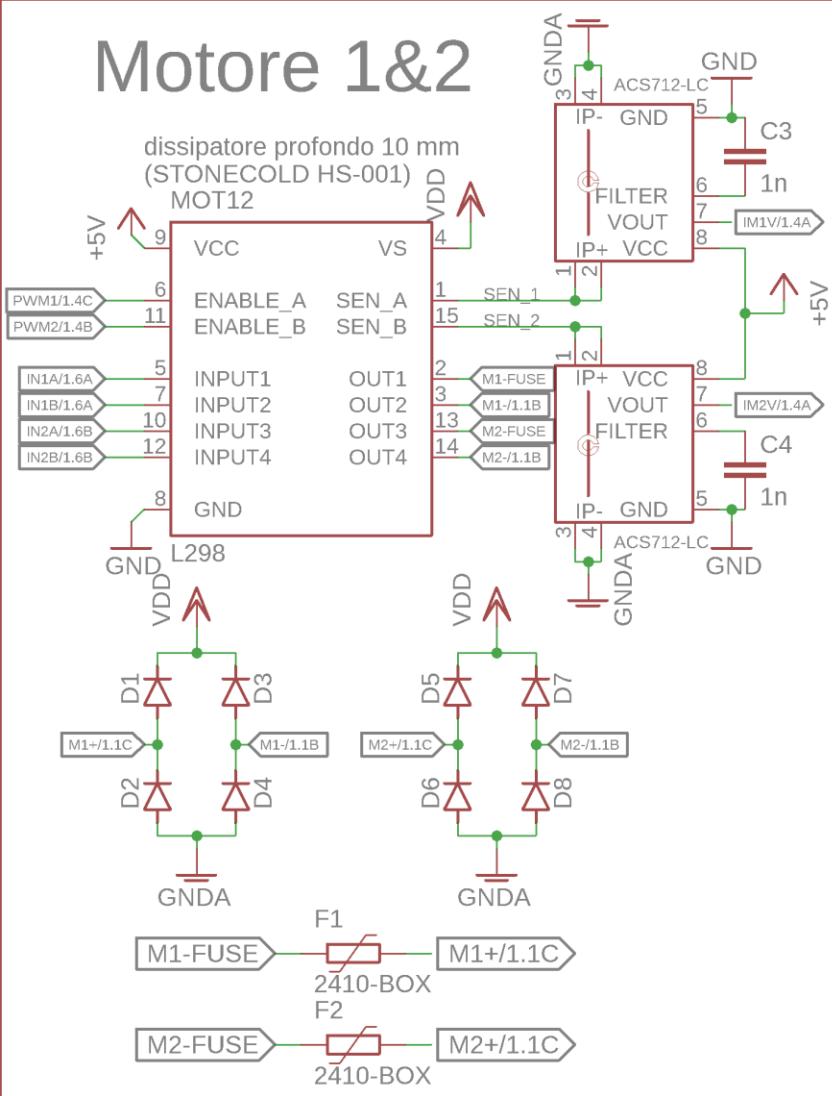
- I motori sono pilotati da 3 L298, i quali sono dei doppi ponti-H



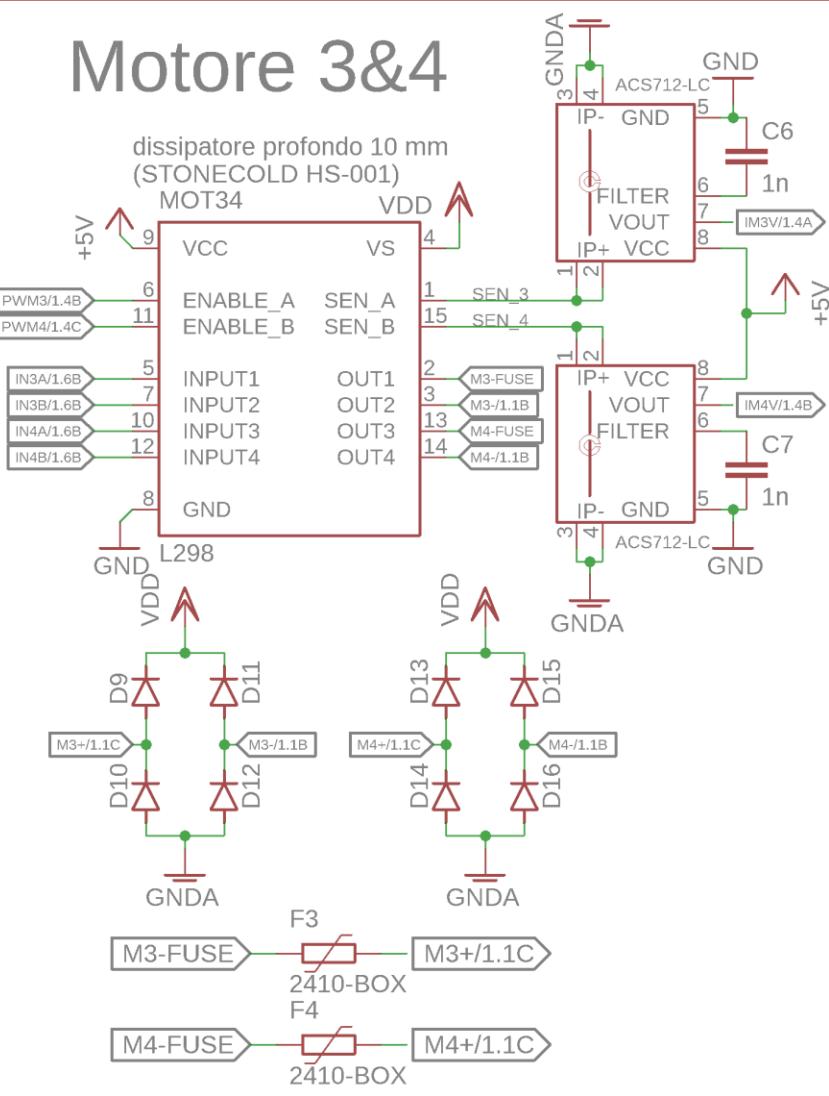
- Ogni driver può erogare per ogni uscita 2 A (normalmente)
- Può gestire ogni chip 25 Watt
- Permette della lettura della corrente passante sul motore
- E' resistente a forti disturbi sui segnali di controllo

Motor Drive Circuit

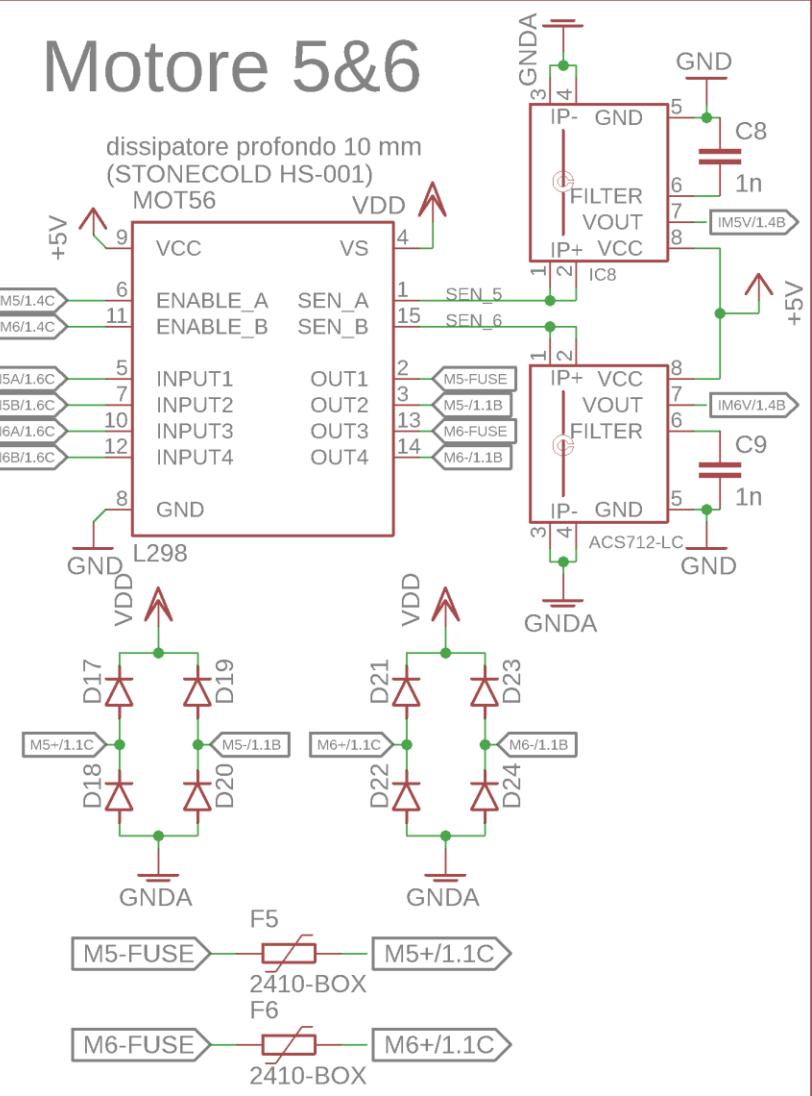
Motore 1&2



Motore 3&4



Motore 5&6

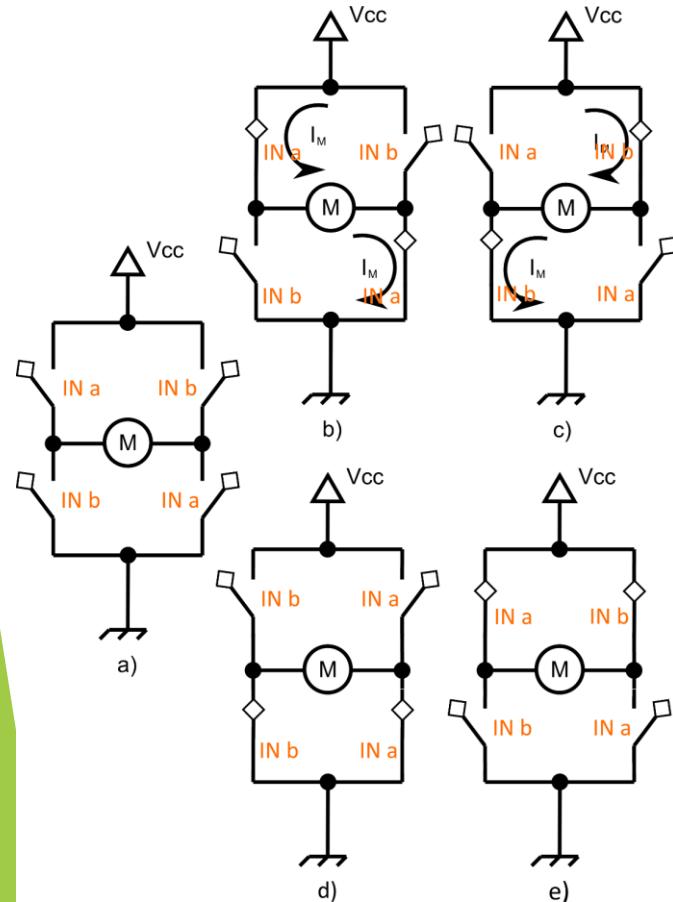


Logica di comando del Software

Per controllare ogni singolo motore si usa una classe di controllo da noi progettata.

Caratteristica chiave di questa classe è che mette a disposizione dei metodi di controllo dei motori temporizzati.

Ovvero alla ricezione di un comando esso viene attuato immediatamente e inizia un conto alla rovescia prima di fermare i motori



È stata pensata al fine di evitare che in caso di disconnessioni dal controllo i motori restino sempre in tensione. Di default il timer è a 2 secondi.

Modalità di controllo dei motori sono:

- `drive_motor(int speed);`
- **(b & c) `drive_motor(int speed, unsigned int delay_time);`**
- `reversDir();`
- `soft_stop();`
- **(e) `hard_stop(unsigned int delay_time);`**
- **(d) `soft_stop(unsigned int delay_time);`**
- **(a) `freeRun();`**

Logica di comando del Software

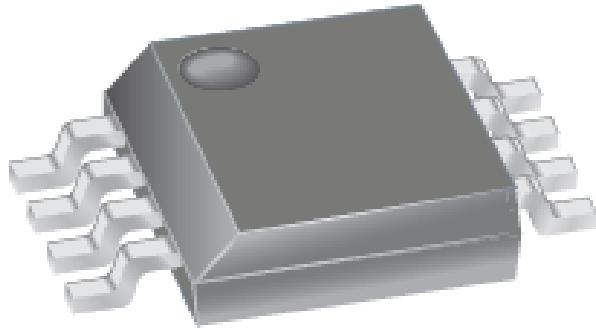
Durante lo svolgimento del codice, nel loop ogni volta viene chiamato per ogni motore il metodo: **void updateMot();**

È lui il responsabile di far evolvere la macchina a stati che ogni comando innesca,

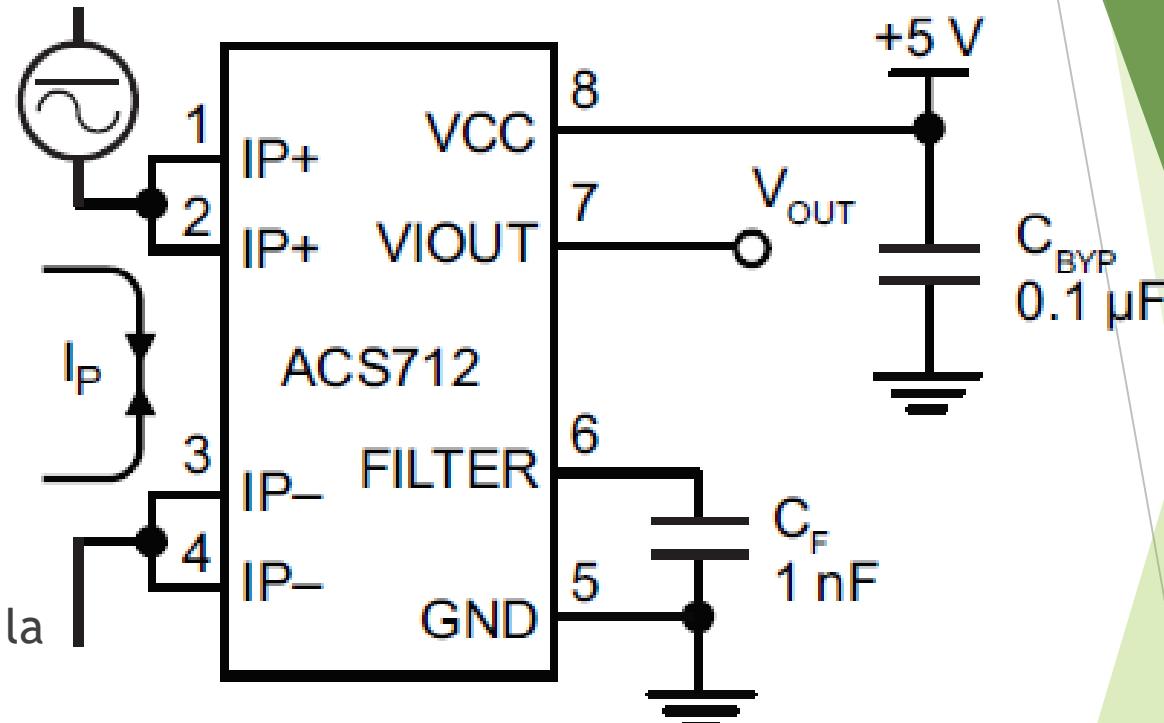
```
void L298N::updateMot() {
    //enum {moving, H_brake, S_brake, wait};
    switch (this->state) {
        case moving:
            //do nothing
            break;
        case movingTiming:
            if (millis() > this->time + this->delay_time)
                this->hard_stop(100);
            break;
        case H_brake:
            if (millis() > this->time + this->delay_time)
                this->soft_stop(1000);
            break;
        case S_brake:
            if (millis() > this->time + this->delay_time)
                this->freeRun();
            break;
        case alwaysBrake:
            break;
        case free_Mot:
        default:
            break;
    }
}
```

Con questa semplice cascata, i comandi più «pericolosi», possono rimanere per poco tempo, a meno che non sia desiderata la loro continuazione e chiamato nuovamente il metodo

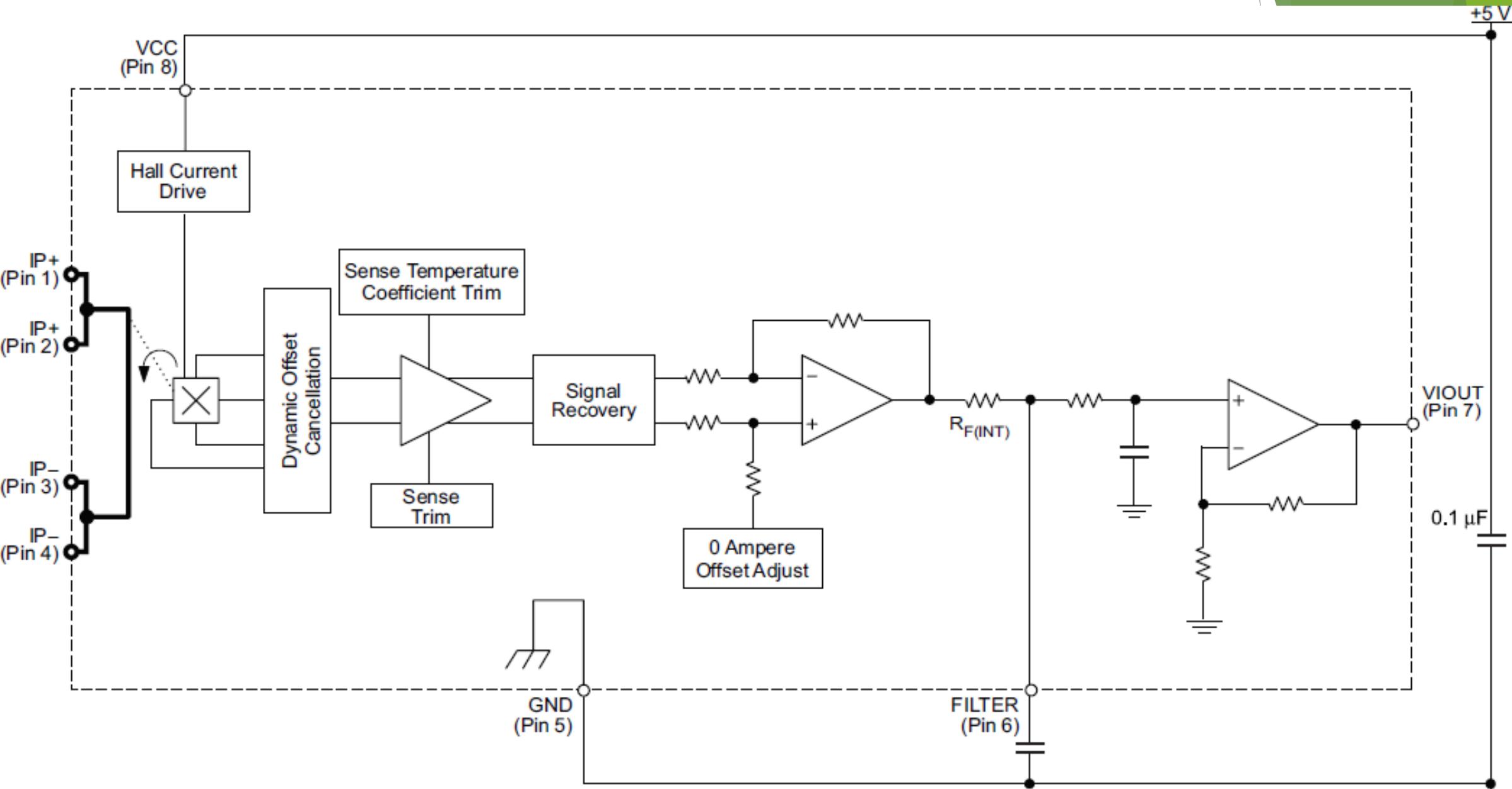
Sensore di Corrente: ACS712



- ▶ Utilizza un sensore Hall per misurare la corrente direttamente
- ▶ La resistenza interna è molto piccola (meno di $1.2\text{ m}\Omega$)
- ▶ Una sensibilità di uscita di 185 mV/A
- ▶ Disaccoppiamento elettrico tra corrente di potenza e lettura della corrente

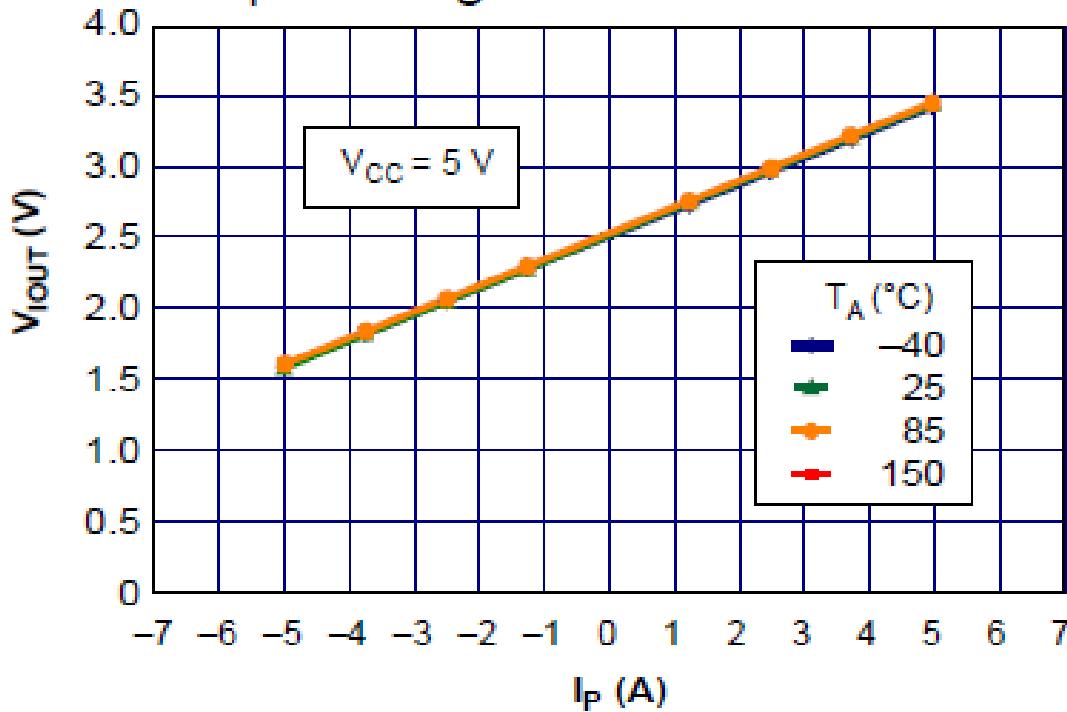


Schema a blocchi:



Comportamento da Datasheet

Output Voltage versus Sensed Current



Notare la linearità al variare della temperatura

Nota bene: 0 A è a 2.5 V

$$I_{read} = \frac{V_{ref} \cdot ADC}{1024 \cdot 0,185}$$
$$ADC = \frac{I_{read} \cdot 1024 \cdot 0,185}{V_{ref}}$$

Leggenda:

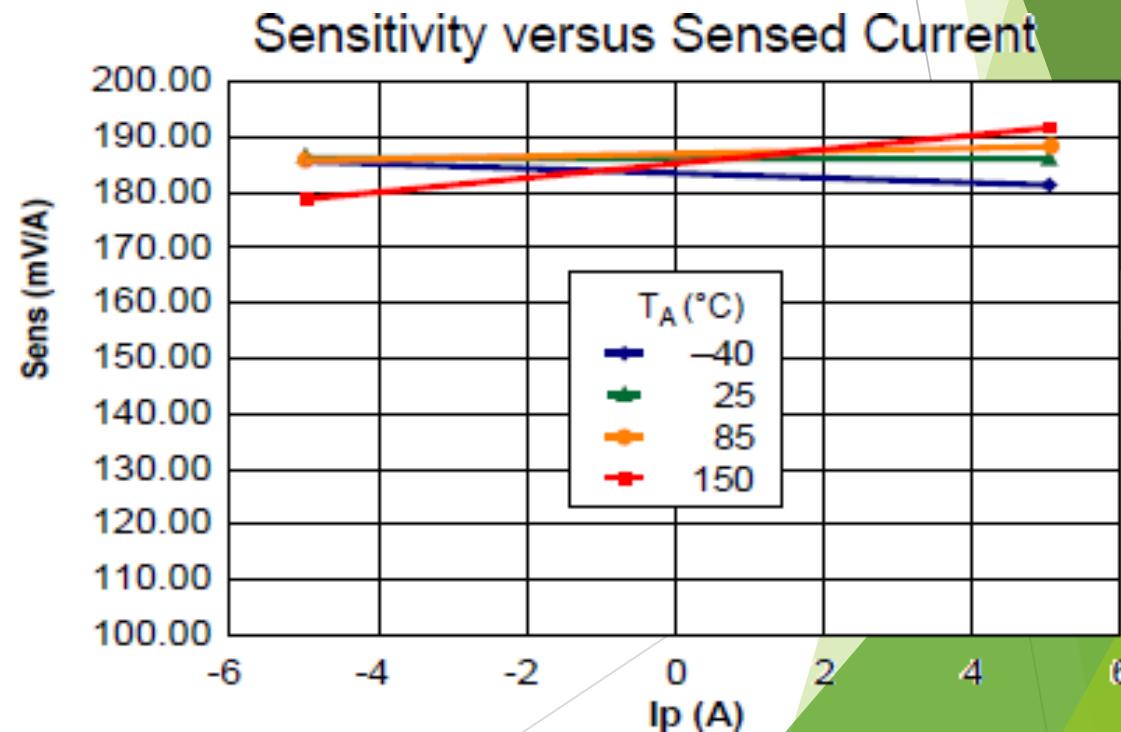
V_{ref} := voltaggio di riferimento dell'ADC [V]

ADC := valore letto dall'ADC

I_{read} := corrente entrante nell'ACS712 [A]

0,185 è la sensibilità dell'ACS712

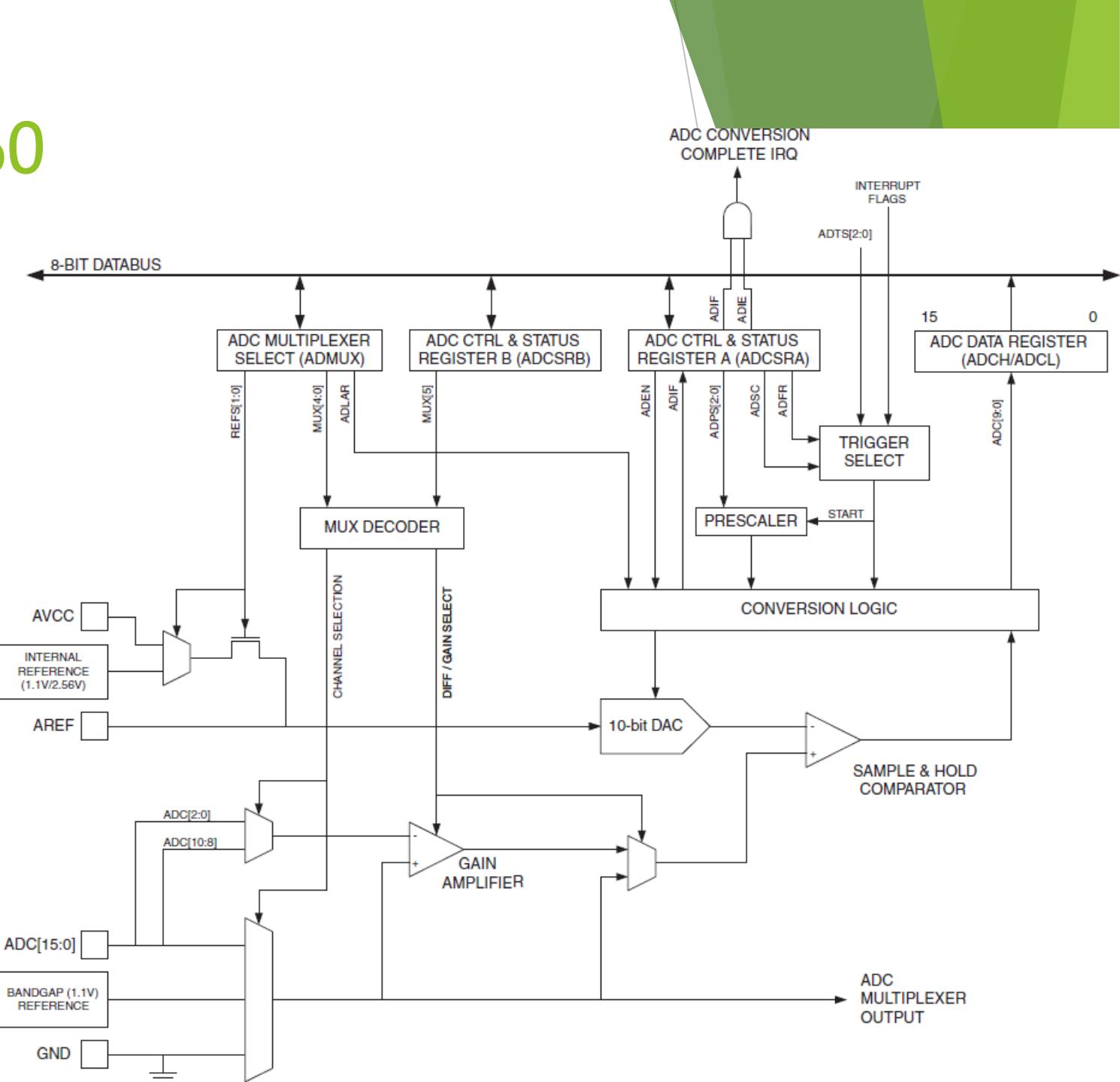
1024 è il fondo scala di un ADC da 10 bit



ADC dell'ATMEGA 2560

Caratteristiche Principali:

- **10-bit Resolution**
- 13 μ s - 260 μ s Conversion Time
- 16 Multiplexed Single Ended Input Channels
- **14 Differential input channels**
- 4 Differential Input Channels with Optional Gain of 10 \times and 200 \times
- 0V - VCC ADC Input Voltage Range
- **2.7V - VCC Differential ADC Voltage Range**
- Selectable 2.56V or 1.1V ADC Reference Voltage
- **Free Running** or Single Conversion Mode
- **Interrupt on ADC Conversion Complete**
- Sleep Mode Noise Canceler



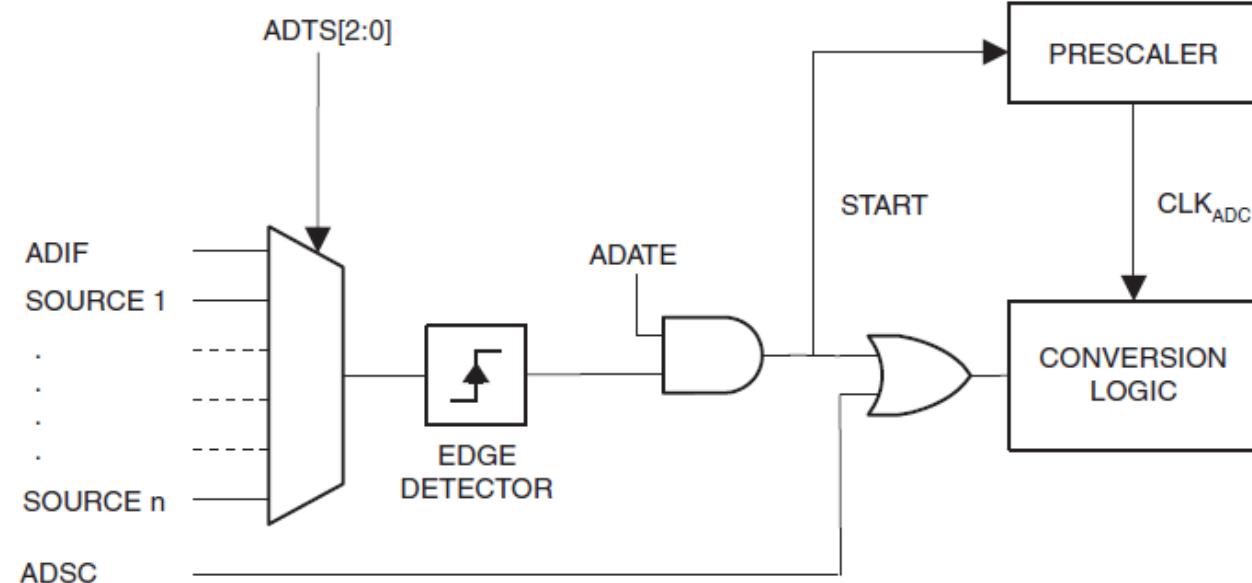
Setup Per leggere la corrente

Visti i valori molto piccoli descritti precedentemente, il comparatore interno è stato impostato con le seguenti caratteristiche:

- ▶ 1.1V Interni Arduino
- ▶ Pin da A0:A6 per input analogico
- ▶ Modalità di free running
 - Finita una conversione inizia immediatamente la successiva, minimizzando i tempi
- ▶ Prescaler a 1/128 Clock
 - per dare 125Khz di clock all'ADC, da datasheet deve essere tra 50khz e 200khz
- ▶ **Lettura differenziale**
- ▶ Gestione in Interrupt

Grazie a questa struttura è possibile:

- Ottenere una misura ogni 112us ~ 8,9Khz
- Diviso per i 6 sensori sono ~ 1,5Khz per sensore



Lettura Differenziale

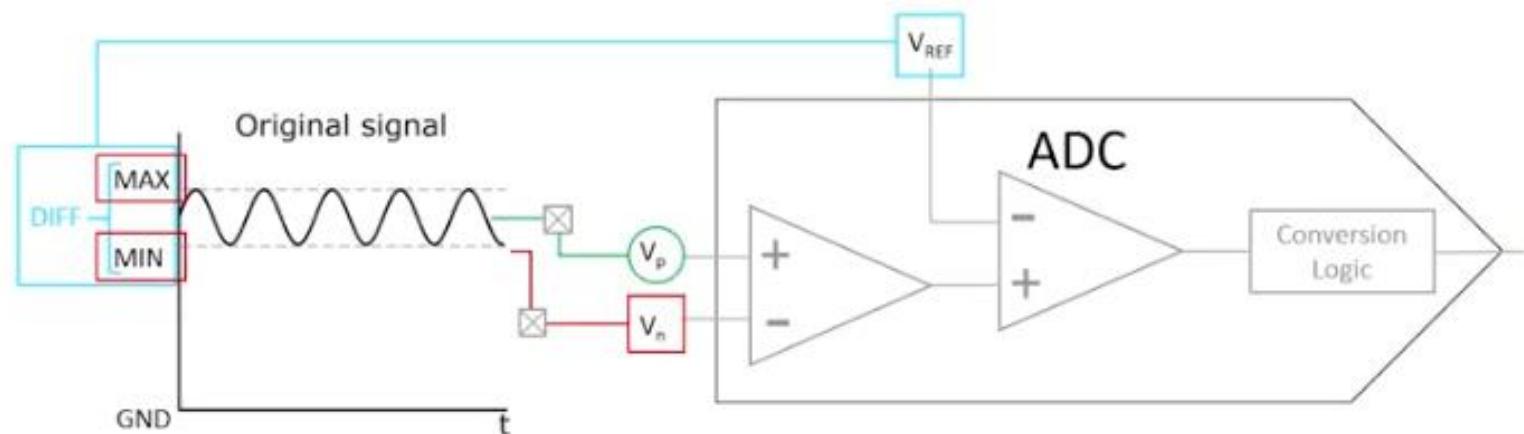
Come detto prima nell'uscita dei sensori di corrente, lo 0A si trova a 2,5V.

Per leggere con la maggiore risoluzione possibile la già flebile tensione, si sfrutta una proprietà interessante degli ADC dell'Atmega 2560:

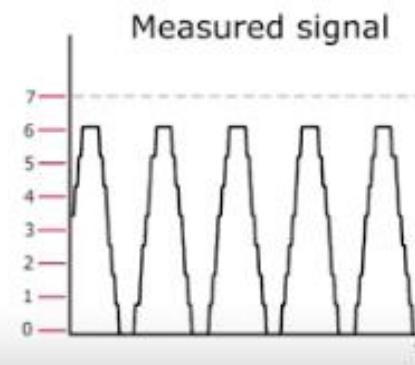
La lettura differenziale:

Nel nostro caso accanto all'Arduino Mega è presente un piccolo trimmer.

Esso permette di scegliere la soglia di sottrazione, in questa maniera si riporta «Virtualmente» a 0 la tensione annullando la continua della lettura



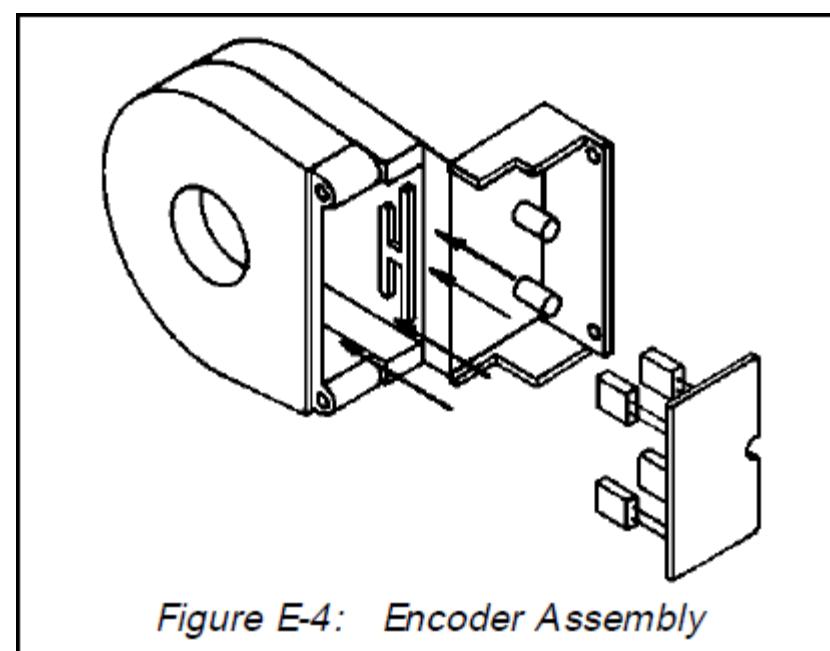
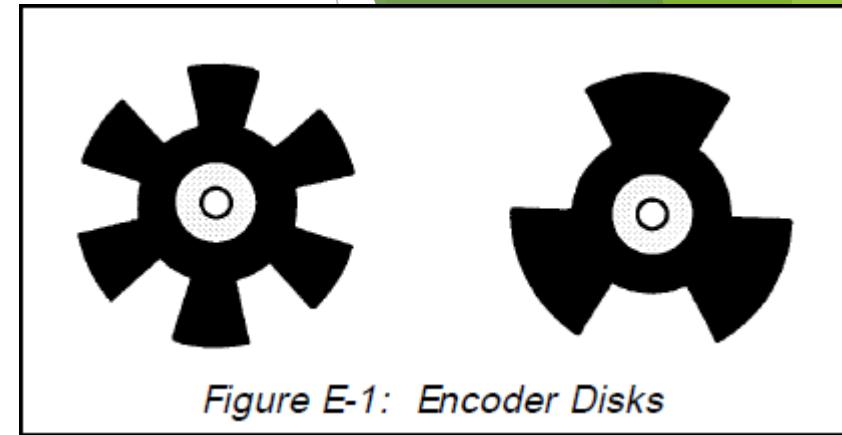
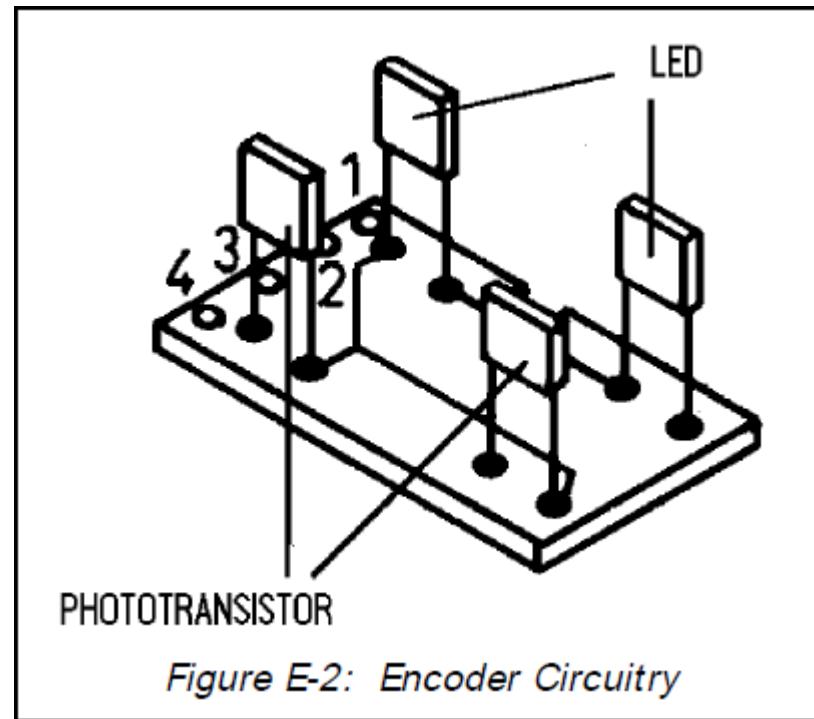
3-bit ADC example
Differential result:



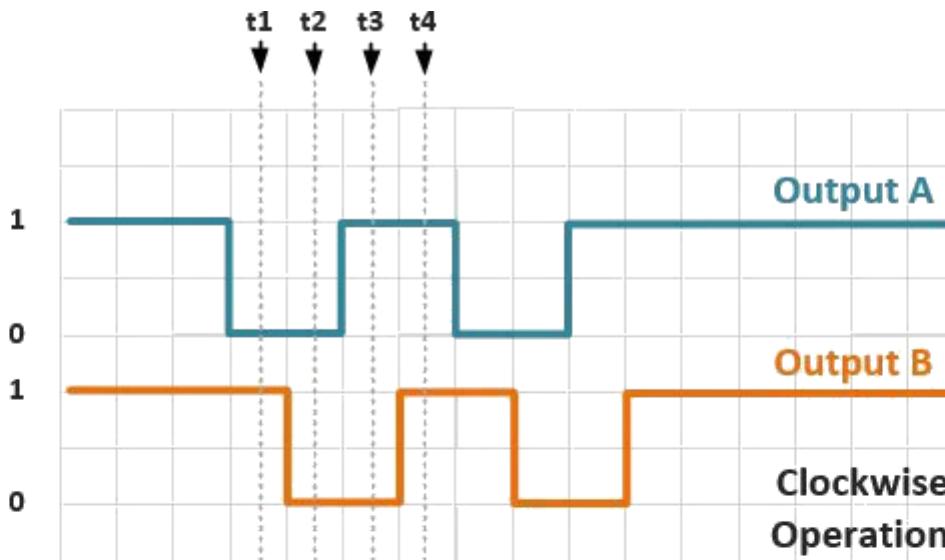
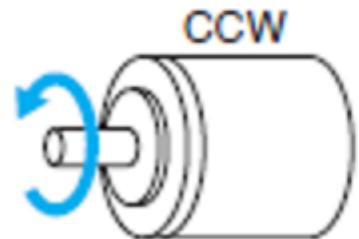
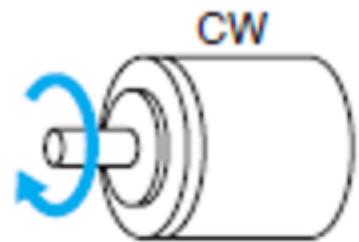
Encoder & MicroSwitch

Per leggere i passi dei motori dello Scrbot, e attuare un controllo, risulta necessario usare degli encoder.

Nel nostro caso sono già Prè-esistenti dentro tutti i motori e sono ottici.

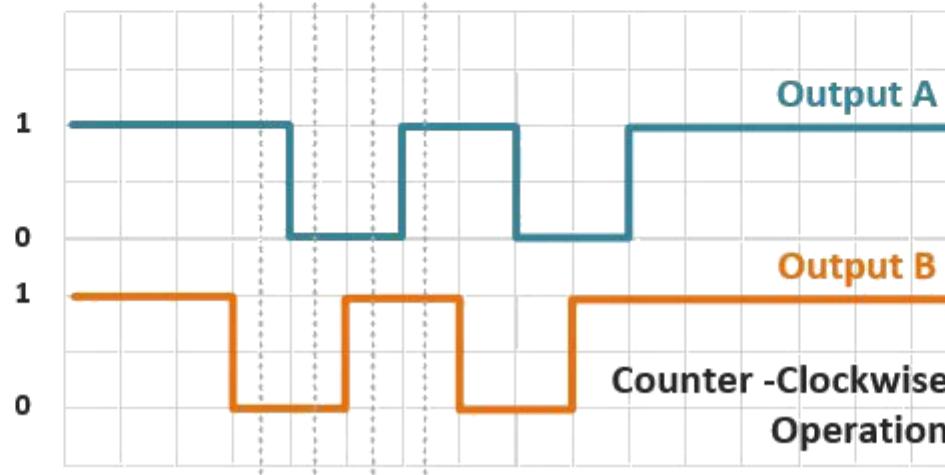


Encoder & MicroSwitch



Clockwise
Sequence

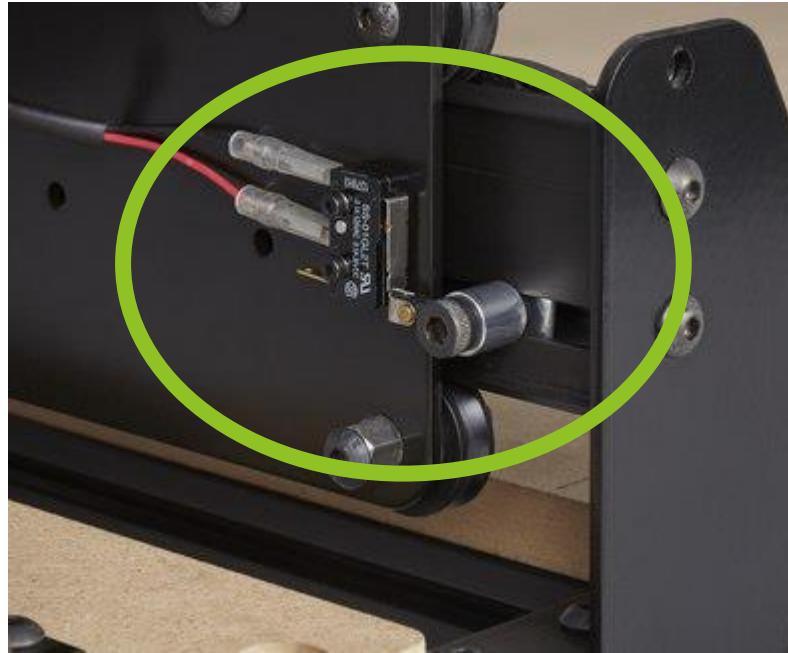
	A	B
t1	0	1
t2	0	0
t3	1	0
t4	1	1



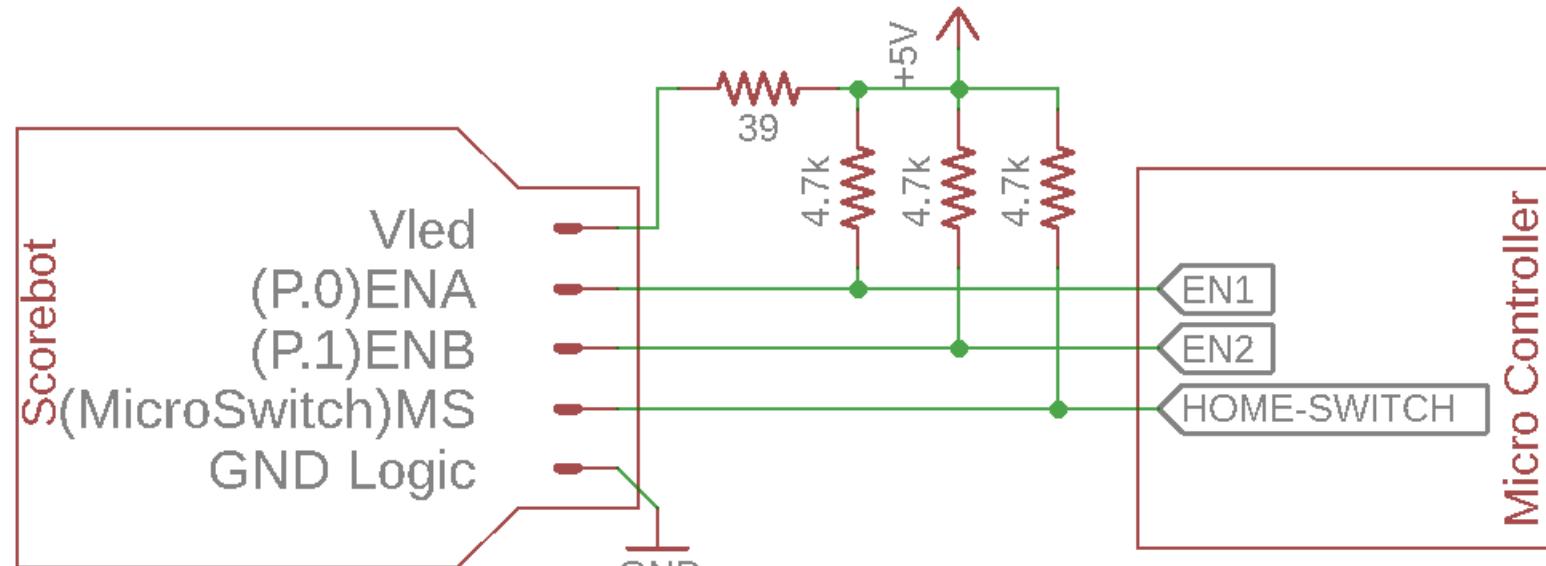
Counter-Clockwise
Sequence

	A	B
t1	1	0
t2	0	0
t3	0	1
t4	1	1

Encoder & MicroSwitch



Per scoprire delle posizioni ASSOLUTE, vengono invece utilizzati dei MicroSwitch,



La rete PULL-UP dei vari sensori

Nel nostro caso abbiamo attivato una funzione dell'Atmega, e la rete pull-up è già attivata dentro il microcontrollore

Encoder & MicroSwitch

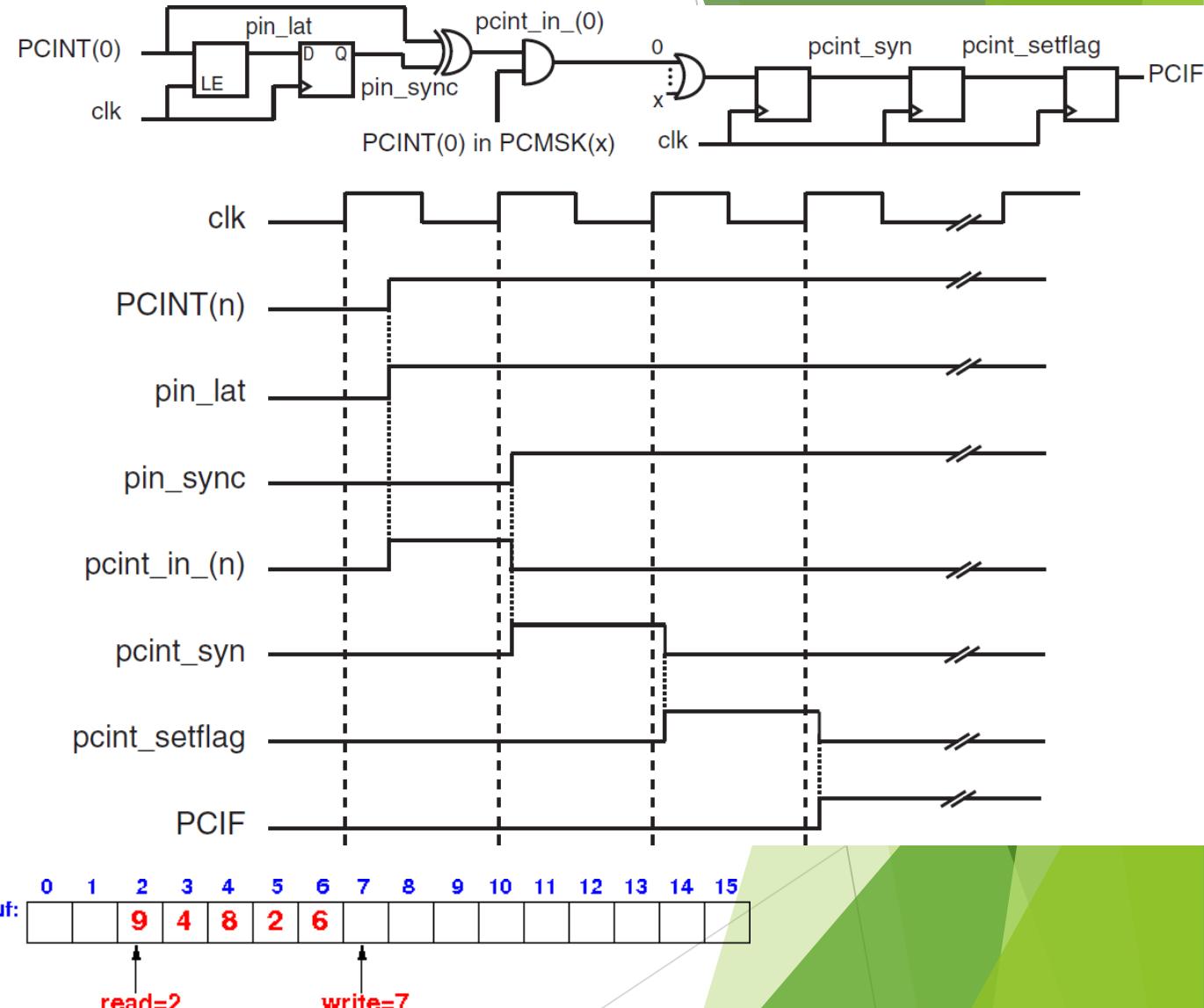
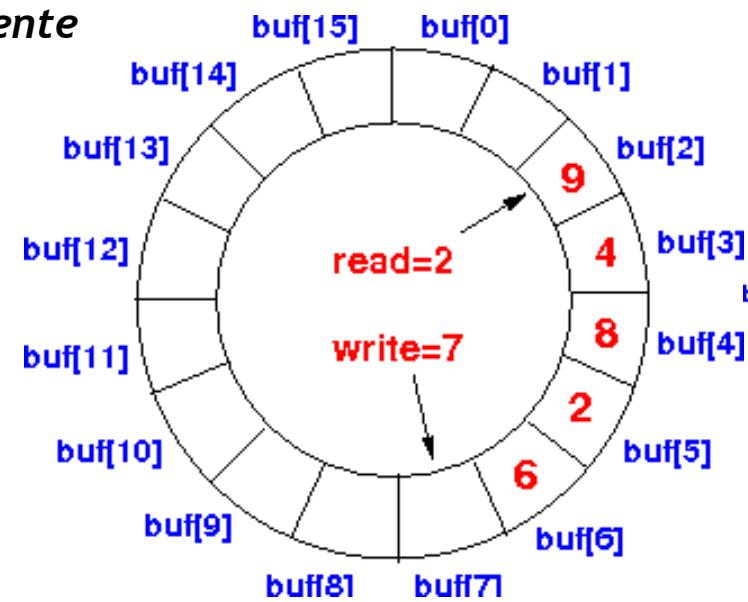
Tutti i pin degli encoder raggiungono dei pin PCINT:

Pin Interrupt Change

La caratteristica di questi pin è che appena uno di loro cambia il suo stato, viene chiamata una routine di gestione.

Nel nostro caso andiamo a leggere lo stato di tutti i pin contemporaneamente e lo salviamo in un **BUFFER CIRCOLARE**.

NOTA BENE, i valori non vengono elaborati immediatamente **buff151** **buff01**



Encoder & MicroSwitch

```
//VARIABILI PRIVATE DI calcStep
#define im 0 //impossibile
//          0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
int8_t enc_states[] = { 0, -1, 1, im, 1, 0, im, -1, -1, im, 0, 1, im, 1, -1, 0 }; /*[old]BA-BA[new]*/
byte chAold, chBold, chAnew, chBnew, code;
void calcStep(int oldEn, int newEn) {
    /* Monto i bit BA (A LSB)*/
    chAold = oldEn & 0x00FF;
    chBold = (oldEn & 0xFF00) >> 7; //disallineo di 1 per semplificare l'or binario
    chAnew = newEn & 0x00FF;
    chBnew = (newEn & 0xFF00) >> 7;

    //Calcolo primo encoder
    code = ((chAold & 1) | (chBold & 0b10)) << 2;
    code |= ((chAnew & 1) | (chBnew & 0b10));
    passi[0] += (enc_states[code]);
    //Calcolo Restanti 6
    for (byte i = 1; i < nMot; i++) {
        chAold >>= 1;
        chBold >>= 1;
        code = ((chAold & 1) | (chBold & 0b10)) << 2;
        chAnew >>= 1;
        chBnew >>= 1;
        code |= ((chAnew & 1) | (chBnew & 0b10));
        passi[i] += (enc_states[code]);
    }
}
```

All'interno del loop, in ogni ciclo viene elaborata tutta la coda accumulata durante la gestione delle altre routine tramite questo algoritmo ←

Per elaborare tutti e 6 gli Encoder impiega:

- ~186 colpi di Clock (~80KHz)

Quindi la frequenza massima di arrivo è lievemente inferiore dovendo gestire le altre routine, approssimiamo circa a 65KHz per stare in sicurezza

Noi non superiamo i 2 KHz a Encoder



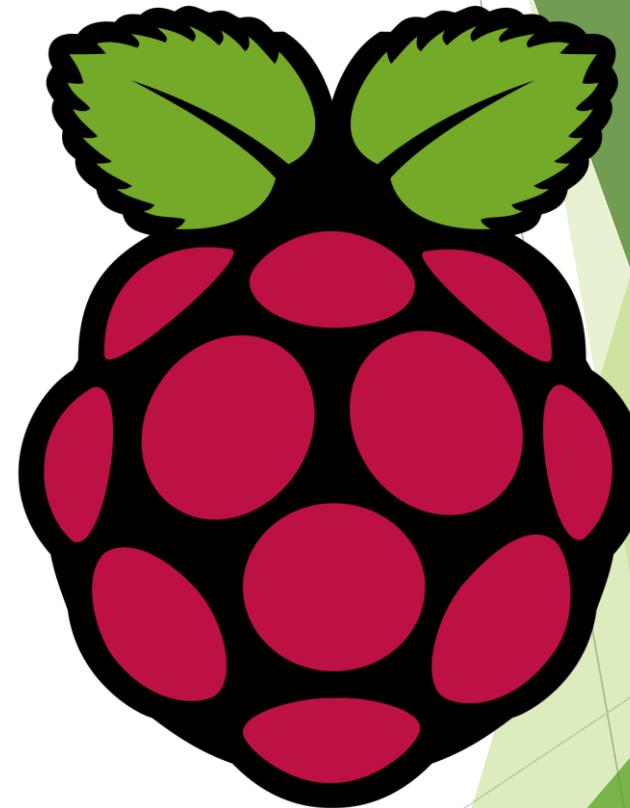
Encoder & MicroSwitch

La strategia di Homming consiste nel muovere singolarmente i motori fino a trovare i MicroSwitch della home.

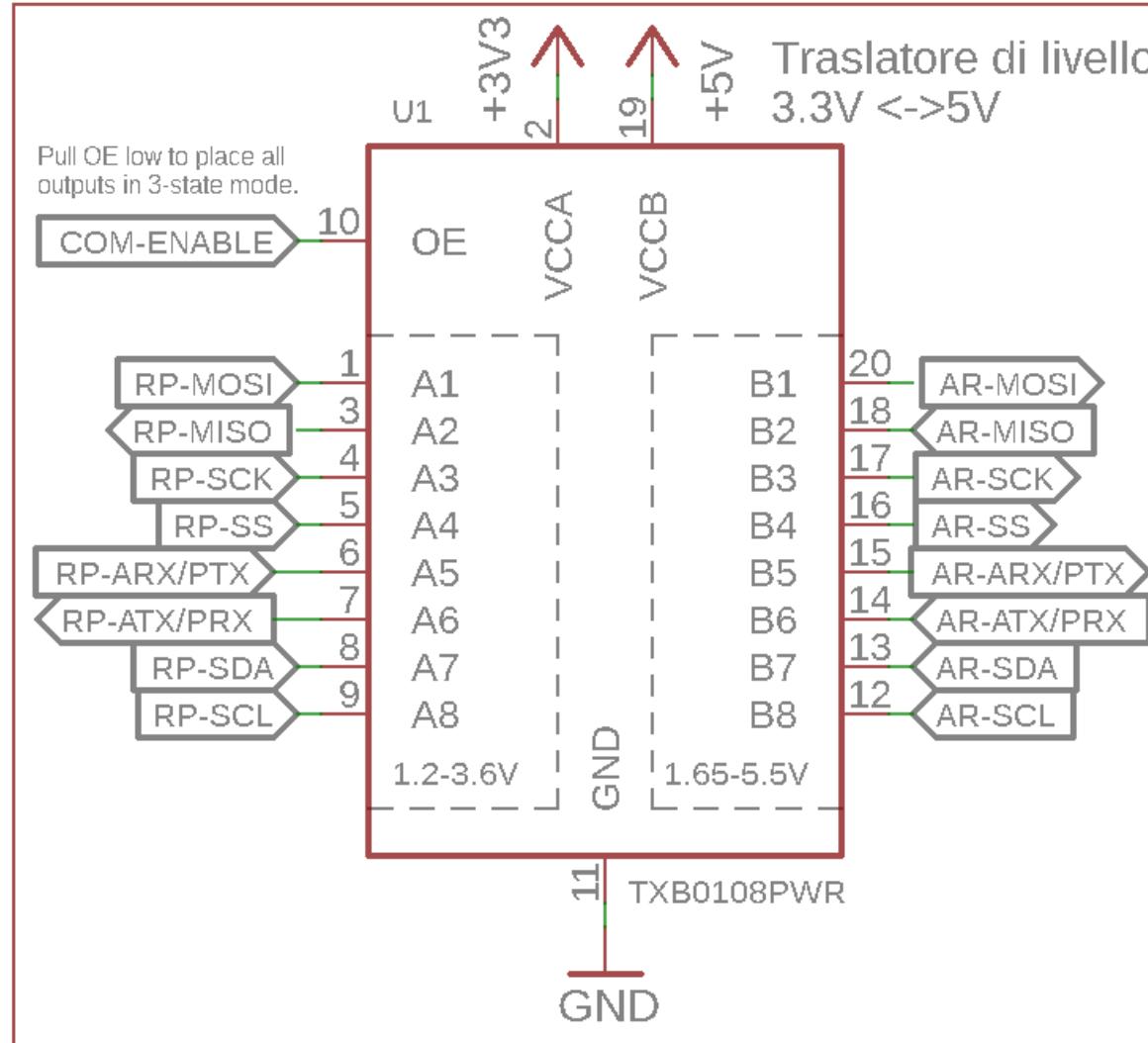
1. Si inizia con l'avambraccio che sale cercando la home
2. Successivamente inizia la ricerca il gomito
3. Polso e pinza hanno i motori a **controllo differenziale**, vanno per tanto cercati insieme:
 - Si cerca la home del ROLL
 - Si cerca la home del PITCH
4. Trovata questa PSEUDO HOME, viene ri-eseguito il ciclo, per garantire che la posizione raggiunta sia vera e non influenzata dagli spostamenti degli altri link
5. Viene cercata la home della base

Al termine tutti gli encoder vengono messi a 0 e la procedura è terminata.

Comunicazione Arduino e Raspberry



Software Arduino e Raspberry Pi



Software Arduino e Raspberry



Alternate Function		
3.3V PWR	1	
I2C1 SDA	GPIO 2	3
I2C1 SCL	GPIO 3	5
GPIO 4		7
GND		9
GPIO 17		11
GPIO 27		13
GPIO 22		15
3.3V PWR		17
SPI0 MOSI	GPIO 10	19
SPI0 MISO	GPIO 9	21
SPI0 SCLK	GPIO 11	23
Reserved		27
GPIO 5		29
GPIO 6		31
GPIO 13		33
SPI1 MISO	GPIO 19	35
GPIO 26		37
GND		39
2	5V PWR	
4	5V PWR	
6	GND	
8	UART0 TX	
10	UART0 RX	
12	GPIO 18	
14	GND	
16	GPIO 23	
18	GPIO 24	
20	GND	
22	GPIO 25	
24	GPIO 8	SPI0 CS0
26	GPIO 7	SPI0 CS1
28	Reserved	
30	GND	
32	GPIO 12	
34	GND	
36	GPIO 16	SPI1 CS0
38	GPIO 20	SPI1 MOSI
40	GPIO 21	SPI1 SCLK

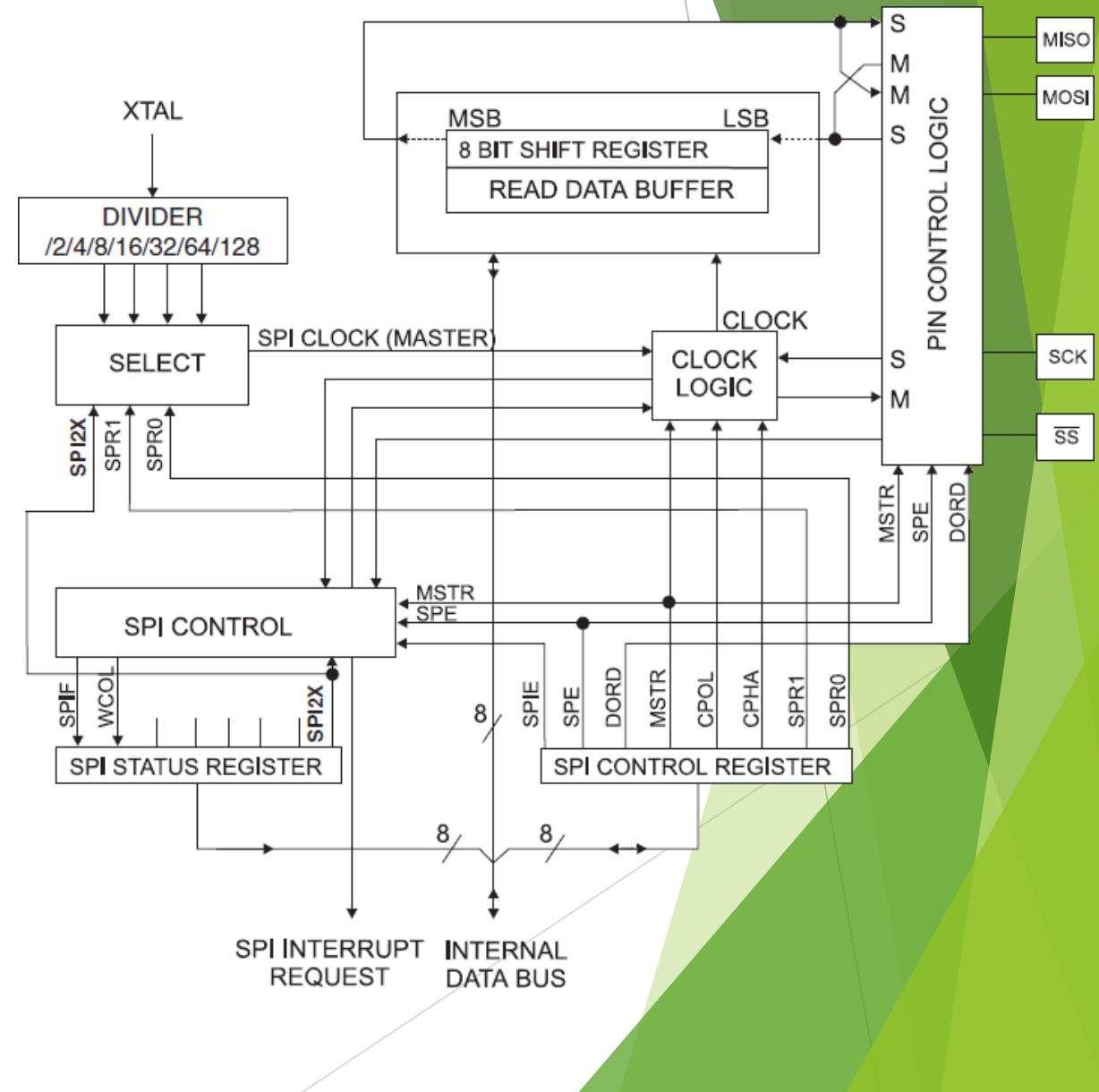
SPI

La SPI è una connessione full duplex sincrona, ovvero i due interlocutori parlano contemporaneamente e con la stessa lunghezza di dati alla velocità imposta dal master (nel nostro caso il raspberry pi)

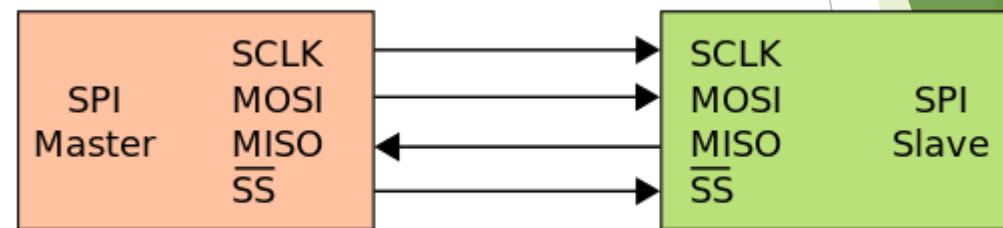
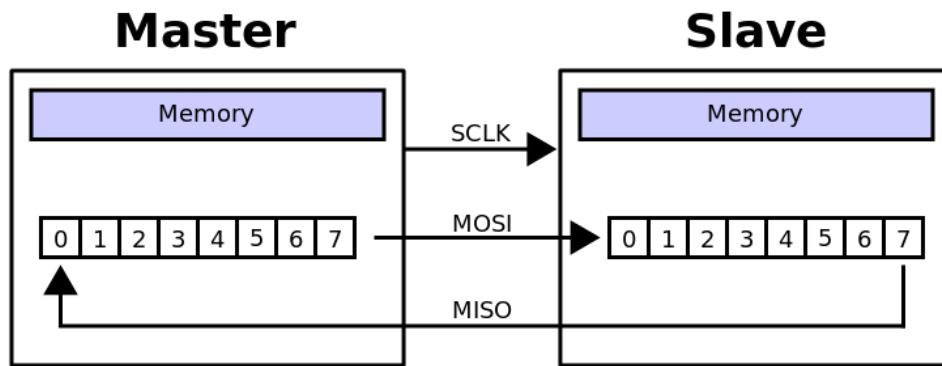
Il significato dei pin è:

- MISO = Master In Slave Out
- MOSI = Master Out Slave In
- SCK = Clock
- SS = Slave Select

Il dispositivo è in grado di generare interrupt quando viene completata una trasmissione (8 bit) Il nostro lavoro è consistito nel progettare un protocollo in cui il primo byte indica il tipo di pacchetto che si vuole trasmettere e successivamente vengono inviati dati da ambo le parti

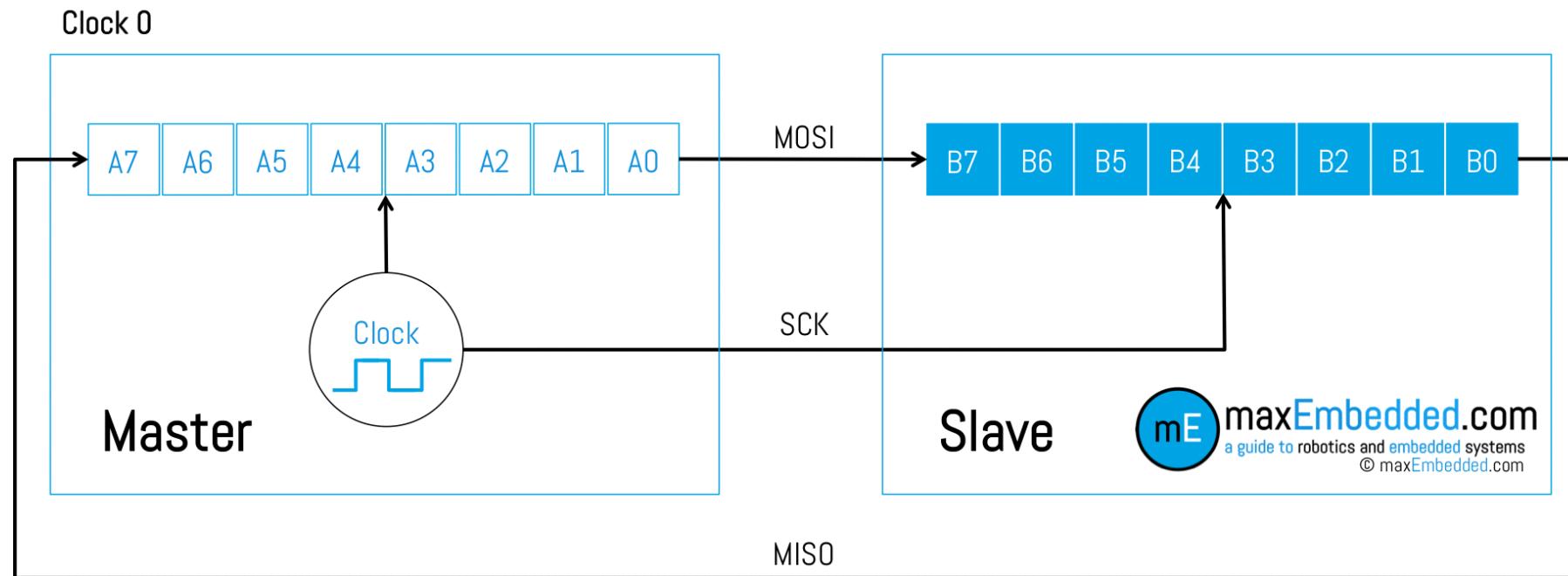


Ma come funziona una comunicazione SPI?



Qui potete vedere la connessione logica che avviene tra i due dispositivi, i quali ad ogni colpo di clock del master faranno shiftare di 1 bit. Di conseguenza per ogni blocco di dati saranno necessari 8 colpi di clock.

Ma come funziona una comunicazione SPI?



Cosa è stato fatto sulla Raspberry Pi?

E' stata utilizzata la libreria di sistema <linux/spi/spidev.h> la quale include una gestione lato kernel della comunicazione della SPI, lasciando a noi solo il compito di impostare velocità di trasmissione, numero dei byte da inviare e allocamento della memoria.

E' stata scritta una libreria in C++ che può essere inclusa in un qualsiasi progetto che utilizza la nostra scheda per governare uno Scrbot

La classe si occupa di mandare messaggi STANDARD all'Arduino il quale è in grado di leggerli ed eseguirli

I possibili comandi sono:

Comando	Raspberry invia	Raspberry riceve
setPWM	Valori dei PWM di tutti i motori	Conteggio degli ENCODER
getCurrent		Ultime letture dei sensori di corrente
getSetting		Impostazioni salvate sulla EEPROM
setSetting	Nuovi impostazioni da salvate in EEPROM	
goHome	Attiva la routine di ricerca della Home	

Settings personali delle Board

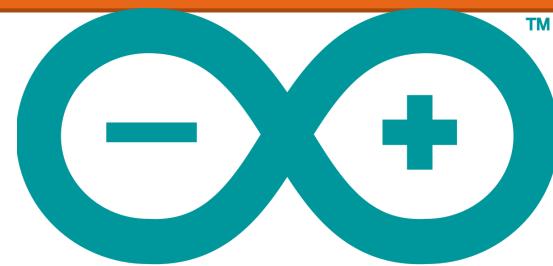
```
typedef struct settingsBoard_ {
    /* Valore massimo di passi prima di considerarsi
     * fuori range di sicurezza (numeri >0) */
    int maxEn[nMot];

    /* Valore minimo di passi prima di considerarsi
     * fuori range di sicurezza (numeri <0) */
    int minEn[nMot];

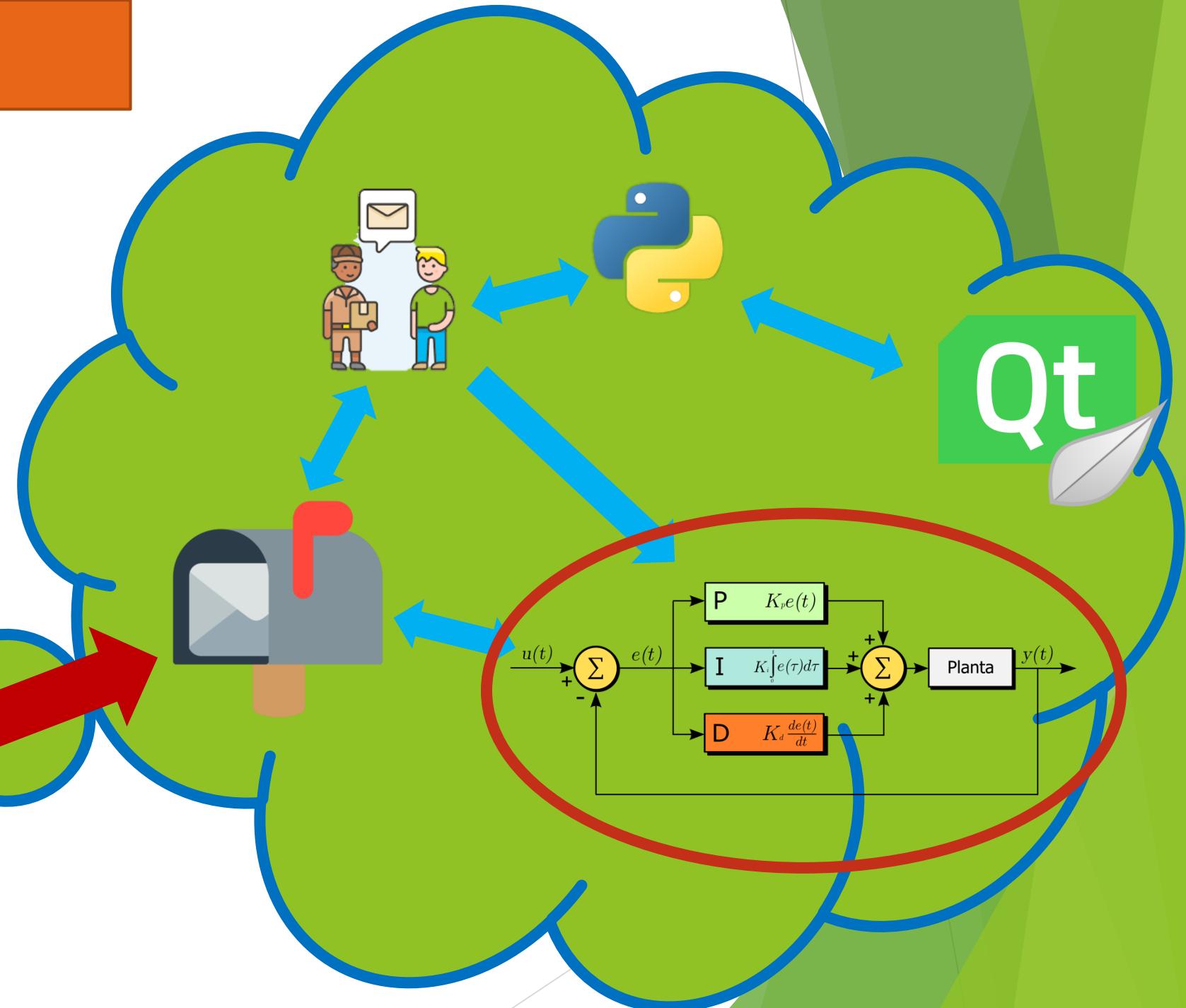
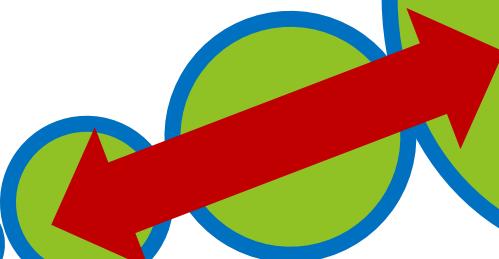
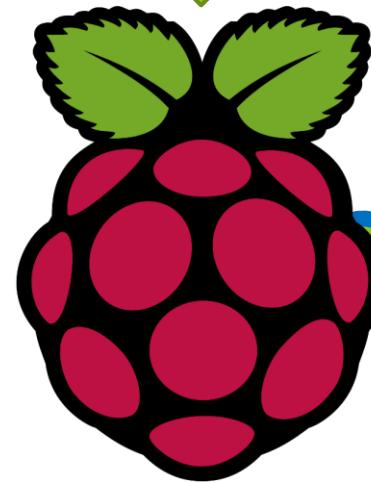
    /* Valore massimo di corrente Efficace
     * (con una media di 1 ms ~ ultime 8 letture),
     * numero * 8 per semplificare i conti */
    int maxCurrMed[nMot];

} settingsBoard;
```

PID Section



ARDUINO

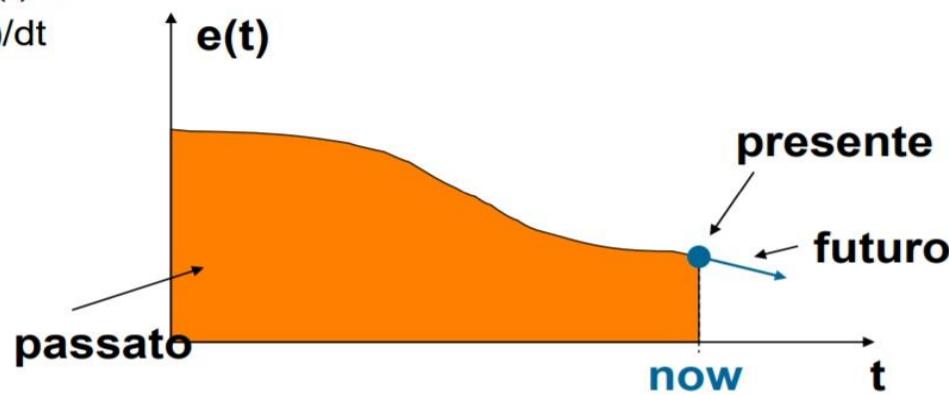


CONTROLLORI PID

- ▶ I regolatori PID sono dei controllori standard che, per mezzo di dispositivi di correzione (manopole o software), permettono di regolare i parametri del sistema di controllo entro ampi limiti, così da poter essere adattati al sistema di regolazione in cui vengono inseriti. Questi dispositivi possono essere utilizzati per esempio per controllare una portata, per controllare una temperatura o per il movimento di bracci robotici.
- ▶ In un regolatore PID la variabile d'uscita viene generata in base al contributo di tre fattori:
 - Il primo è proporzionale all'errore tra il riferimento e la variabile da controllare $y(t)$.
 - Il secondo è proporzionale all'integrale dell'errore.
 - Il terzo è proporzionale alla derivata dell'errore, ciò significa che risente della velocità di variazione dell'errore stesso.

Cosa vogliamo conoscere sul segnale errore $e(t)$?

- Presente $\rightarrow e(t)$
- Passato $\rightarrow \int e(t)dt$
- Futuro $\rightarrow de(t)/dt$



- ▶ La legge di controllo del PID può essere descritta come:

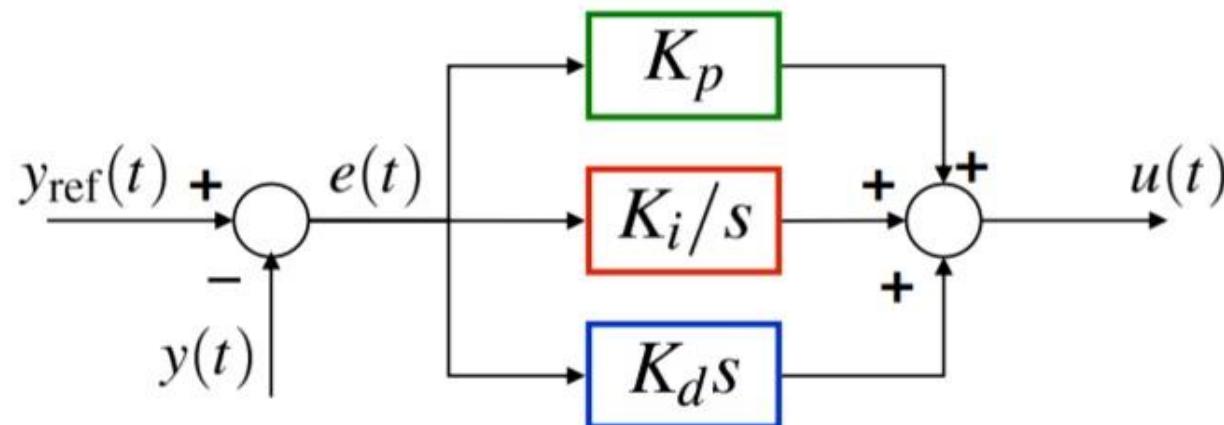
$$e(t) = r(t) - y(t)$$

$$u(t) = K_p e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt}$$

Dove $T_i = K_p / K_i$ e $T_d = K_d / K_p$.

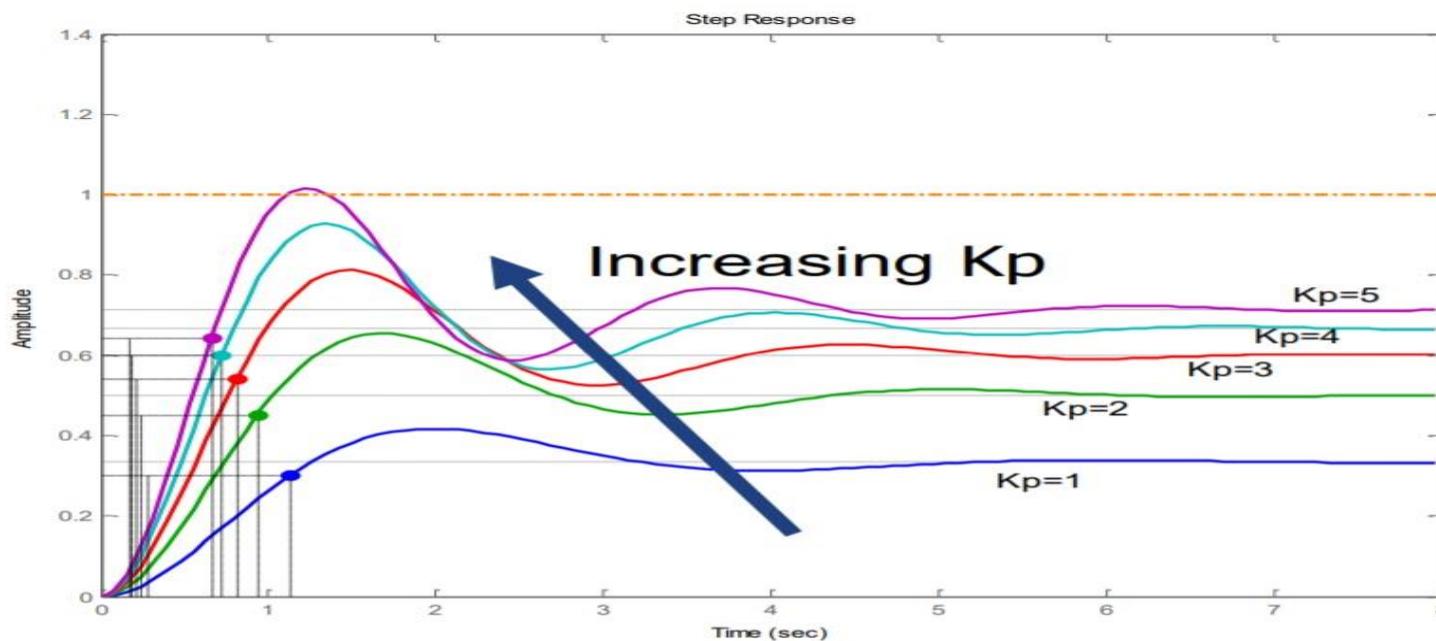
Graficamente lo schema a blocchi :

- **azione proporzionale all'errore**
- **azione proporzionale all'integrale dell'errore**
- **azione proporzionale alla derivata dell'errore**



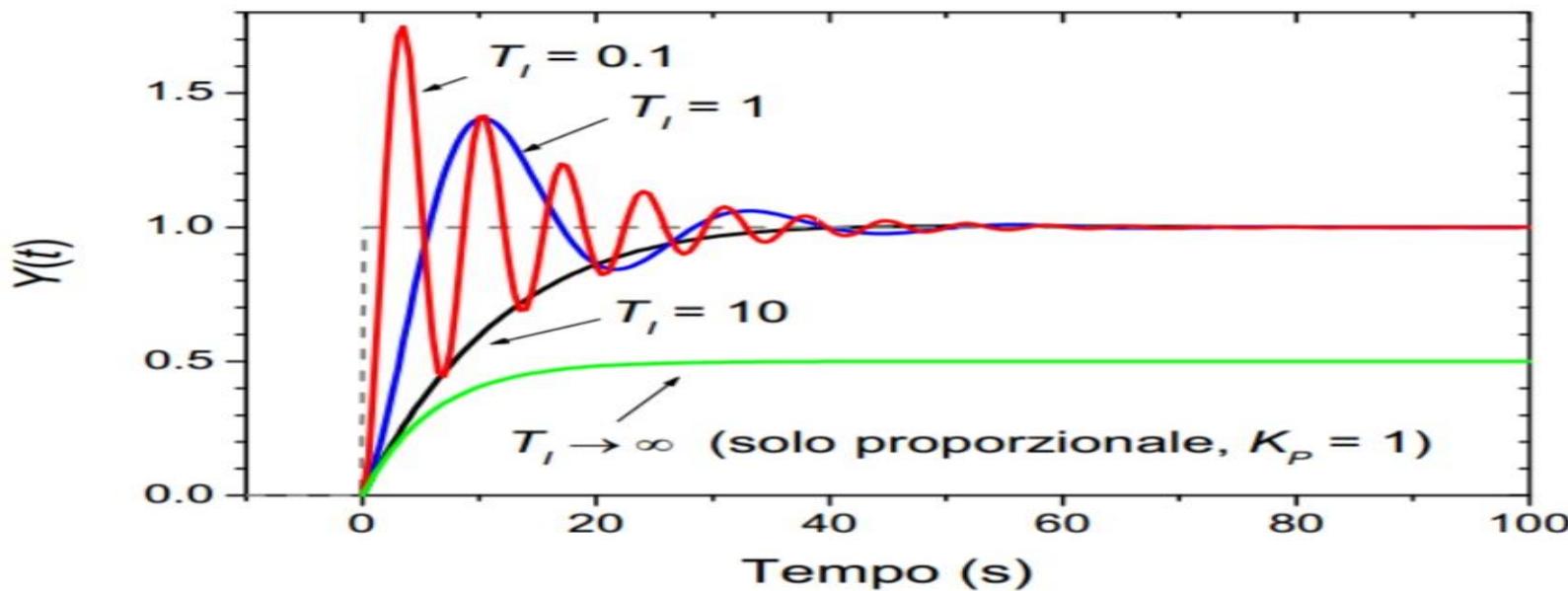
Azione Proporzionale

Simbolo	Funzione di regolazione	Scopo principale	Stabilità	Tempo di risposta
K_p	Uno scostamento sull'ingresso (Errore) produce una variazione dell'uscita proporzionale all'ampiezza dello scostamento	Fa variare la grandezza regolante in funzione della grandezza regolata		
Aumenta K_p	-	-	Migliora	Rallenta
Diminuisce K_p	-	-	Peggiora	Accelera



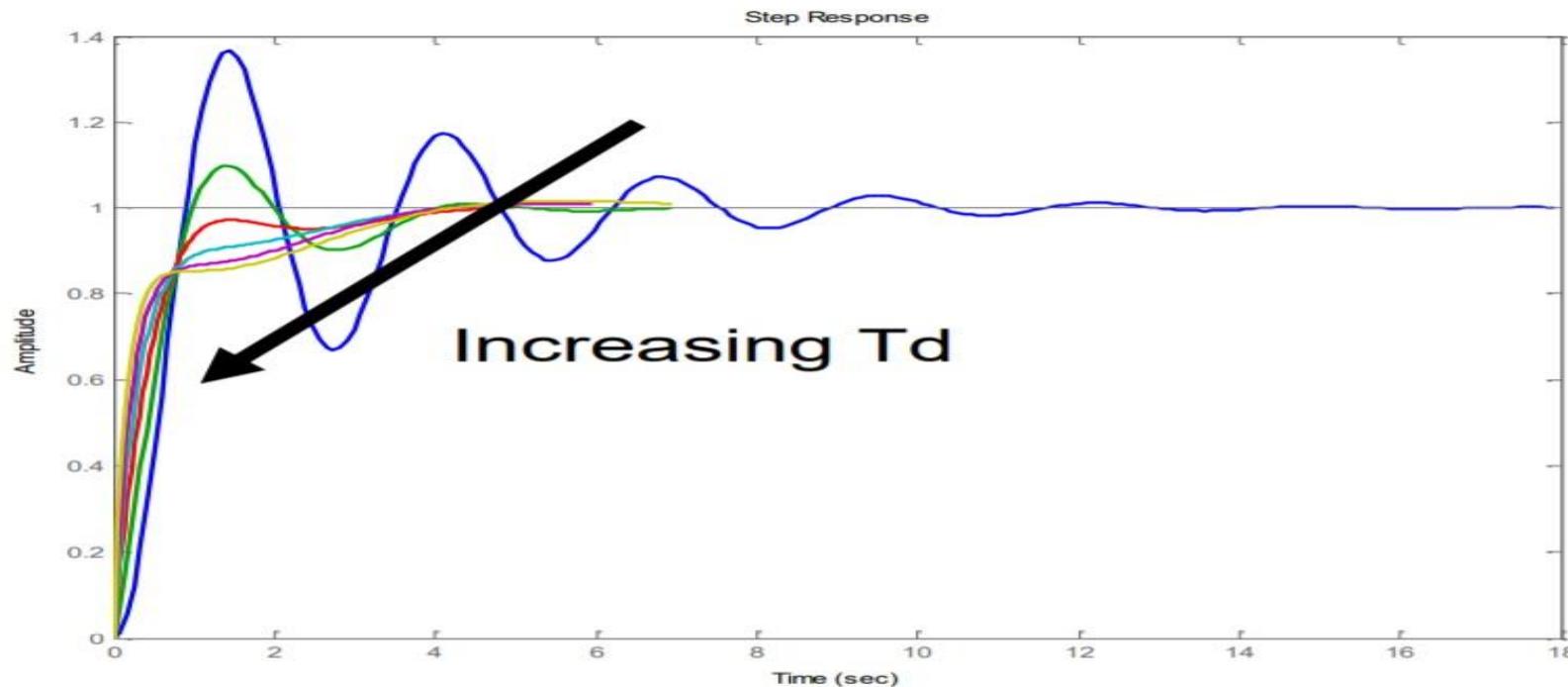
Azione Integrale

Simbolo	Funzione di regolazione	Scopo principale	Stabilità	Tempo di risposta del loop
T_I	Appena si ha uno scostamento sull'ingresso (Errore), si produce una variazione dell'uscita con velocità proporzionale allo scostamento	Fissa il punto di regolazione (Elimina l'offset dato dall'azione proporzionale)		
Aumenta T_I	-	-	Migliora	Rallenta
Diminuisce T_I	-	-	Peggiora	Accelera



Azione Derivativa

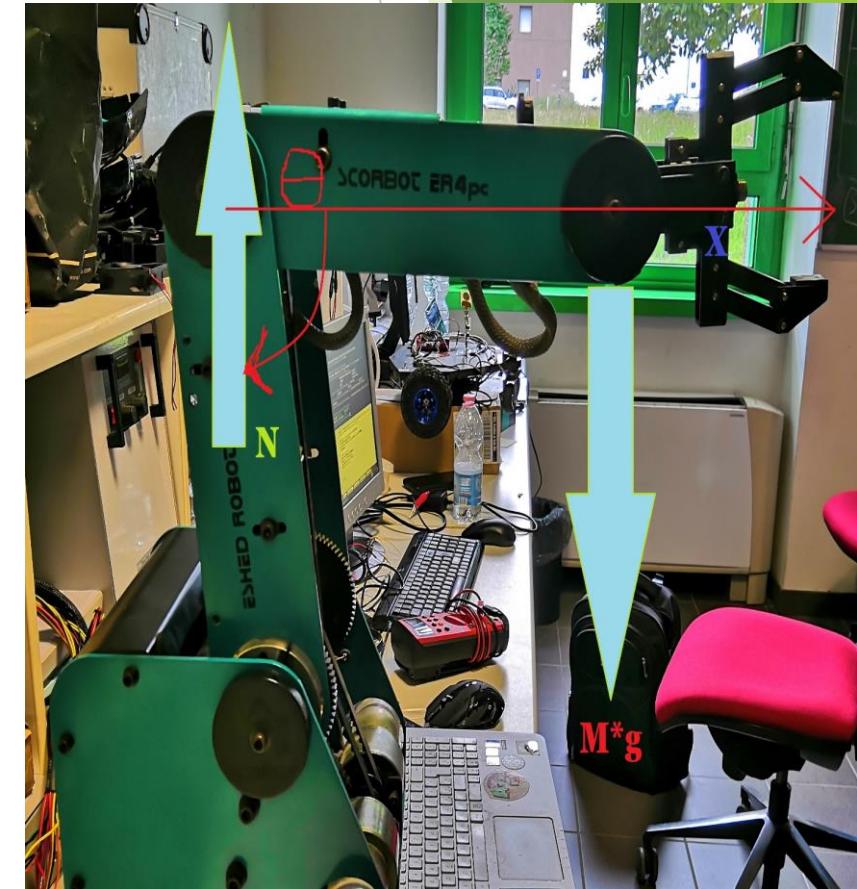
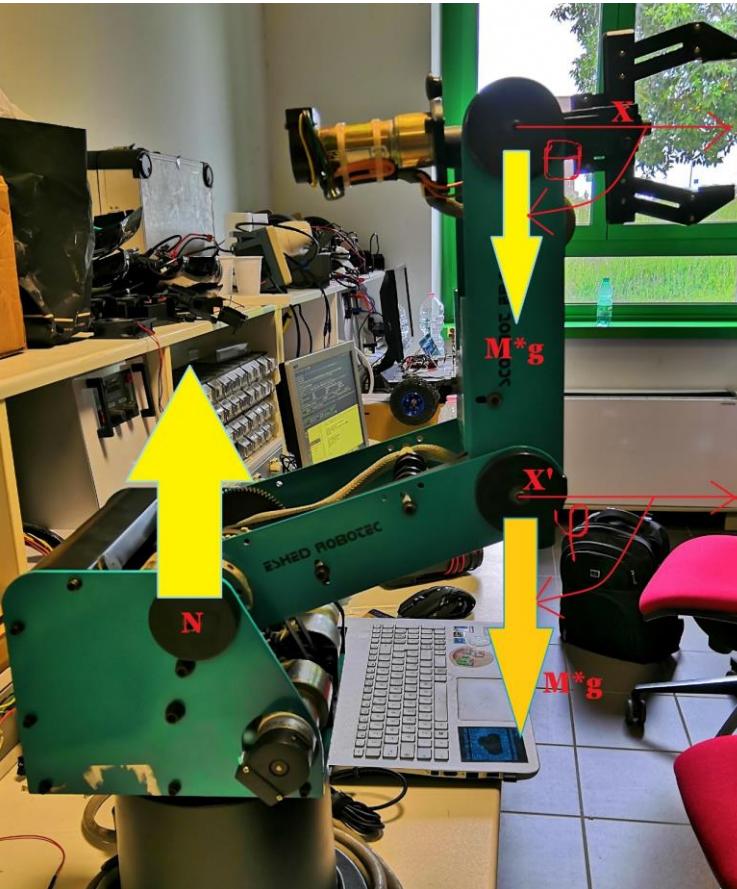
Simbolo	Funzione di regolazione	Scopo principale	Stabilità	Tempo di risposta del loop
T_D	Uno scostamento sull'ingresso (Errore), produce una variazione dell'uscita proporzionale alla velocità di variazione dello scostamento	Diminuisce il tempo di risposta per il ritorno al punto di regolazione	-	-
Aumenta T_D	-	-	Peggiora	Accelerata
Diminuisce T_D	-	-	Peggiora	Rallentata



Sistema non lineare

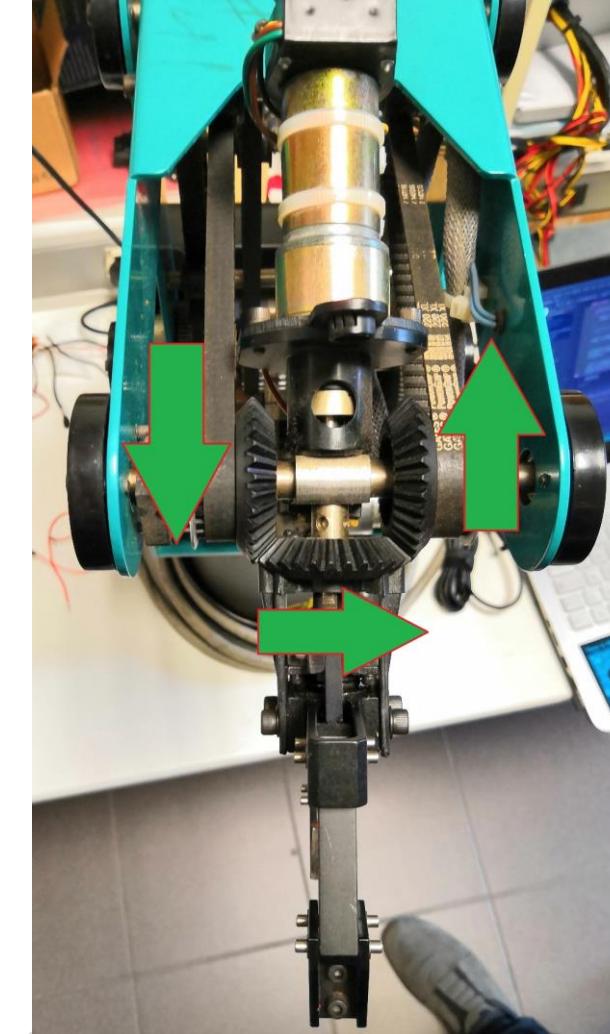
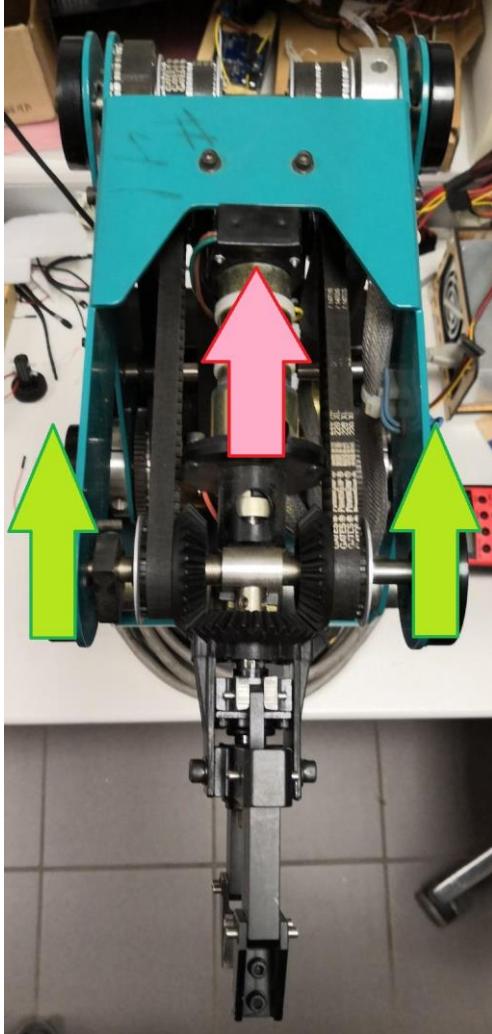


Complicata la taratura con Ziegler-Nichols.
Abbiamo optato per una taratura sperimentale che
si comportasse in generale bene.



COPPIA DIFFERENZIALE

- La coppia deve ruotare contemporaneamente per variare il ‘Pitch’, cioè l’angolo θ intorno all’asse y.
- La coppia deve ruotare in senso opposto per variare il ‘Roll’, cioè l’angolo ψ intorno all’asse x.



Oggetto PID nel codice

```
Kp          Ki          Kd          dz          dir          Maxsat          minsat
↓            ↓            ↓           ↓           ↓           ↓             ↓
pidM[cMot1] = new PIDScorbot(0.00055,0.0000000015,10,50,false,.8,0.02);
pidM[cMot2] = new PIDScorbot(0.00055,0.0000000015,10,50,false,1.0,0.02);
pidM[cMot3] = new PIDScorbot(0.00055,0.0000000015,10,50,true,.8,0.02);
pidM[cMot4] = new PIDScorbot(0.00055,0.0000000015,10,110,true,.8,0.02);
pidM[cMot5] = new PIDScorbot(0.00055,0.0000000015,10,133,false,.8,0.02);
pidM[cMot6] = new PIDScorbot(0.00055,0.0000000015,10,85,false,.8,0.02);
sender = new SpiSend();
memset(&pack,0, sizeof(SPIPACK));
```

Struttura per la
trasmissione dati

Comunicazione
con Arduino

UpdateSat

Define the following update function $UpdateSat(x,dx,a,k,S)$ of a state x given an increment dx with a saturation level S of the term $a + kx$ such as

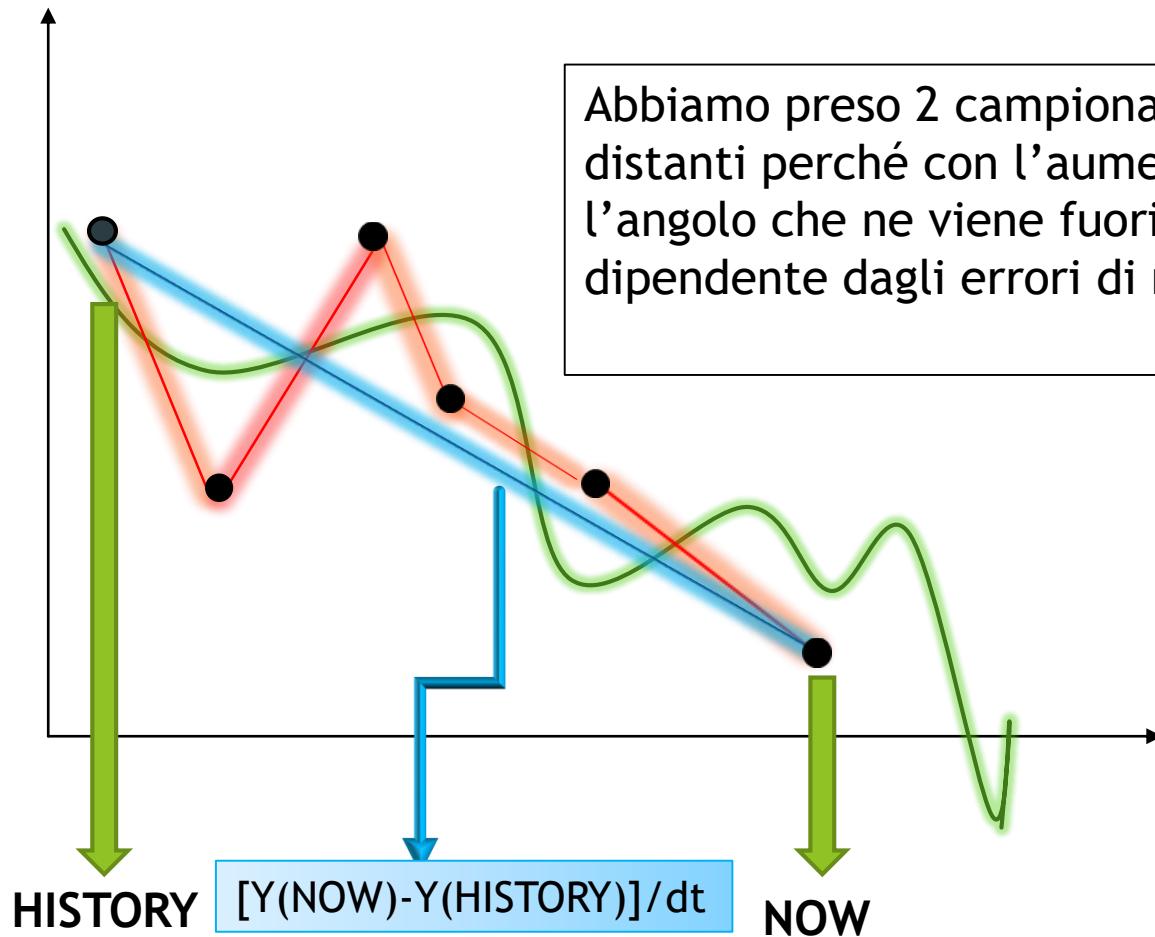
```
 $\hat{x} = x + dx$ 
 $\theta = S - |a + k\hat{x}|$ 
 $if(\theta <= 0) \quad \% a + kx is over the saturation threshold S$ 
     $if(|a + k\hat{x}| < |a + kx|) \quad \% the new value \hat{x} is profitable$ 
         $x = x + dx$ 
     $else$ 
         $x = x + sign(dx) max(0, (S - |a + kx|)/k)$ 
     $end$ 
 $else$ 
     $x = x + dx$ 
 $end$ 
```

- ▶ A differenza di questa UpdateSat,abbiamo aggiunto un parametro ‘s’,di “sotto saturazione”.
- ▶ L’originale UpdateSat controllava soltanto il caso in cui si saturava in eccesso,mentre la nuova controlla anche se l’incremento dell’integrale è troppo piccolo.



if $|x+dx|$ is less than s return 0

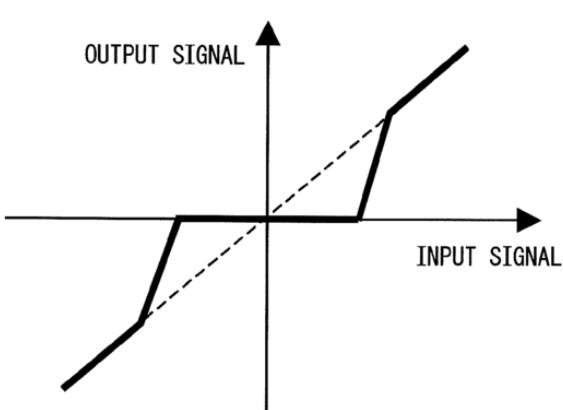
La Derivata sperimentale



Abbiamo preso 2 campionamenti abbastanza distanti perché con l'aumentare del tempo l'angolo che ne viene fuori è molto meno dipendente dagli errori di misura.

Deadzone

FIG.8



La deadzone è una problematica dei motori.

E' un intervallo per il quale il sistema restituisce 0 in output per ogni input ricevuto.

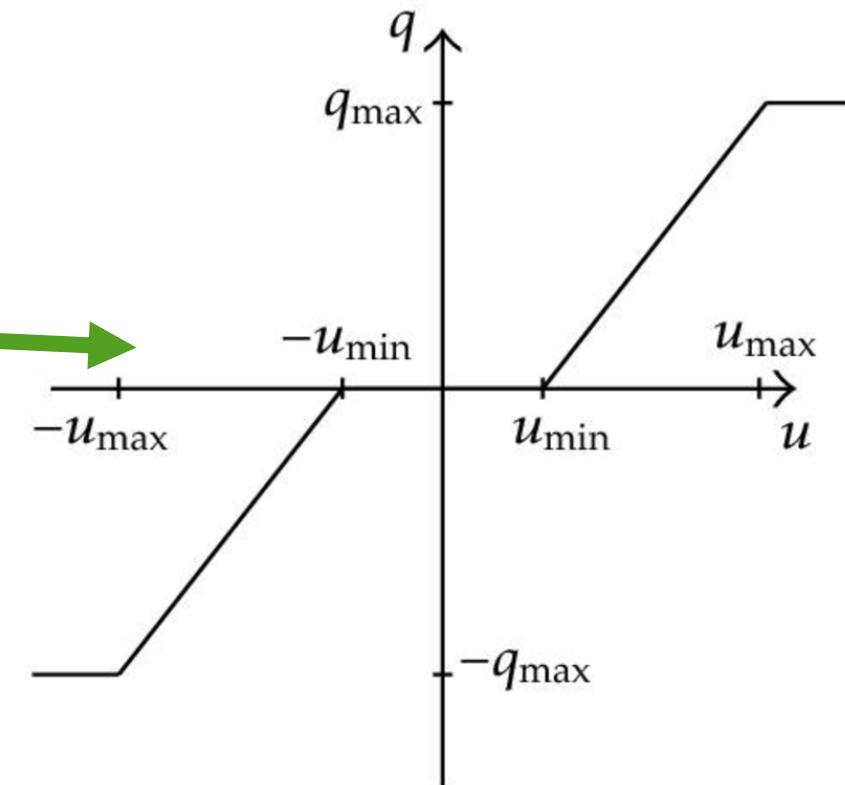
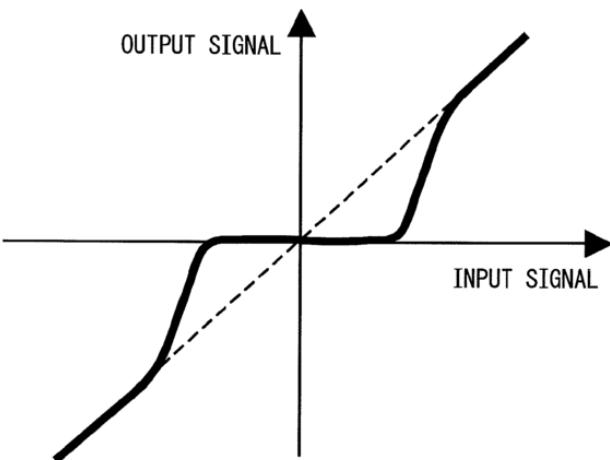
Al di fuori della deadzone ci sarà un'uscita diversa da 0.

La figura 8 è una approssimazione mentre la fig9 è una visione realistica.

Per semplicità di codice abbiamo approssimato come qui

Si può vedere la saturazione unita alla deadzone.

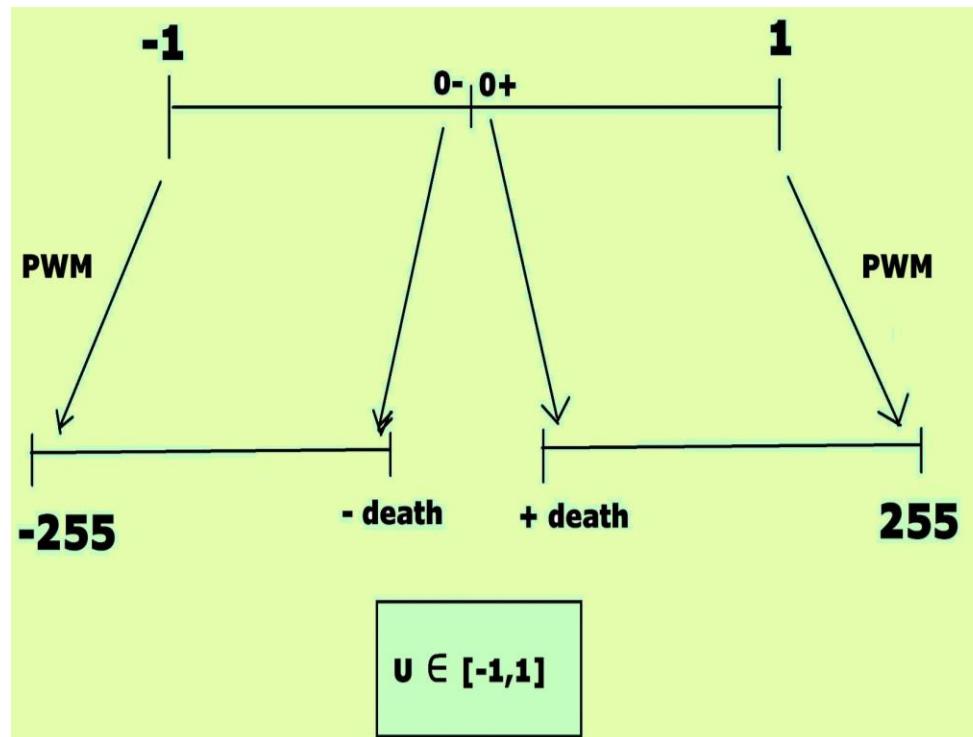
FIG.9



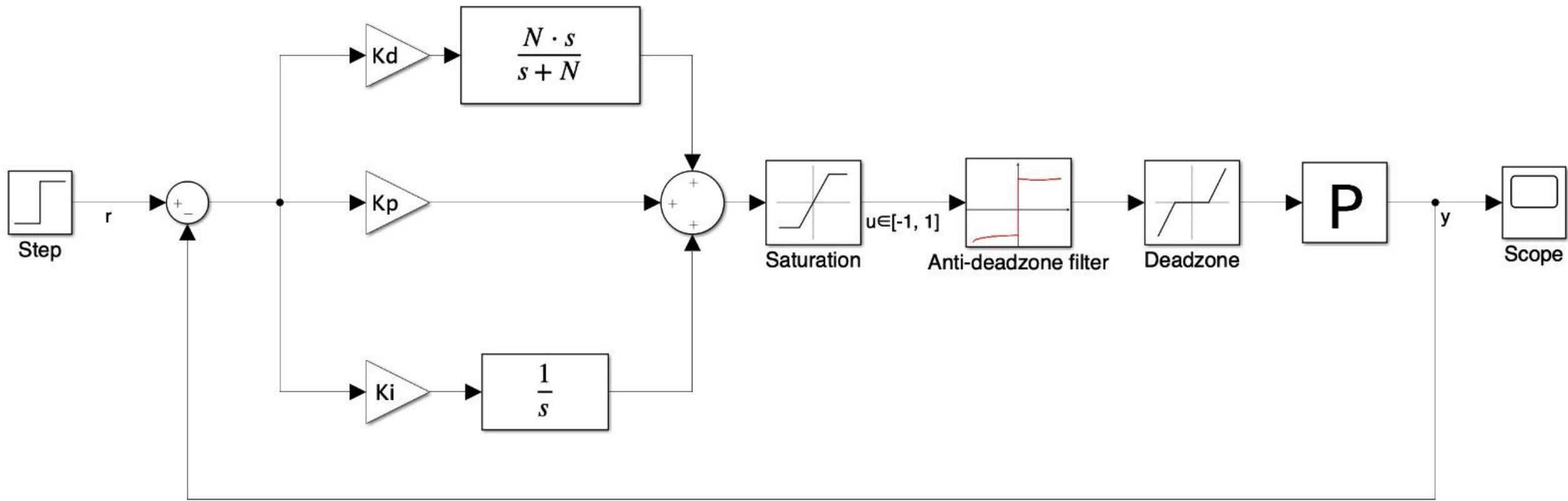
Filtro antiDeadzone

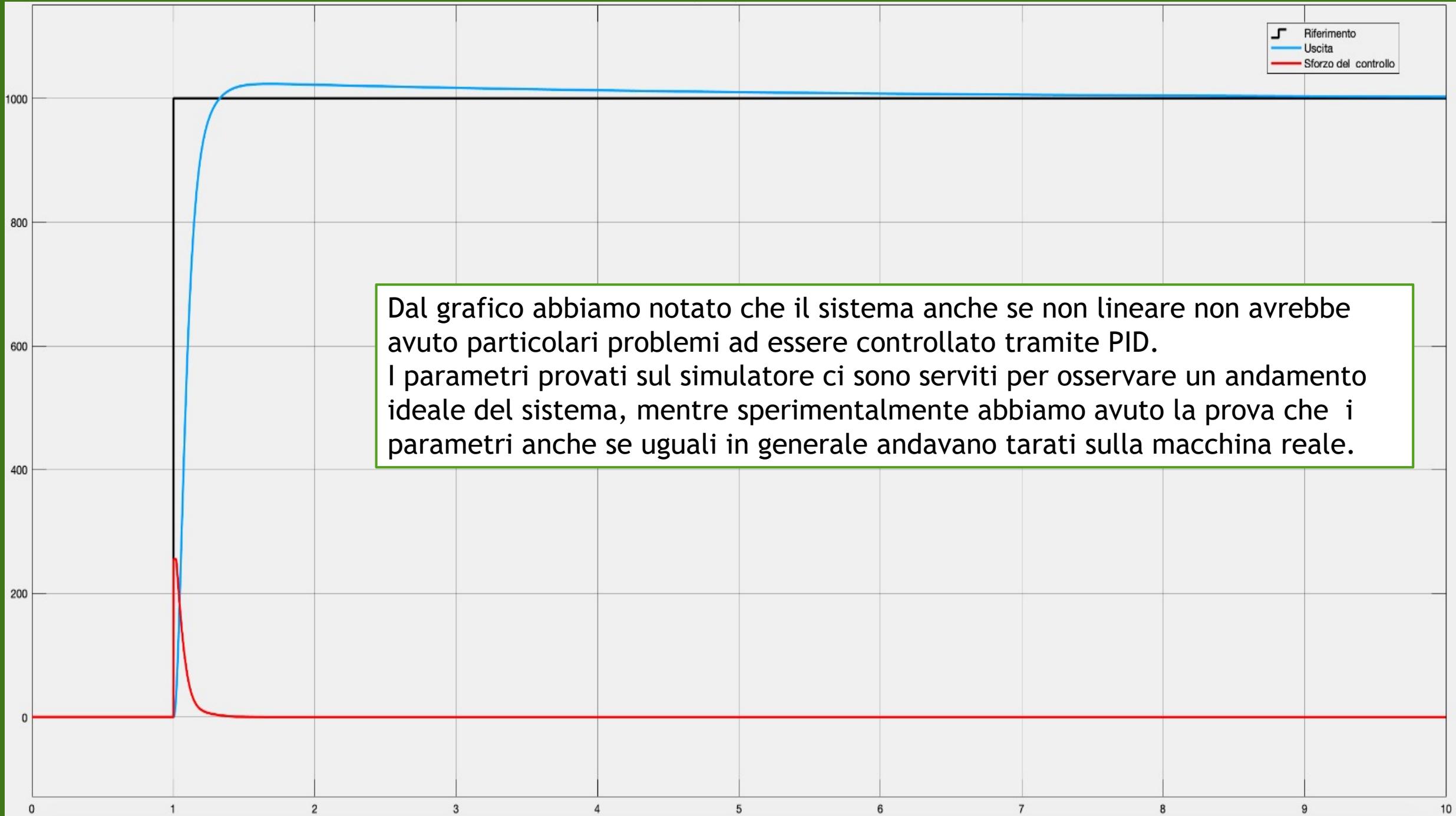
```
if(fabsf(vOut)<this->CONTROL_DEADZONE)
{
    return ss; //soft stop
} else {
    if(vOut>0) return int(fmap(vOut, 0.0, 1.0, this->MOTOR_DEADZONE, 255)+0.5);
    else
    {
        return int(-fmap(-vOut, 0.0, 1.0, this->MOTOR_DEADZONE, 255)-0.5);
    }
}
```

Per risolvere il chattering abbiamo introdotto una piccola deadzone (min_{sat}) nella quale comunque fermiamo il motore.

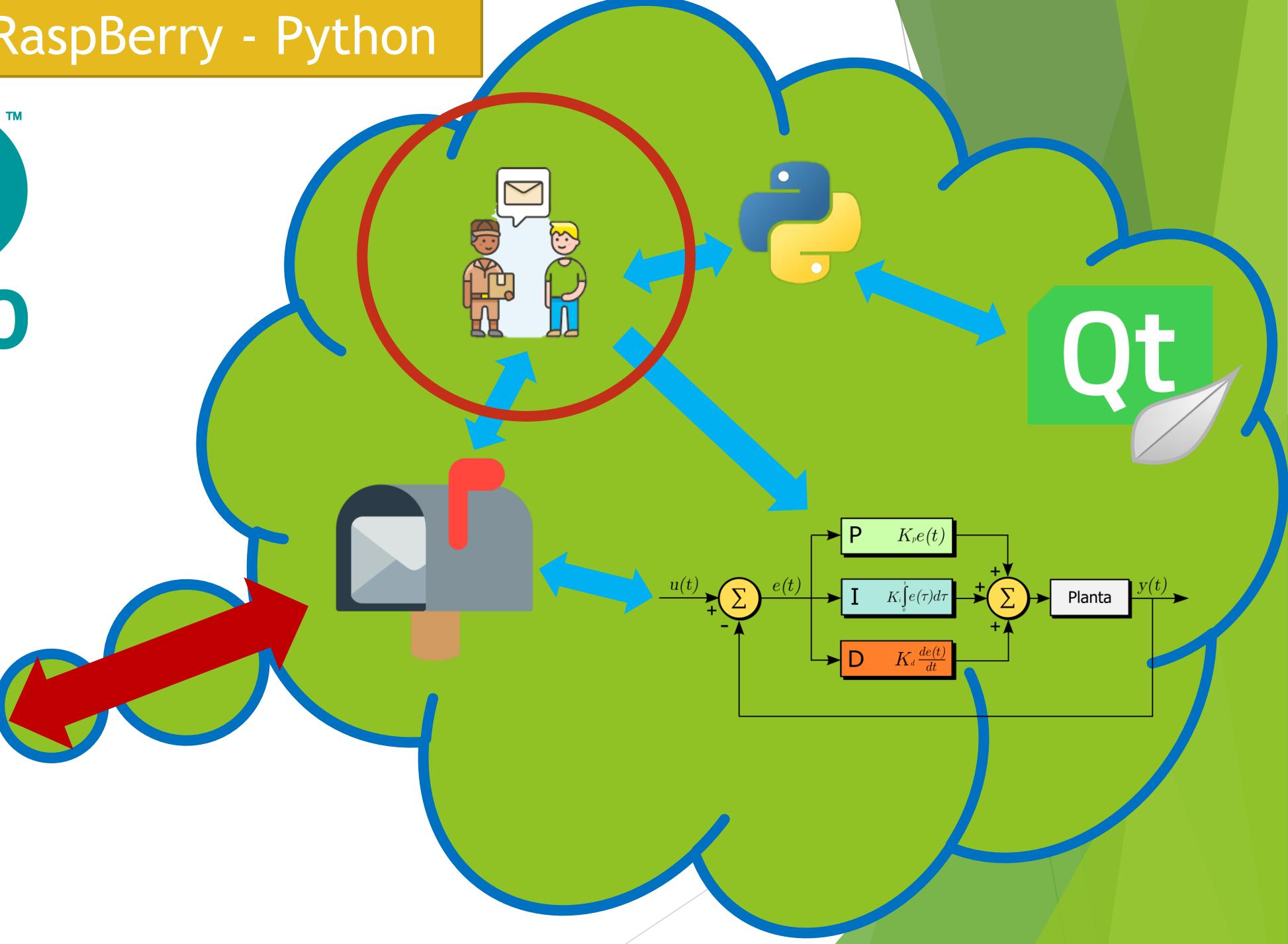
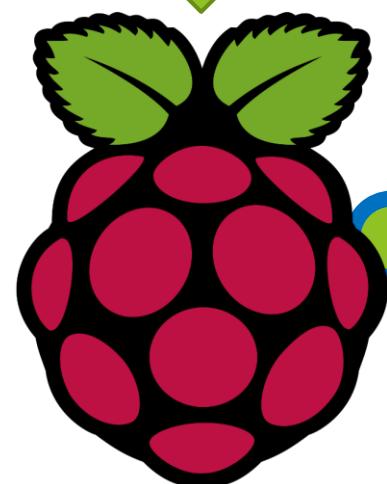


Schema pid completo su Simulink

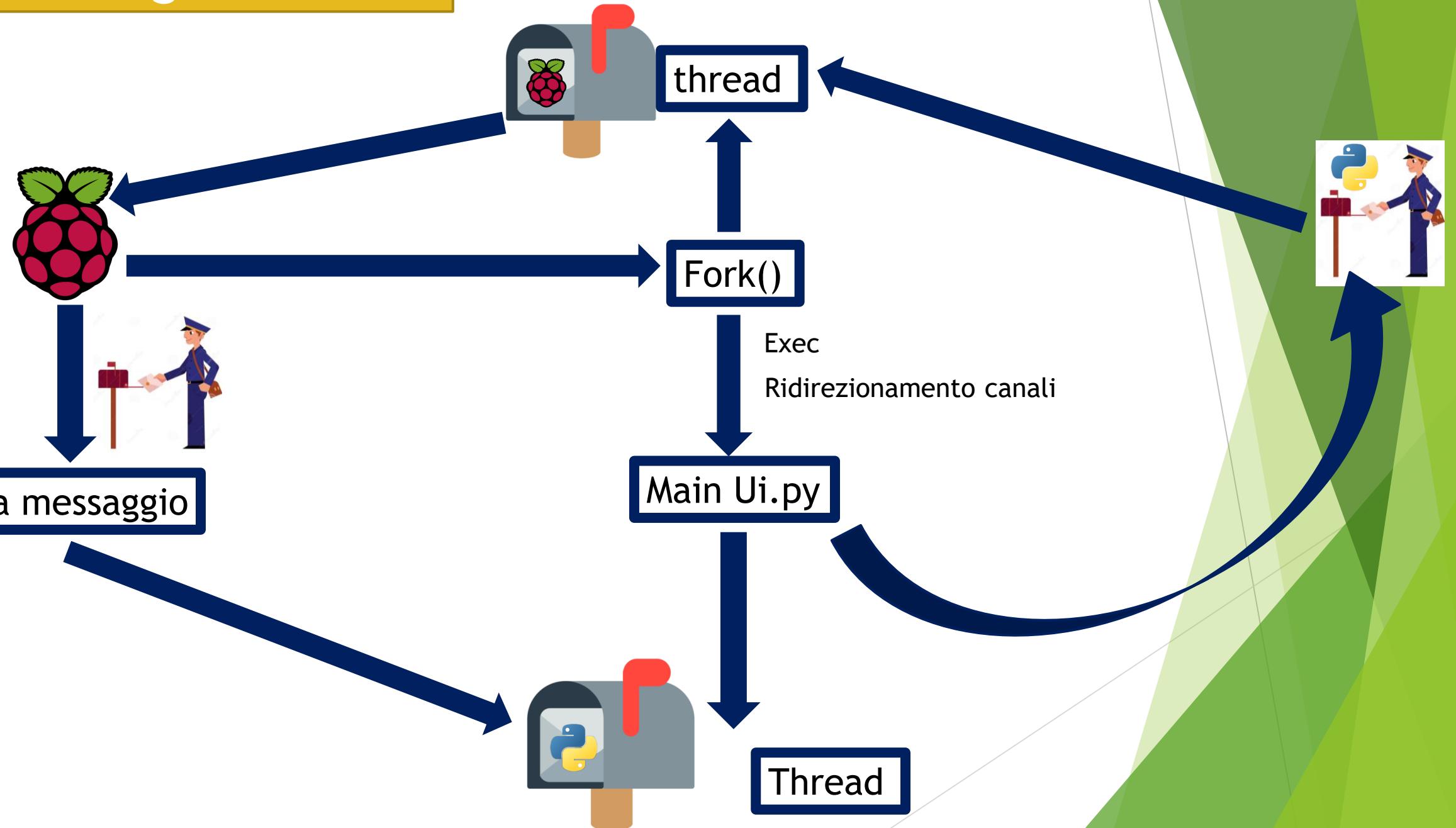




Comunicazione Raspberry - Python

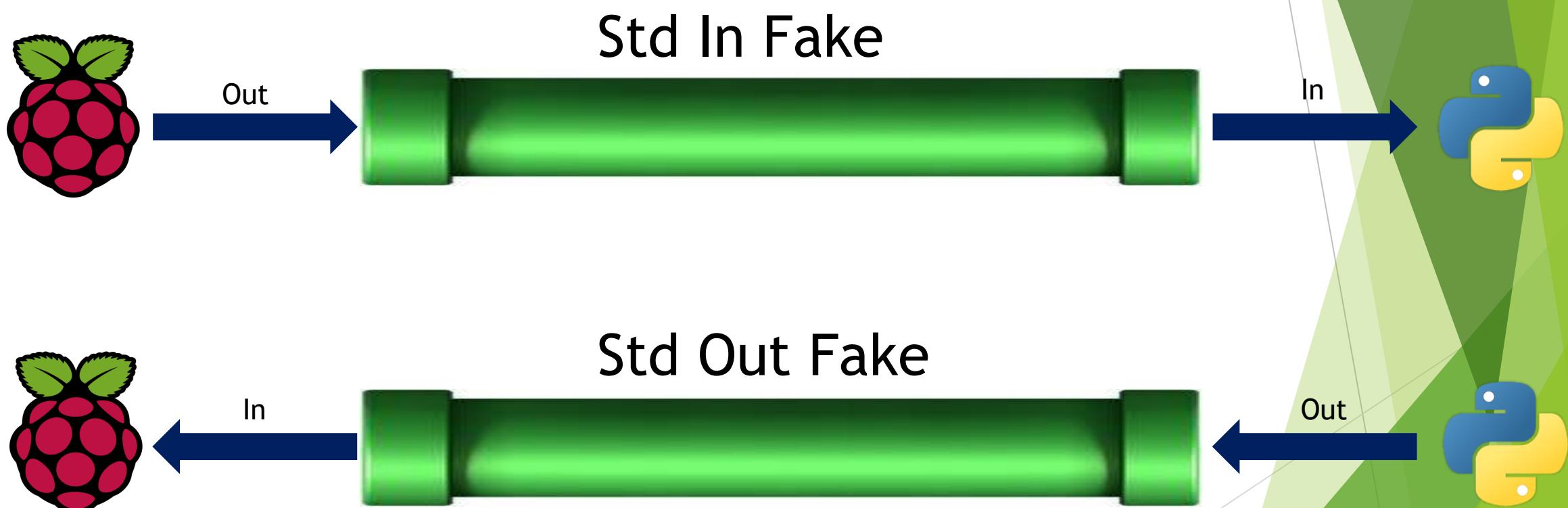


Albero degli eventi



Ridirezionamento Canali

PIPE()



Funzione invia:

Struttura messaggio:

● e|enc1|enc2|enc3|enc4|enc5|enc6|cor1|cor2|cor3|cor4|cor5|cor6\n

● i|eMax1|eMax2|eMax3|eMax4|eMax5|eMax6|emin1|emin2|emin3|emin4|emin5|emin6|cMax1|cMax2|cMax3|cMax4|cMax5|cMax6\n

Due tipologie:

Messaggio Encoder

Messaggio impostazioni

Funzione invia: Controllo

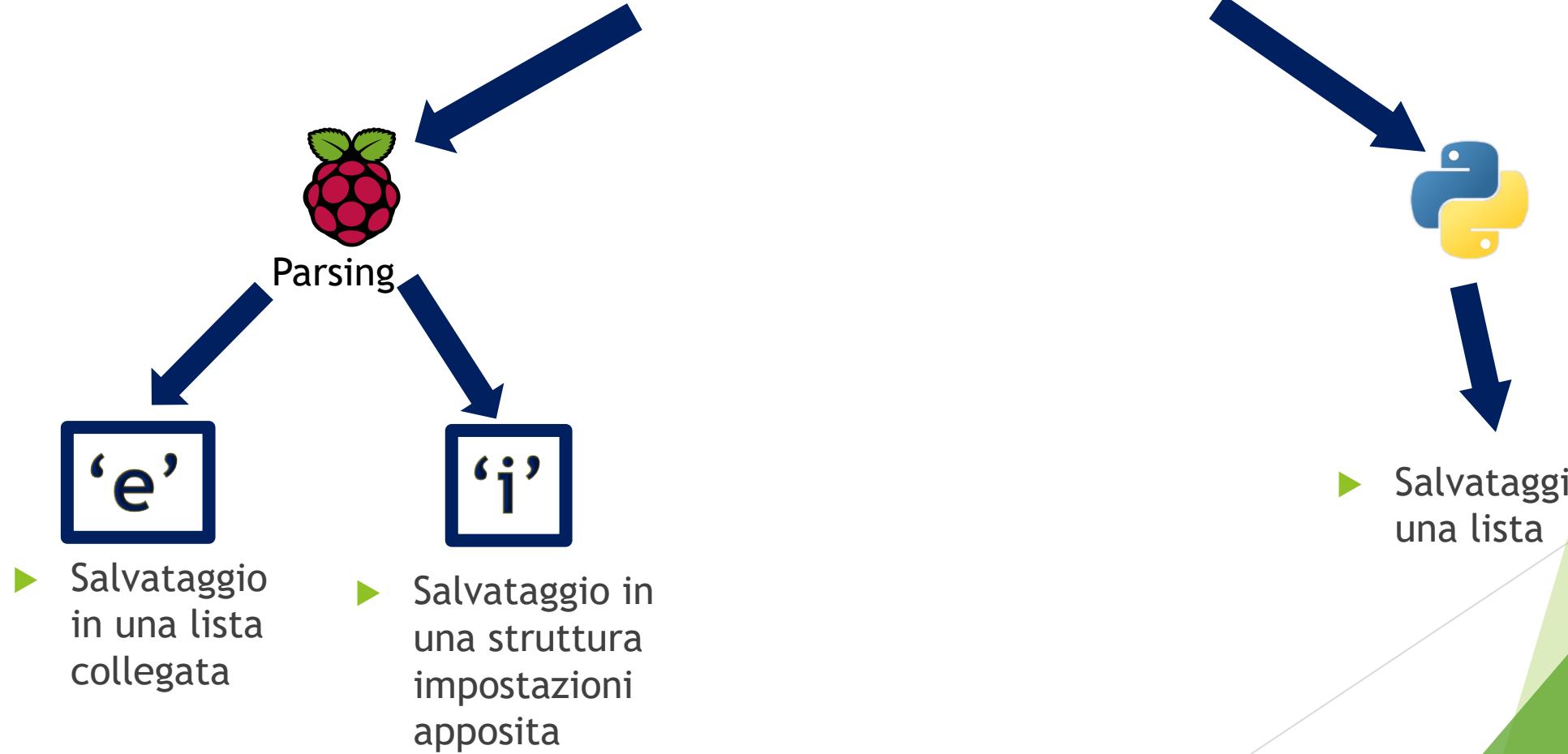
```
int cor,pas; //byte correnti inviati, e byte inviati nel passato
int size=strlen(mes);
cor=dprintf(stdInFake[writeEnd],"%s\n",mes);
if(cor<0){
    return -1;
}
pas=cor;
while(pas<size){
    cor=dprintf(stdInFake[writeEnd],"%s\n",&mes[pas]);
    if(cor<0){
        return -1;
    }
    pas=pas+cor;
}

printf("\nInvio: %s\n",mes);
return 1;
}
```

```
def invia(mes_list,tipo):
    #messaggio e' la stringa da inviare,tipo deve essere="e" o ="i"
    #restituisce -1 se ho fallito l' invio,1 altrimenti
    messaggio=""
    messaggio=messaggio + str(tipo)
    for i in range(0,len(mes_list)):
        messaggio=messaggio + "|" + str(mes_list[i])
    messaggio=messaggio + "|\\n"
    canale_r,canale_w=os.pipe()
    canale_r=sys.stdin
    canale_w=sys.stdout
    cor=canale_w.write(messaggio)
    sys.stdout.flush()
    #cor=sys.stdout.write(messaggio) #cor indica i byte inviati ora
    if(cor<0):
        return -1
    pas=cor #pas tiene traccia di tutti i byte del messaggio inviati

    while(messaggio[pas-1]!='\\n'):
        cor=canale_w.write(messaggio)
        #cor=sys.stdout.write(messaggio[pas:])
        if(cor<0):
            return -1
        pas=pas+cor
return 1
```

Controllo del primo carattere per identificare il tipo di messaggio

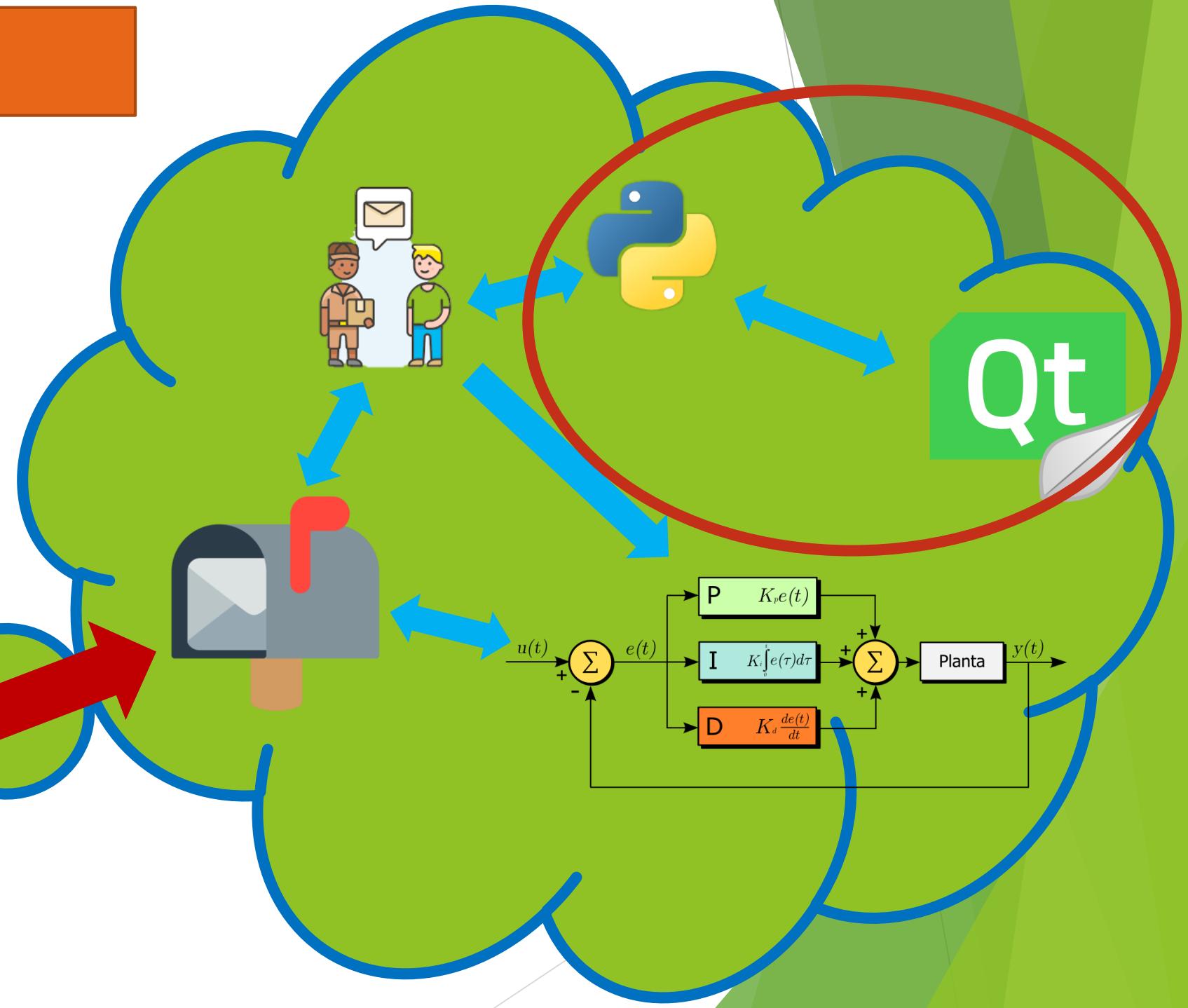
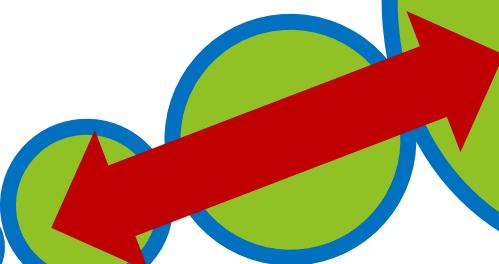
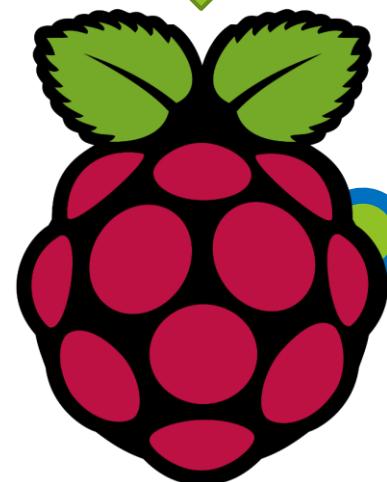


Funzione leggi: Controllo

```
def leggi(self): #legge il messaggio e fa il parsing  
#restituisce la lista dei dati  
pas=0  
messaggio = input()  
lista=[]  
testo=""  
for i in range (0,len(messaggio)):  
    if (i == len(messaggio)-1):  
        testo = testo + messaggio[i]  
        lista.append(testo)  
    if messaggio[i]== '|':  
        lista.append(testo)  
        testo=""  
    else:  
        testo=testo + messaggio[i]  
  
    if (lista[0]=='e'):  
        self.ui.inserisci(lista[1:], 0)  
    else:  
        self.ui.inserisci(lista[1:], 1)  
return
```

```
pas = 0;  
ver = 1; //variabile di controllo  
while (ver) {  
    cor = read(stdOutFake[readEnd], &text[pas], size);  
    if (cor < 0) {  
        printf("ERRORE");  
        exit(-2);  
    }  
    if (text[pas] == '\n') {  
        ver = 0;  
    }  
    pas += cor;
```

Interfaccia Utente



Grafica

- Sviluppo a cura di : Angeloni Ilaria e Menichelli Alberto

Cosa abbiamo fatto:

- Sviluppo di un'interfaccia grafica per comandare un Robot-Scorbot; il codice sviluppato è versatile, permette quindi di settare i dati di lavoro indipendentemente dallo Scorbot fisicamente utilizzato così da poter riutilizzare l'interfaccia creata anche su altri dispositivi dello stesso tipo.

Come :

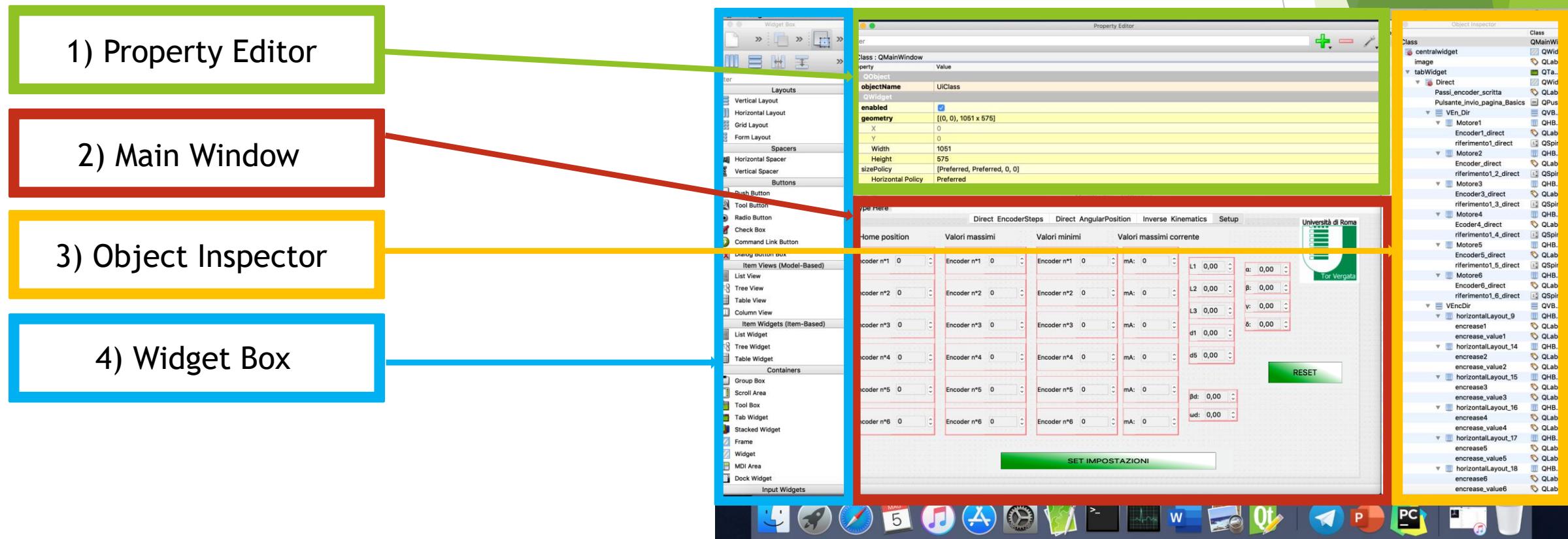
- Per lo sviluppo dell'interfaccia grafica sono stati utilizzati principalmente i seguenti due programmi:
- **Qt designer**: componente del toolset della libreria grafica Qt, con il quale è stata creata la parte visuale.
- **PyCharm** : ambiente di sviluppo per Python, utilizzato per scrivere il codice che rende funzionante le pagine create tramite Qt Creator.



Qt Designer



- ▶ La parte visuale è stata sviluppata con Qt Designer, uno strumento per la creazione e progettazione di interfacce grafiche.
- ▶ La schermata principale è divisa in più parti; di seguito elenchiamo quelle più importanti e che sono state più utili per il progetto.



Qt creator: Pulsanti principali

- ▶ **Spin Box e Double Spin Box** : per l'inserimento di valori interi e float rispettivamente.



- ▶ **Label** : per l'inserimento di valori che saranno visti come ‘stringhe’ all'interno del codice.



- ▶ **Push Button** : utilizzato per la creazioni di pulsanti nella schermata.



- ▶ **Horizontal Slider**: utilizzato come barra % per l'inserimento di valori tramite lo spostamento di un cursore.

Horizontal Slider

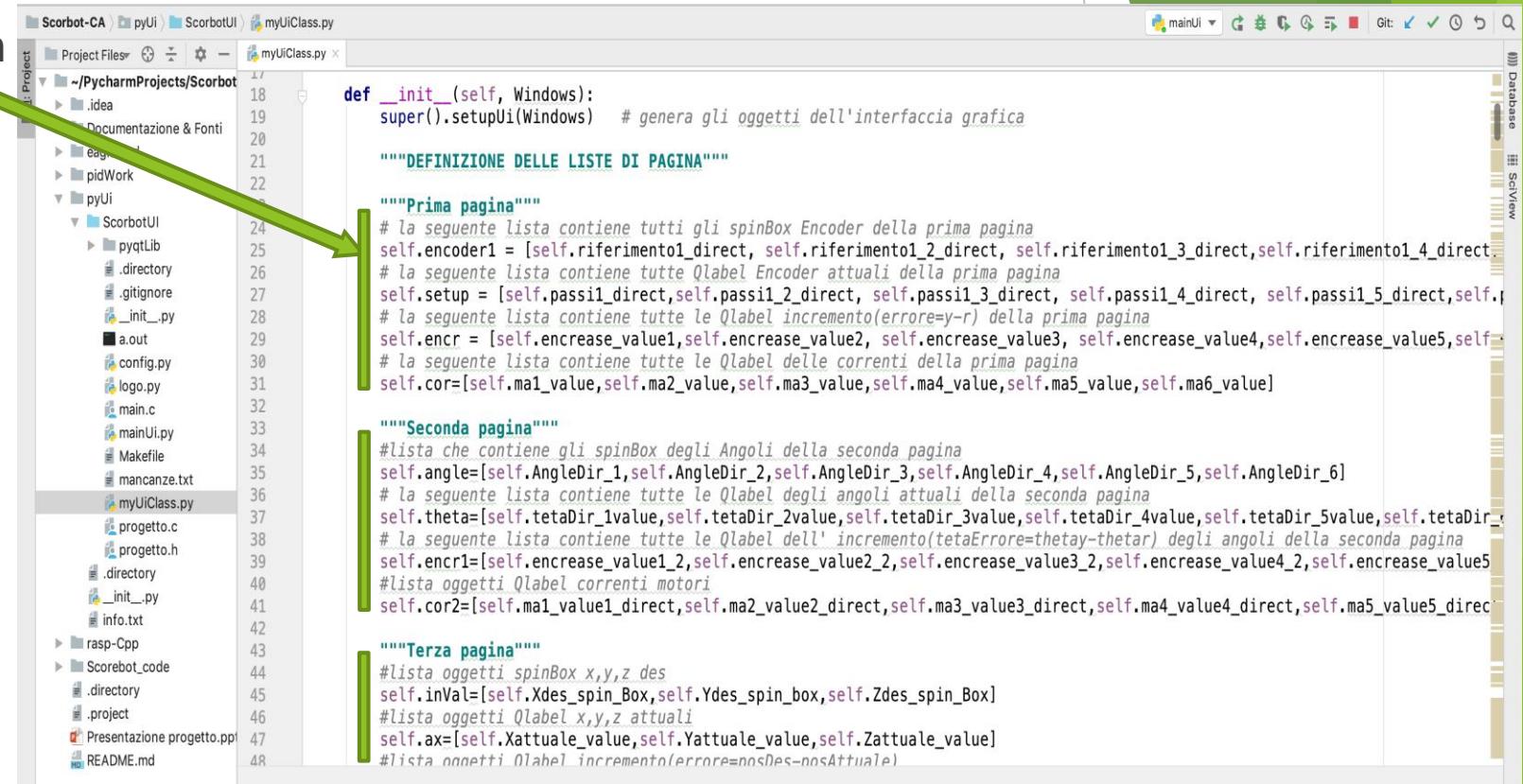


- ▶ **Radio Button**: utilizzato per poter scegliere tra diverse modalità:



Struttura del codice : Definizione delle liste di pagina

- ▶ Ogni pagina è identificata da un certo numero di liste che contengono gli oggetti presenti nella schermata stessa.
- ▶ L'inserimento degli oggetti all'interno di liste, definite direttamente nel costruttore, ha permesso uno sviluppo più snello del codice potendo richiamare ogni volta una lista, piuttosto che un oggetto.



The screenshot shows the PyCharm IDE interface with the project structure on the left and the code editor on the right. The project structure includes files like mainUi.py, logo.py, main.c, mainUi.py, Makefile, mancanze.txt, myUiClass.py, progetto.c, progetto.h, rasp-Cpp, Scorebot_code, .directory, .project, Presentazione progetto.ppt, and README.md. The code editor displays myUiClass.py with the following content:

```
def __init__(self, Windows):
    super().setupUi(Windows) # genera gli oggetti dell'interfaccia grafica

"""DEFINIZIONE DELLE LISTE DI PAGINA"""

"""Prima pagina"""
# la seguente lista contiene tutti gli spinBox Encoder della prima pagina
self.encoder1 = [self.riferimento1_direct, self.riferimento1_2_direct, self.riferimento1_3_direct, self.riferimento1_4_direct,
                 self.riferimento1_5_direct, self.riferimento1_6_direct]
# la seguente lista contiene tutte Qlabel Encoder attuali della prima pagina
self.setup = [self.passi1_direct, self.passi1_2_direct, self.passi1_3_direct, self.passi1_4_direct, self.passi1_5_direct, self.passi1_6_direct]
# la seguente lista contiene tutte le Qlabel incremento(error=y-r) della prima pagina
self.enqr = [self.increase_value1, self.increase_value2, self.increase_value3, self.increase_value4, self.increase_value5, self.increase_value6]
# la seguente lista contiene tutte le Qlabel delle correnti della prima pagina
self.cor=[self.ma1_value, self.ma2_value, self.ma3_value, self.ma4_value, self.ma5_value, self.ma6_value]

"""Seconda pagina"""
#lista che contiene gli spinBox degli Angoli della seconda pagina
self.angle=[self.AngleDir_1, self.AngleDir_2, self.AngleDir_3, self.AngleDir_4, self.AngleDir_5, self.AngleDir_6]
# la seguente lista contiene tutte le Qlabel degli angoli attuali della seconda pagina
self.theta=[self.tetaDir_1value, self.tetaDir_2value, self.tetaDir_3value, self.tetaDir_4value, self.tetaDir_5value, self.tetaDir_6value]
# la seguente lista contiene tutte le Qlabel dell' incremento(tetaErrore=thetaY-thetaR) degli angoli della seconda pagina
self.enqr1=[self.increase_value1_2, self.increase_value2_2, self.increase_value3_2, self.increase_value4_2, self.increase_value5_2, self.increase_value6_2]
#lista oggetti Qlabel correnti motori
self.cor2=[self.ma1_value1_direct, self.ma2_value2_direct, self.ma3_value3_direct, self.ma4_value4_direct, self.ma5_value5_direct, self.ma6_value6_direct]

"""Terza pagina"""
#lista oggetti spinBox x,y,z des
self.inVal=[self.Xdes_spin_Box, self.Ydes_spin_box, self.Zdes_spin_Box]
#lista oggetti Qlabel x,y,z attuali
self.ax=[self.Xattuale_value, self.Yattuale_value, self.Zattuale_value]
#lista oggetti Qlabel incremento(error=nowDes-nowAttuale)
```

A large green arrow points from the text "Ogni pagina è identificata da un certo numero di liste che contengono gli oggetti presenti nella schermata stessa." to the section of the code where lists are defined.

Python : Lista Eventi

- ▶ Nella lista eventi sono chiamate le funzioni che svolgeranno i compiti relativi alla pagina di interesse. Ogni schermata ha una funzione che si preoccupa di determinate azioni e per richiamarla è sufficiente premere il pulsante Enter della pagina che attiverà automaticamente la chiamata di funzione.

```
"""LISTA Eventi"""
#pulsante prima pagina
self.Pulsante_invio_pagina_Basics.clicked.connect(self.encoderValue)
#pulsante seconda pagina
self.PulsanteInvioPAginaAdvance.clicked.connect((self.angleValue))
#pulsante terza pagina
self.PulsanteInvioPAgina_Inverse.clicked.connect((self.inverseValue))
#pulsante quarta pagina (set impostazioni)
self.Home.clicked.connect(self.setup1)
#pulsante quarta pagina (reset impostazioni)
self.reset.clicked.connect(self.default)
```



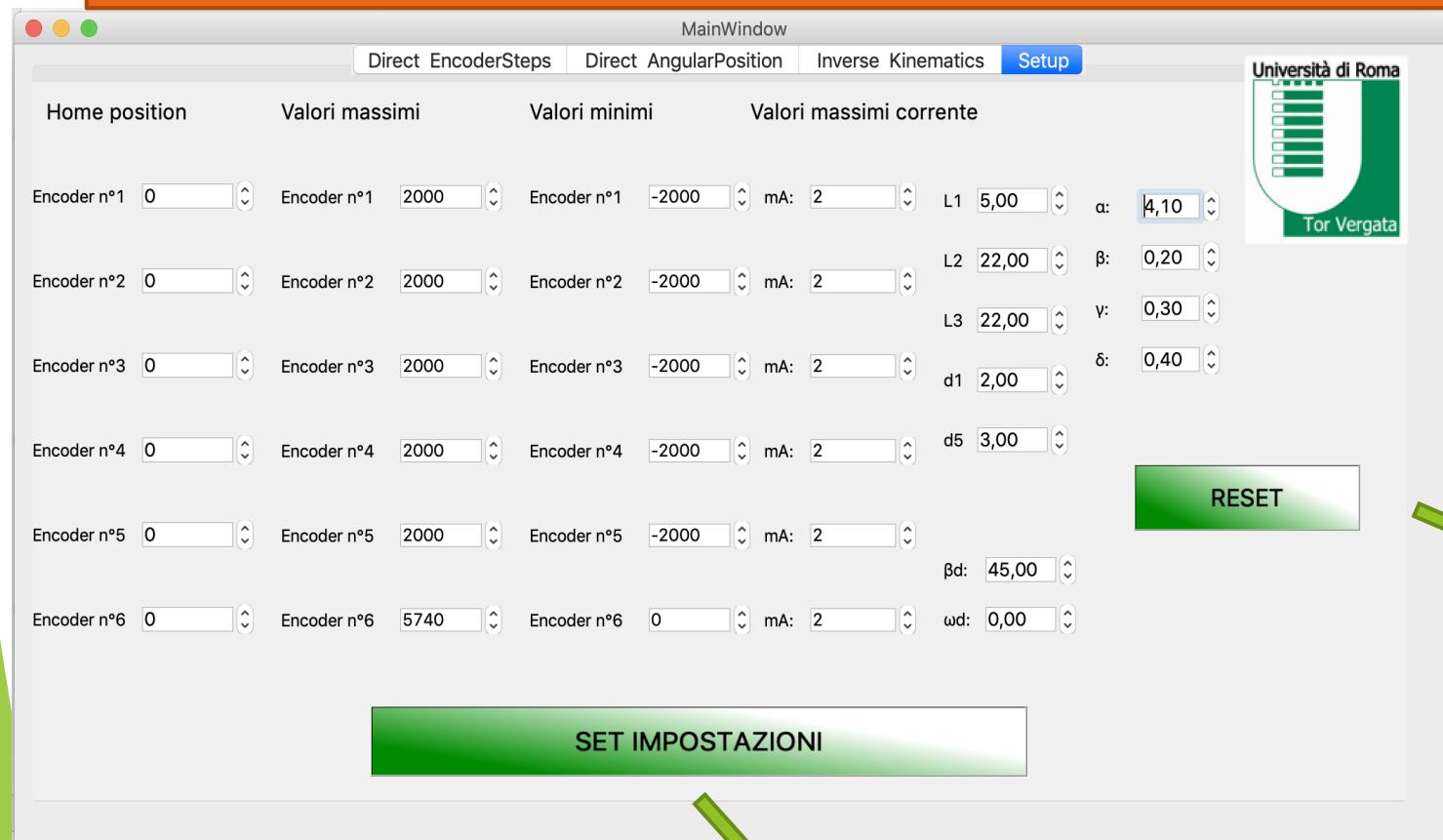
ENTER

Set up : Schermata

- ▶ Nella pagina di Setup vengono inseriti i valori desiderati per la macchina che si ha a disposizione.
- ▶ Al momento dell'Avvio del programma, i vari Spin Box appaiono con il valore con cui questo progetto è stato sviluppato.



Set up : Schermata

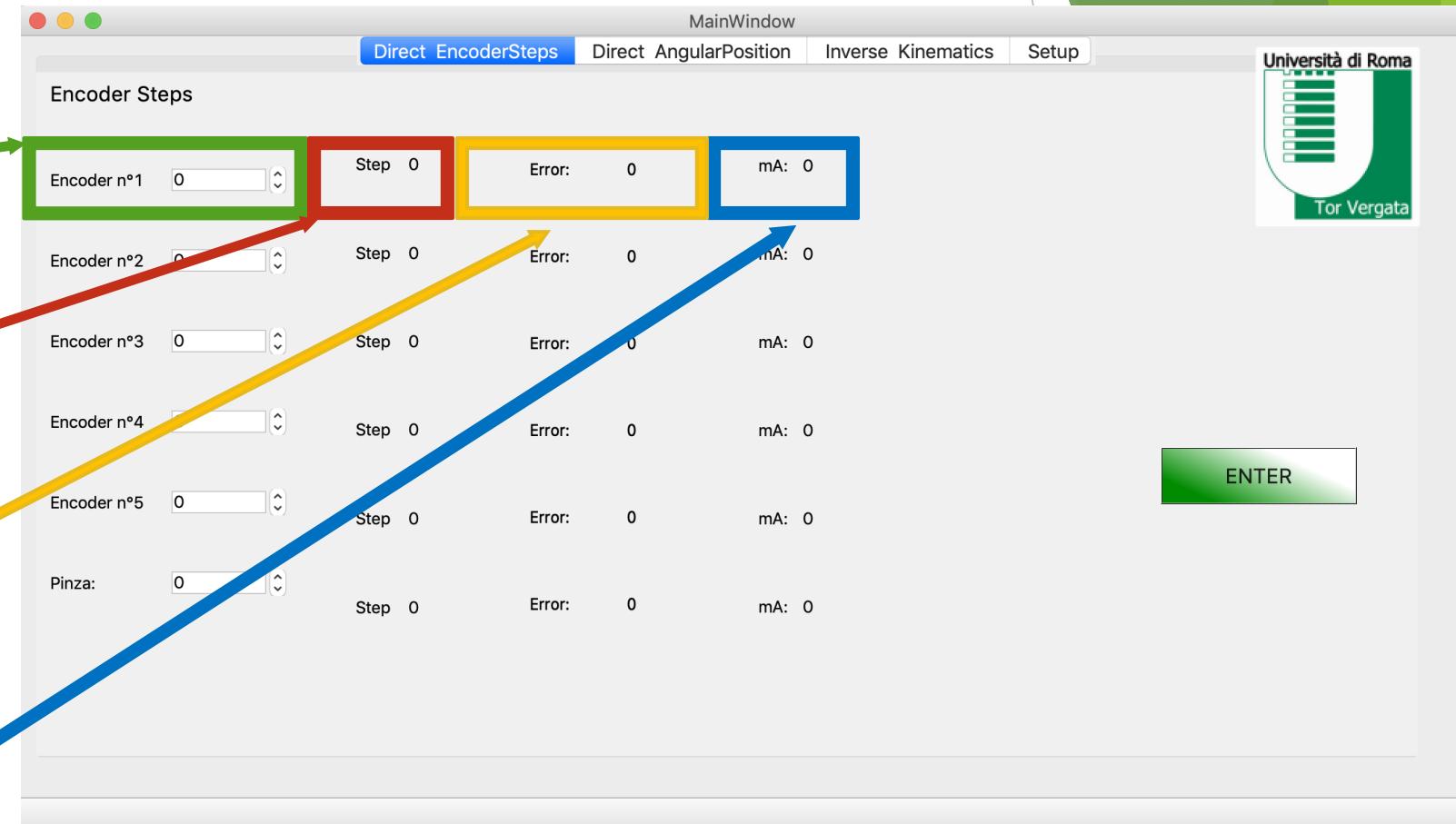


▶ ‘Reset’ riporta il sistema nelle condizioni iniziali.

▶ ‘Set Impostazioni’ aggiorna tutti i valori modificati dall’utente. Il risultato viene salvato in un file ‘config’ rendendo persistenti le impostazioni.

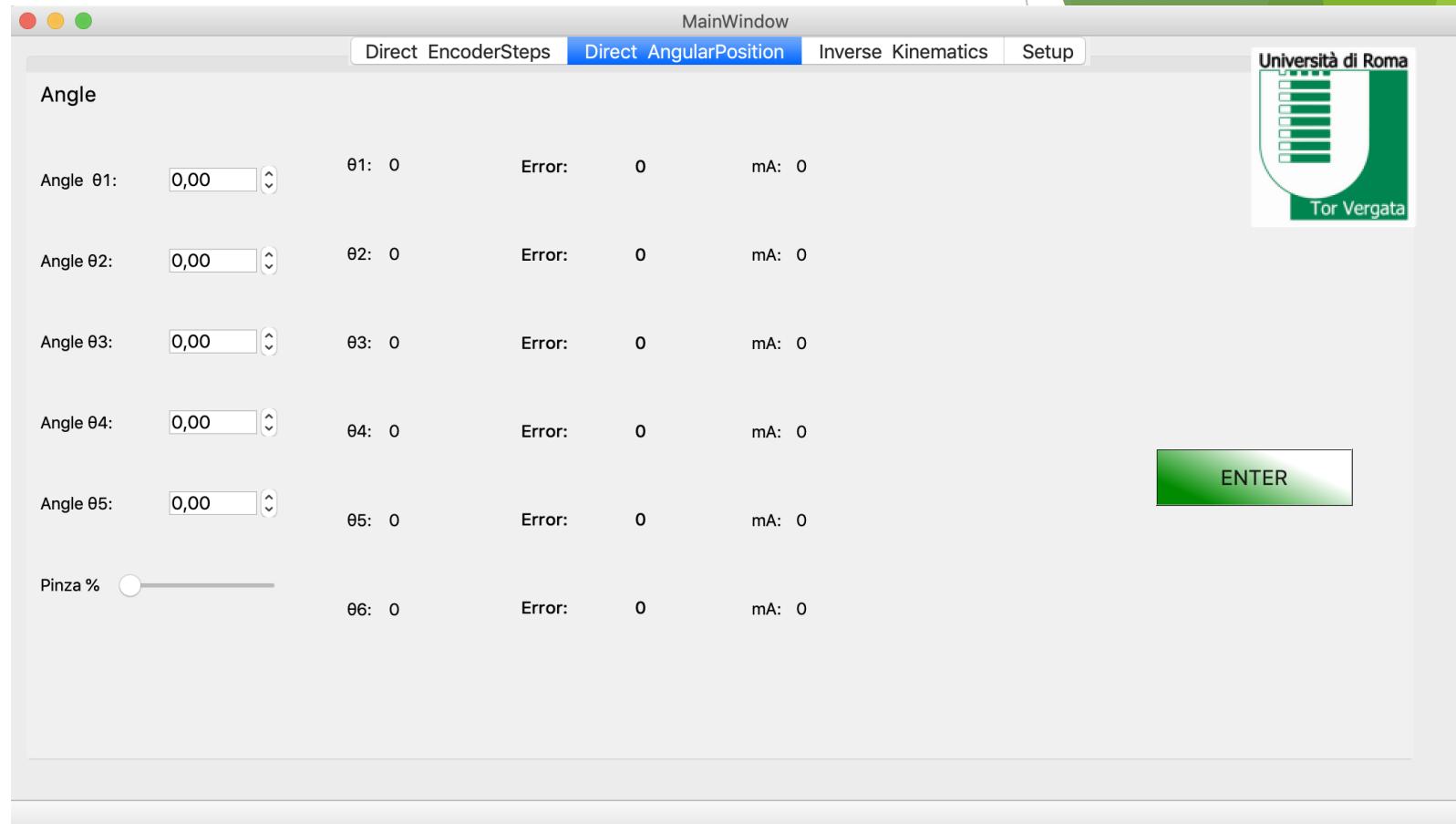
Direct EncoderSteps : Schermata

- In questa Schermata, viene implementata la cinematica diretta attraverso i passi encoder.
- in 'Encoder n°' è possibile inserire i valori a cui si intende portare lo Scrbot.
- In 'Step' leggiamo i valori reali degli encoder, che il robot possiede istante per istante.
- 'Error' rappresenta l'errore, ovvero il margine per arrivare alla posizione desiderata
- In 'mA' è possibile leggere i valori delle correnti utilizzate dai vari motorini.



Direct AngularPosition

- ▶ In questa schermata, viene implementata la cinematica diretta attraverso gli angoli.
- ▶ Stesse modalità di inserimento e lettura ma andremo ad inserire e leggeremo angoli piuttosto che encoder. Quindi, ci sarà una conversione da angoli ad encoder prima di passare i dati alla raspberry.



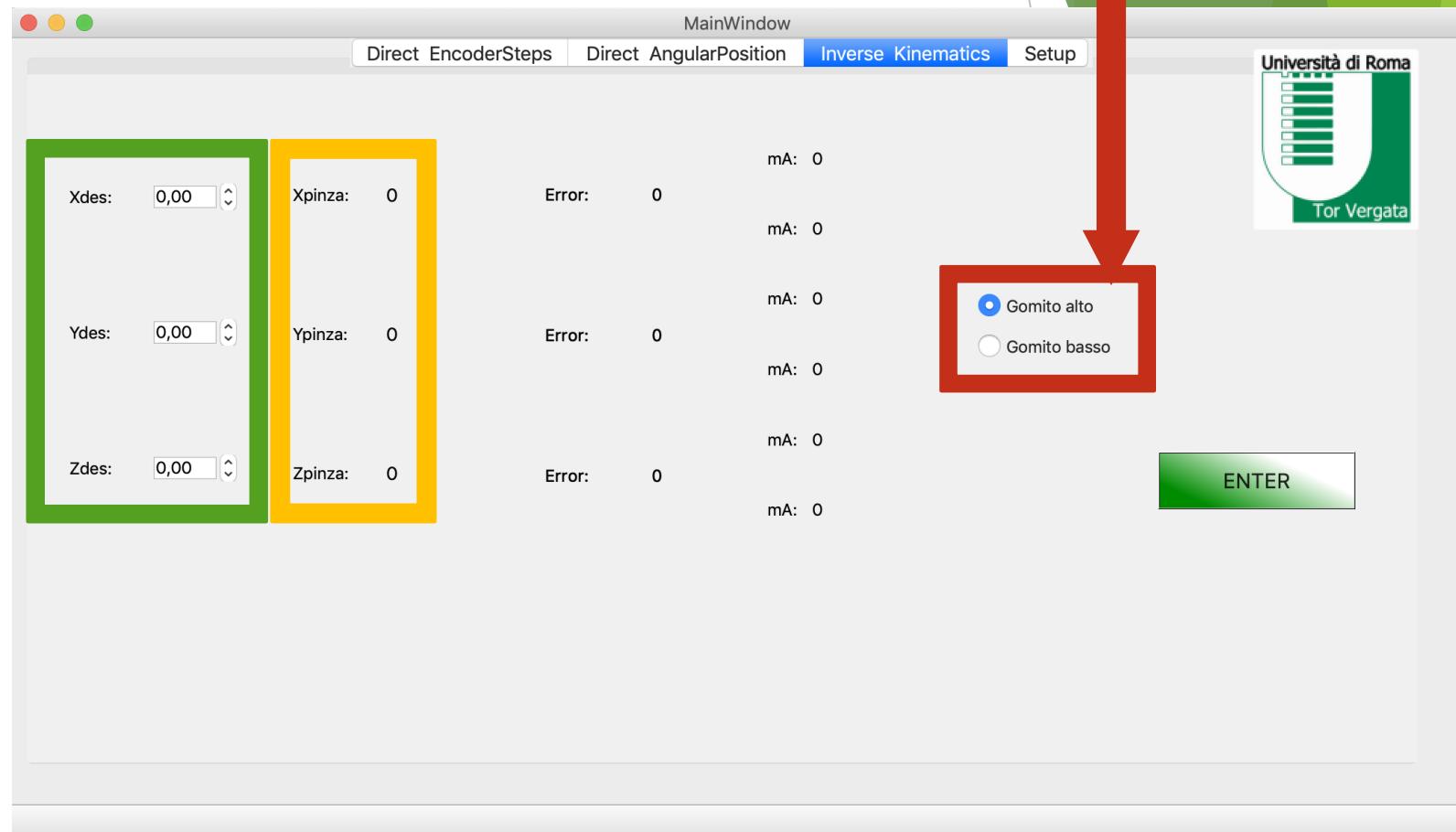
Direct AngularPosition: Formule

$$\begin{array}{lll} \theta_1 & = & -\alpha \cdot p_0 \\ \theta_2 & = & -\beta \cdot p_1 \\ \theta_3 & = & \beta(p_1 + p_2) \\ \theta_4 & = & -\beta \cdot p_2 + \gamma(p_3 - p_4) \\ \theta_5 & = & \delta(p_3 + p_4) \end{array} \quad \iff \quad \begin{array}{lll} p_0 & = & -\theta_1/\alpha \\ p_1 & = & -\theta_2/\beta \\ p_2 & = & (\theta_2 + \theta_3)/\beta \\ p_3 & = & 0.5 [(\theta_2 + \theta_3 + \theta_4)/\gamma + \theta_5/\delta] \\ p_4 & = & 0.5 [\theta_5/\delta - (\theta_2 + \theta_3 + \theta_4)/\gamma] \end{array}$$

Inverse Kinematics

Scelta della
modalità

- ▶ In questa Schermata, viene implementata la cinematica inversa: è quindi possibile inserire le coordinate desiderate per la pinza, con la possibilità anche di scegliere tra la modalità gomito alto, e la modalità gomito basso.



Inverse Kinematics: Formule

$$\left\{ \begin{array}{lcl} \theta_1 & = & \text{atan2} (y_d, x_d), \\ \theta_3 & = & \pm \arccos \frac{A_1^2 + A_2^2 - \ell_2^2 - \ell_3^2}{2\ell_2\ell_3}, \\ \theta_2 & = & \text{atan2} ((\ell_2 + \ell_3 C_3)A_2 - \ell_3 S_3 A_1, (\ell_2 + \ell_3 C_3)A_1 + \ell_3 S_3 A_2), \\ \theta_4 & = & \beta_d - \theta_2 - \theta_3 - \pi/2, \\ \theta_5 & = & \omega_d. \end{array} \right.$$

ove C_3 indica la funzione $\cos \theta_3$, S_3 indica la funzione $\sin \theta_3$, ed inoltre

$$\begin{aligned} A_1 &= x_d C_1 + y_d S_1 - d_5 \cos(\beta_d) - \ell_1 \\ A_2 &= d_1 - z_d - d_5 \sin(\beta_d). \end{aligned}$$

