

## E-bike : motori

Riccardo Di Battista, Lorenzo Meloccaro  
Antonio Brigida, Lorenzo D'Agostino

anno accademico 2019-2020



# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Hardware</b>	<b>4</b>
2.1	YRDKRX63N . . . . .	4
2.1.1	Modifiche hardware . . . . .	5
2.2	Attuatori . . . . .	6
2.2.1	Motore trazione . . . . .	6
2.2.2	Motore sterzata . . . . .	7
2.2.3	Driver motori . . . . .	8
2.3	Trasduttori . . . . .	9
2.3.1	Encoder . . . . .	9
2.3.2	Adattatore encoder . . . . .	10
2.3.3	IMU : giroscopio e accelerometro . . . . .	11
2.3.4	Sensori di corrente . . . . .	12
2.4	Collegamenti . . . . .	13
2.4.1	Collegamenti comuni . . . . .	13
2.4.2	Alimentazione . . . . .	14
<b>3</b>	<b>Il nostro contributo al progetto</b>	<b>15</b>
3.1	Introduzione alle problematiche . . . . .	15
3.2	PWM : Pulse Width Modulation . . . . .	15
3.2.1	Duty Cycle . . . . .	16
3.2.2	Generazione PWM analogico . . . . .	17
3.2.3	Generazione PWM digitale . . . . .	18
3.3	Controllo : PID . . . . .	19
3.4	Software . . . . .	20
3.4.1	Motori : pwm.h/pwm.c . . . . .	20
3.4.2	PID : PID.h/PID.c . . . . .	22
3.4.3	Main program : algoritmo di controllo motore . . . . .	23
3.4.4	Controllo in cascata . . . . .	24
<b>4</b>	<b>Modello matematico</b>	<b>26</b>
4.1	Premesse . . . . .	26
4.2	Cinematica . . . . .	26
4.3	Dinamica laterale . . . . .	27
4.4	Funzione di trasferimento del sistema . . . . .	28
4.5	Controllore proporzionale . . . . .	30
4.6	Controllore PI . . . . .	31
4.7	La retroazione intrinseca del sistema . . . . .	32

# 1 Introduzione

La seguente relazione tratta lo sviluppo di un firmware per il controllo della stabilità di una bicicletta tramite l'utilizzo della scheda di sviluppo Renesas : RDKRX63N. I vari gruppi si sono focalizzati sulla comprensione ed adattamento di codici riguardanti i vari sensori e la generazione dei PWM per il pilotaggio dei motori, inoltre si è studiato il modello matematico della bicicletta come pendolo inverso. Ogni sezione della relazione conterrà una descrizione dettagliata dell'hardware utilizzato e tutti i documenti utili alla comprensione dello stesso.



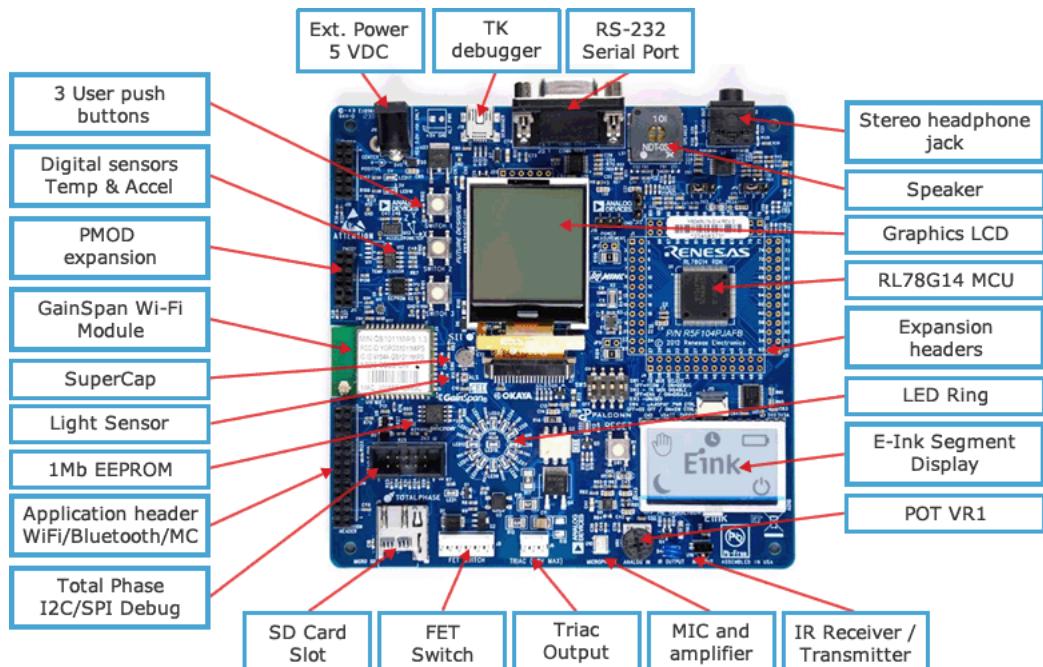
## 2 Hardware

Di seguito una descrizione della componentistica adottata per lo sviluppo del software.

### 2.1 YRDKRX63N

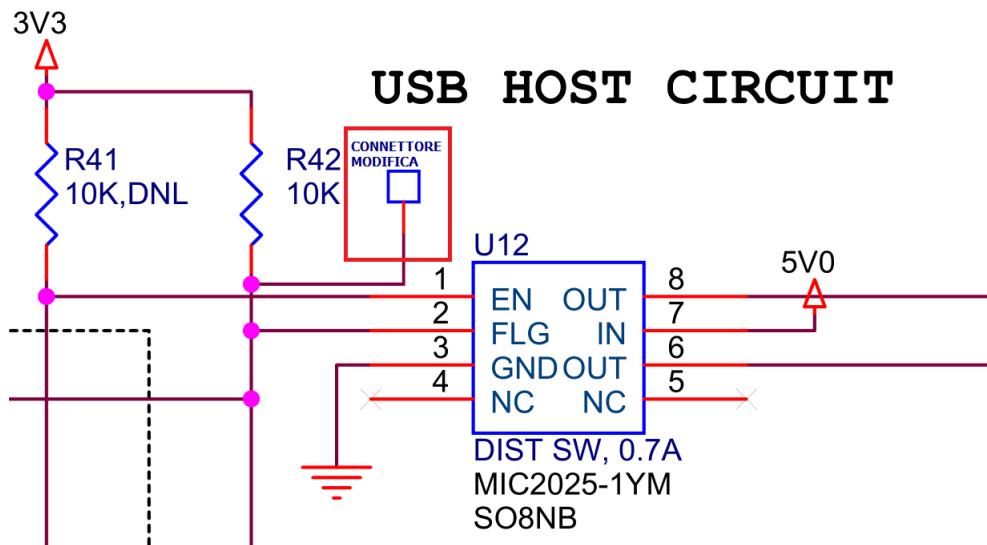
La scheda utilizzata è un kit di sviluppo che si basa sul micro-controllore Renesas RX63N. Esso fornisce alte prestazioni che comprendono :

- 32-bit MCU capace di operare oltre i 100 MHz;
- Oltre 21 canali per ADC a 12-bit e oltre 2 canali per DAC- unità Timers;
- MTU2 [Multi Timer Unit Function], con funzioni di: input capture / output compare / counter clearing per generazione di segnali PWM, e controllo motori;
- Diverse periferiche per la comunicazione quali : Ethernet, SCI, RSPI, CAN, I2C;

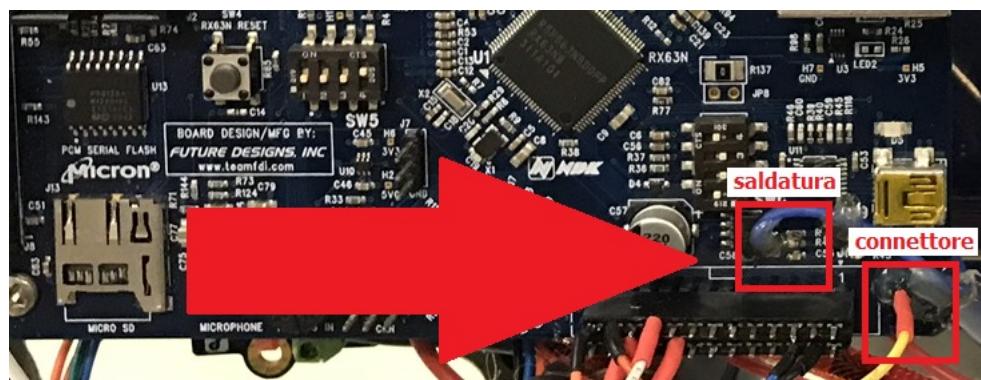


### 2.1.1 Modifiche hardware

Uno degli encoder utilizzati sfrutta la porta P14 che non è disponibile nei connettori esterni dell' YRDKRX63N. Questa porta è collegata a "U12 porta usb-user" che non viene utilizzato per il nostro progetto. Studiando lo schematico siamo riusciti a capire che questa porta è collegata al capo della "resistenza R42", che si trova nelle vicinanze della periferica U12, a questo punto è bastato saldare un connettore jumper femmina al capo sinistro per semplificare il cablaggio ed ottenere un punto per accedere facilmente a quella porta.



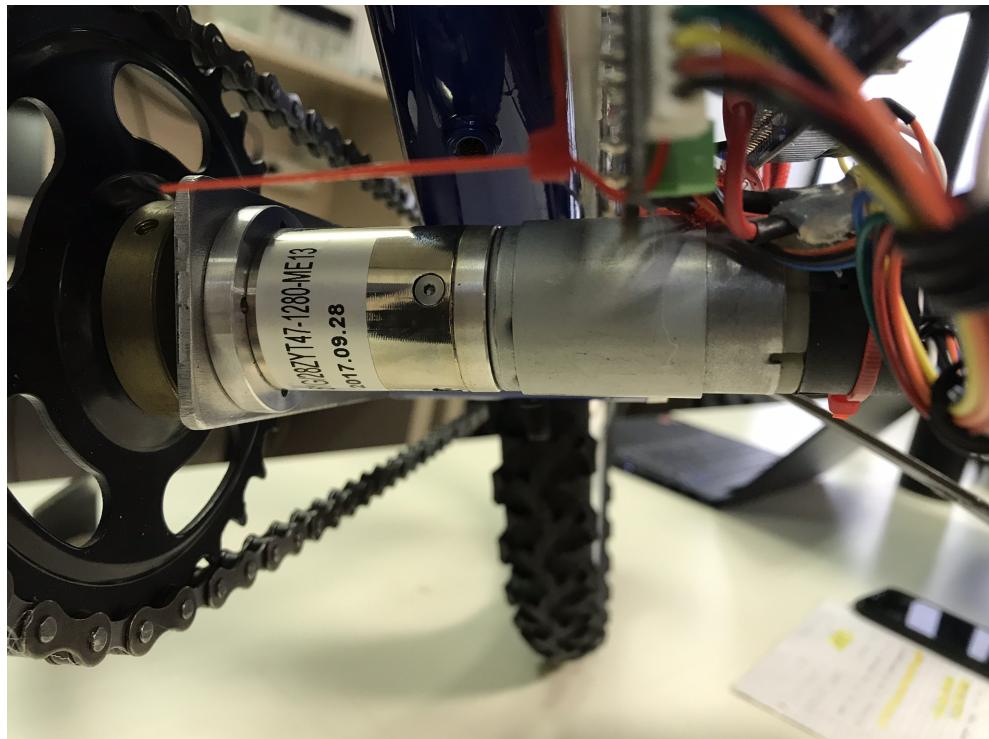
Modifica hardware dettaglio



## 2.2 Attuatori

Per le attuazioni degli sforzi di controllo sono stati utilizzati due diversi motori per la trazione e per la sterzata della bicicletta.

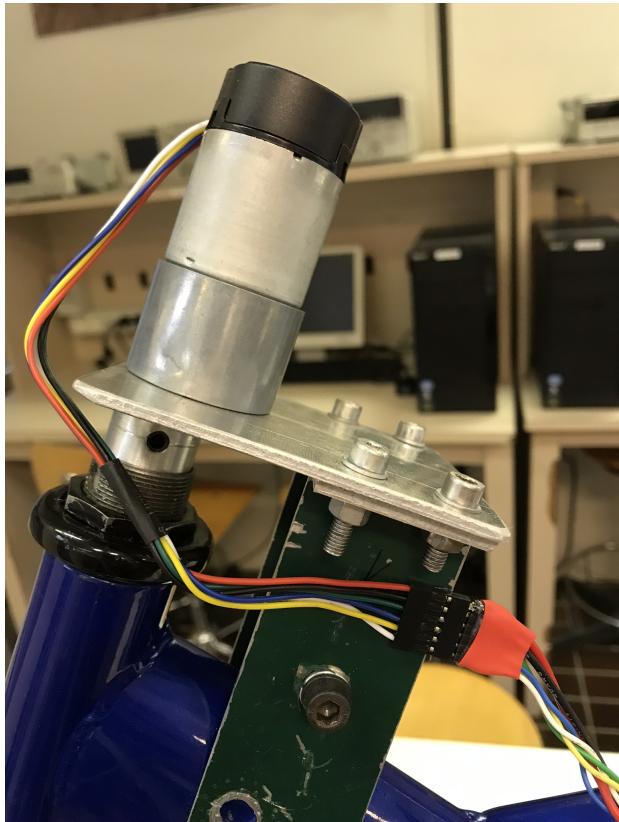
### 2.2.1 Motore trazione



Il motore utilizzato per la trazione è il modello 28PA51G personalizzato da DFRobot. Esso è un motore elettrico di alta qualità, con coppia in uscita elevata e bassa rumorosità :

- Voltaggio nominale : 12V;
- Giri senza carico : 8000*rpm*;
- Giri con riduttore : 146*rpm*;
- Rapporto di riduzione : 51 : 1;
- Corrente a rotore fermo : 3.6A;

### 2.2.2 Motore sterzata



Il motore utilizzato per la sterzata è un pololu 37d metal gear motor 131:1 . È un motore DC brushless con un gearbox metallico **131.25:1** e un encoder integrato capace di una risoluzione di 64 conti per rivoluzione dell'albero motore, che corrisponde a **8400** conti per rivoluzione dall'output del gearbox :

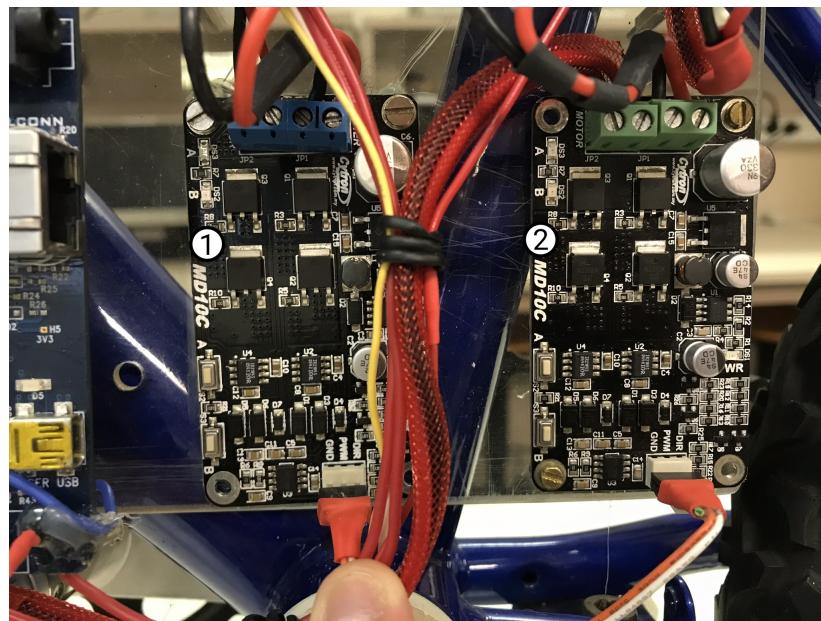
- Voltaggio nominale : 12V;
- Giri senza carico : 76rpm;
- Rapporto di riduzione : 131.25 : 1;
- Corrente a rotore fermo : 5.5A;

### 2.2.3 Driver motori

Driver motore dc MD10C:

Ciascun driver ci permette di regolare la tensione ai capi del motore, mediante un segnale PWM, e la sua direzione di rotazione mediante un segnale digitale. La scheda restituisce in uscita una tensione continua proporzionale alla tensione d'ingresso con costante di proporzionalità data dal duty cycle del PWM generato :  $V_{out} = V_{in}$ .

I driver utilizzati sono quindi 2, uno per la trazione ed uno per la sterzata:



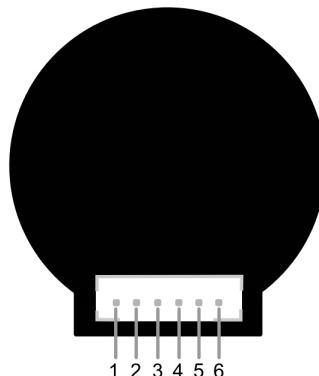
## 2.3 Trasduttori

Per la buona riuscita del progetto si necessita di vari trasduttori/sensori sia per il controllo della corrente in ingresso ai motori che per le altre variaibili.

### 2.3.1 Encoder

Ogni motore utilizzato viene fornito con un encoder incrementale incorporato che fornisce due uscite attraverso le quali è possibile leggere la velocità di rotazione dell' albero motore e la sua direzione.

Nello specifico dai canali A e B dell'encoder escono rispettivamente due onde quadre identiche di valore compreso tra 0V e Vcc, sfasate tra loro di  $90^\circ$  . La frequenza delle due onde indica la velocità del motore mentre l'ordine di queste indica il suo verso di rotazione: se l'onda del canale A precede l'onda del canale B, allora il motore gira in senso orario, altrimenti in senso antiorario.



- 1:Motor -
- 2:Motor +
- 3:Hall Sensor Vcc
- 4:Hall Sensor GND
- 5:Hall Sensor A Vout
- 6:Hall Sensor B Vout

L'encoder montato sul motore di trazione è un encoder a due fasi ad effetto Hall che produce 13 impulsi per ogni rivoluzione dell'albero motore e 663 impulsi per ogni rivoluzione dell'albero all'uscita della gearbox (riduttore di giri).

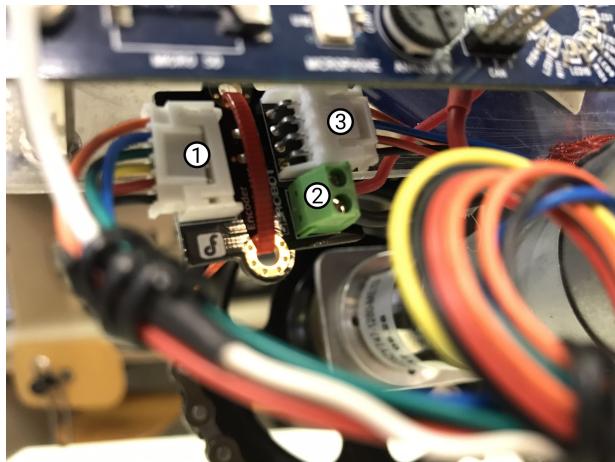
L'encoder montato sul motore adibito alla sterzata è un encoder a quadrature che fornisce 64 impulsi per ogni rivoluzione dell'albero motore e 8400 impulsi per ogni rivoluzione dell'albero all'uscita della gearbox.

### 2.3.2 Adattatore encoder

Per il l'encoder relativo al motore di trazione è stato utilizzato un adattatore che ha la funzione di separare i collegamenti relativi all'alimentazione del motore da quelli relativi all'encoder stesso, e di applicare delle resistenze di pull-up alle uscite dell'encoder.

In figura distinguiamo :

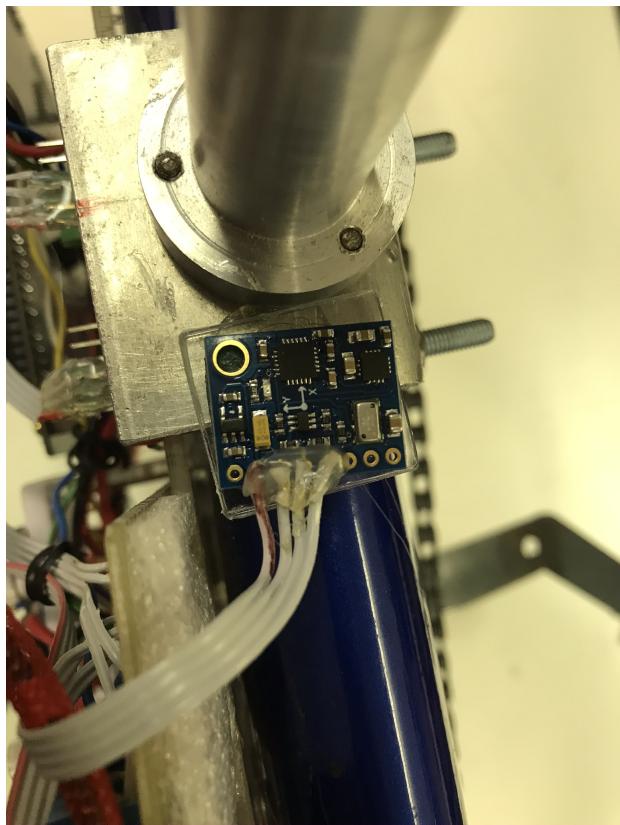
1. Uscita di alimentazione : motore + encoder
2. Ingresso di alimentazione
3. Uscita adattatore : scheda renesas



### 2.3.3 IMU : giroscopio e accelerometro

L'**IMU** (Inertial Measurement Unit) è un sensore di movimento in grado di rilevare variazioni di angolazione e di accelerazione cui viene sottoposto.

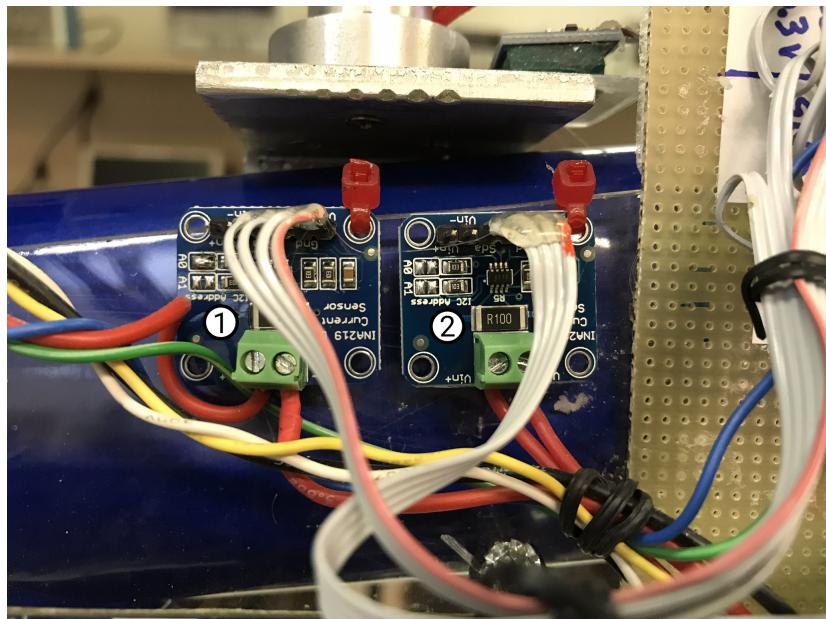
Quello utilizzato da noi si basa sul sensore della Invensense MPU-6050 (accelerometro a 3 assi e giroscopio a 3 assi), sulla bussola digitale a 3 assi HMC5883L della Honeywell e sul sensore di pressione ad alta risoluzione MS5611 della MEAS. Per la comunicazione dei dati al controllore sfrutta il protocollo di comunicazione standard  $I^2C$ .



#### 2.3.4 Sensori di corrente

I sensore di corrente utilizzati (INA219) misurano correnti fino ai  $\pm 3.2A$  con una risoluzione di  $\pm 0.8mA$  con una tensione massima del carico di  $26V$ .

La corrente viene misurata mediante una resistenza di shunt del valore di  $0.1\Omega$ .



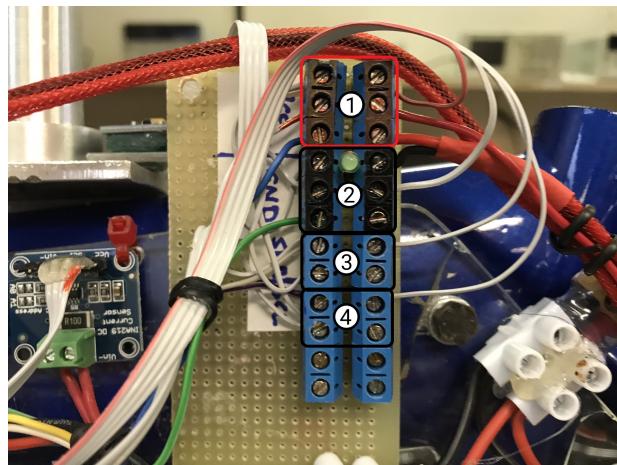
## 2.4 Collegamenti

### 2.4.1 Collegamenti comuni

Per facilitare i collegamenti e per evitare un ammontare di cavi, che percorrono la bicicletta troppo alto, si è optato per l'installazione di alcuni blocchi di tipo mammut (4 nello specifico), ognuno collegato alla rispettiva porta della scheda renesas (SDA SCL) o al blocco di alimentazione (3.3V GND).

Nell'immagine sono riportati i collegamenti :

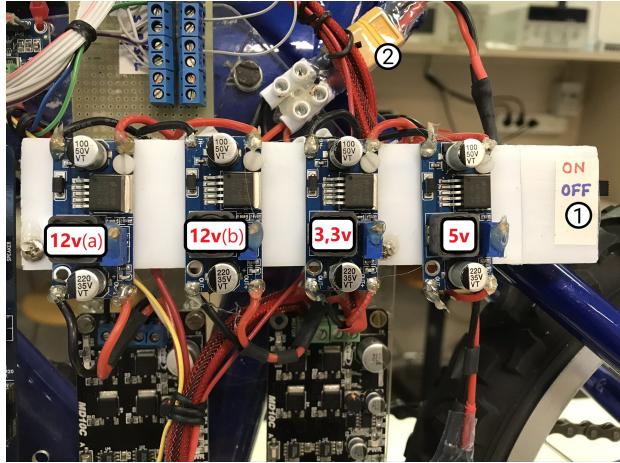
1. **3.3V**
2. **GND**
3. **SDA**
4. **SCL**



### 2.4.2 Alimentazione

L'alimentazione ad ogni componente viene fornita da un'unica batteria lipo da 12V per cui risulta necessario installare degli abbassatori di tensione di tipo buck che forniscono in uscita le diverse tensioni desiderate.

Si sono utilizzati dei componenti basati sul regolatore LM2596S :



### 3 Il nostro contributo al progetto

#### 3.1 Introduzione alle problematiche

Il nostro gruppo di lavoro si è focalizzato sull' implementazione ed il riadattamento dei vecchi codici relativi all' utilizzo dei motori.

Verranno poi discussi le modalità di generazione dei segnali in uscita alla scheda sia dal punto di vista hardware che software, oltre che gli algoritmi utilizzati per il controllo in coppia e corrente dei motori.

#### 3.2 PWM : Pulse Width Modulation

Il funzionamento dei motori elettrici si basa sulla legge fisica secondo cui un conduttore percorso da corrente generi un campo magnetico. La forza elettromotrice indotta negli avvolgimenti rotorici dal campo magnetico generato dallo statore fa sì che si sviluppi una forza di Lorenz che permette al rotore di girare. Ovviamente maggiore è la corrente che scorre negli avvolgimenti statorici e maggiore sarà la forza generata dagli avvolgimenti rotorici, e quindi la velocità di rotazione dell'albero.

Possiamo quindi controllare la velocità di rotazione dell' albero motore applicando tensioni diverse ai capi dei morsetti di alimentazione del motore.

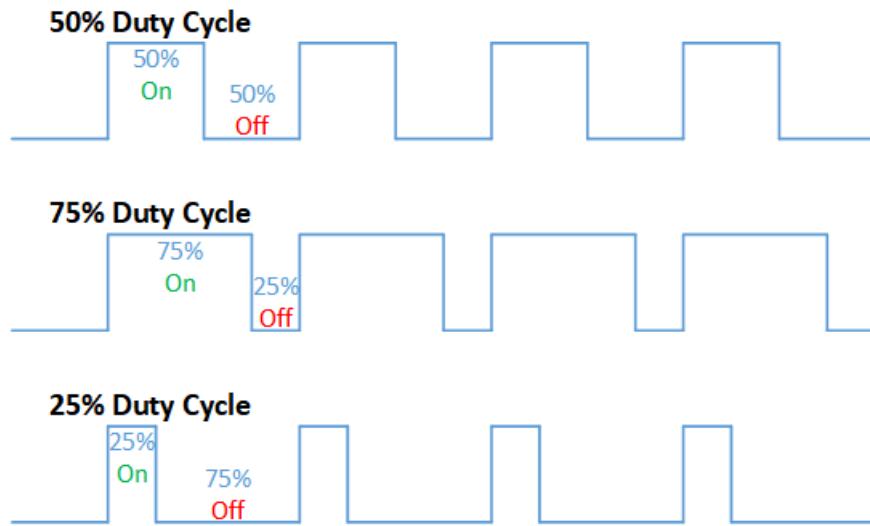
La batteria con cui viene alimentato l' intero progetto fornisce però una tensione costante (idealmente) di 12V (tensione nominale). Per modulare la tensione in ingresso si utilizza un segnale ad onda rettangolare di tipo PWM. Infatti frazionando la tensione  $v_{in}$  in più impulsi con una durata  $\delta$  si dimostra come la tensione che arriva al carico sia proporzionale alla tensione di ingresso :  $V_{out} = \delta V_{in}$ .

la modulazione di larghezza di impulso o PWM, è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e di quello negativo (duty cycle).

### 3.2.1 Duty Cycle

Il duty cycle è la frazione di tempo che un'entità passa in uno stato attivo in proporzione al tempo totale considerato. Infatti sia  $T$  il periodo dell'onda quadra generata e  $t$  il tempo in cui l'onda è ad un livello logico alto, definiamo il duty cycle percentuale:

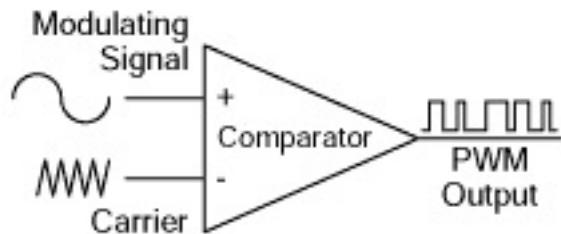
$$\delta = \frac{t}{T}$$



### 3.2.2 Generazione PWM analogico

Il PWM viene generato confrontando un'onda portante (generalmente un dente di sega o un'onda triangolare a frequenze dell'ordine dei khz) con un'onda modulante (che può essere un segnale costante o variabile) mediante l'utilizzo di un comparatore.

Nel nostro caso il segnale generato viene utilizzato per il pilotaggio dei motori dc di cui analizzeremo in seguito le caratteristiche.



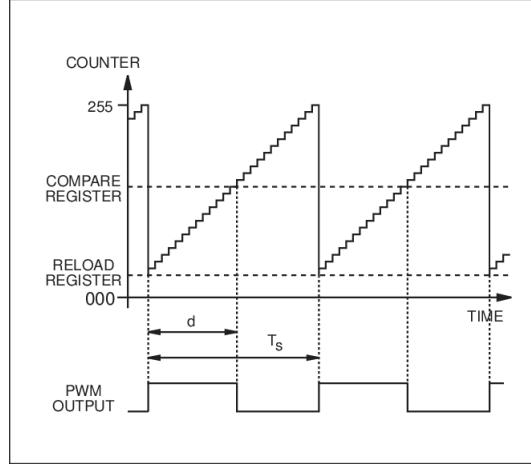
L'uscita del comparatore riproduce lo stato logico del comparatore di ingresso, che confronta continuamente segnale portante e segnale modulante. Nel caso di onda modulante costante l'uscita del comparatore è un'onda quadra la il cui valor medio risulta proporzionale all'ampiezza della modulante :

sia  $V$  l'ampiezza della modulante  $\rightarrow V_{med} = V\delta$ .

Possiamo per cui variare il duty cycle per ottenere la tensione desiderata al carico.

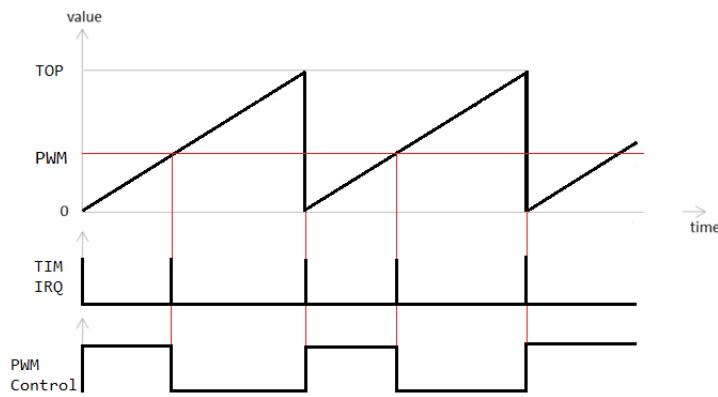
### 3.2.3 Generazione PWM digitale

È possibile realizzare un modulatore PWM anche ricorrendo a circuiti digitali in cui il segnale portante è sostituito da un contatore binario (ad n bit) e l'azione del comparatore viene svolta da un circuito di “compare”. La frequenza del conteggio del contatore è pari alla frequenza della portante moltiplicata per  $2^n$ .



Avremo quindi bisogno di un Timer adibito al conteggio, quindi in modalità compare. In questa modalità il registro di compare assume valori che rappresentano la frazione di periodo in cui il segnale di uscita rimane alto (il duty cycle). L'utilizzo del timer permette la generazione di un interrupt ogni qual volta l'onda modulante è uguale alla portante. In questo frangente può essere modificato il livello logico dell'uscita ottenendo così il PWM richiesto.

In questa modalità il registro di compare assume quindi il valore del duty cycle.



### 3.3 Controllo : PID

Un regolatore **PID** è un controllore standard applicabile a qualsiasi tipo di processo.

In questo tipo di controllori la variabile di uscita  $u(t)$  viene generata in base al contributo di tre termini :

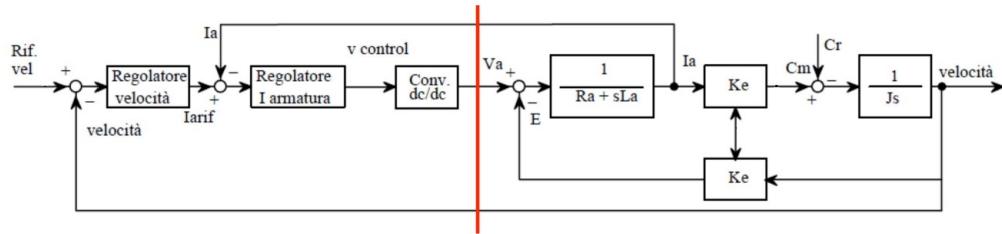
- il primo proporzionale all'errore tra il riferimento e la variabile da controllare :  $K_p e(t)$ ;
- il secondo dipende invece dal valor medio dell'errore ed è proporzionale all'integrale di quest'ultimo :  $K_i \int_0^t e(\tau) d\tau$ ;
- il terzo dipende dalla velocità con cui varia l'errore, quindi proporzionale alla sua derivata :  $K_d \frac{d}{dt} e(t)$ .

Nel complesso scriviamo la legge di controllo del PID come :

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Variando opportunamente i valori di  $K_p$ ,  $K_i$  e  $K_d$  si ottiene la legge di controllo che si necessita.

Il controllo dei motori avviene solitamente tramite 2 controllori posti in cascata atti alla regolazione della coppia motrice generata dal motore ed alla limitazione delle correnti di picco che vengono a circolare all'interno degli avvolgimenti. Di seguito lo schema a blocchi corrispondente :



La parte a destra della linea rossa rappresenta la funzione di trasferimento di un motore elettrico a collettore.

## 3.4 Software

In questa sezione verranno discusse le caratteristiche software relative all'inizializzazione e controllo delle variabili relative agli attuatori ed al loro controllo.

### 3.4.1 Motori : pwm.h/pwm.c

Il file **pwm.h** contiene la struttura dati relativa al modello di un motore elettrico e le definizioni di variabili comuni.

```
typedef struct {
    float volt_signal;           //Voltaggio motore
    float duty;                  //Duty Cycle
    float max_volt;              //Voltaggio nominale motore
    unsigned char ch;             //Canale di riferimento
    int direction_signal;         //Direzione di rotazione
} Motor;
```

La struttura dati *Motor* contiene quindi tutte le variabili utili relativa ad un singolo motore :

- **volt\_signal** : voltaggio applicato al motore;
- **duty** : duty cycle da calcolare in base al voltaggio da applicare;
- **max\_volt** : voltaggio nominale motore;
- **ch** : canale di uscita dei segnali di controllo;
- **diregction\_signal** : direzione di rotazione del motore.

Una volta creata la variabile **Motor** possiamo inizializzarla ed associarle un canale di uscita pwm :

```
void PWM_Init(unsigned char channel_number, float max_volt,
               Motor* m);
```

Nel nostro caso i canali utilizzati sono il 3 ed il 4 (rispettivamente trazione e sterzo) i cui pin di uscita sono :

- **canale 3** : JN2 PIN 23;
- **canale 4** : JN2 PIN 22.

Successivamente si necessita dell'inizializzazione delle porte di uscita dei segnali digitali adibiti alla selezione della direzione di rotazione dei motori :

```
void Init_Port_Dir(void);
```

I PIN di uscita della scheda sono :

- **Motore ch 3** : JN2 PIN 19;
- **Motore ch 4** : JN2 PIN 16.

Una volta inizializzate le tutte le porte di uscita dei segnali di controllo le restanti funzioni si occuperanno del calcolo del *duty cycle* e della modifica della *direzione* di rotazione qualora necessario.

```
void DutyCycle_to_Motor(Motor* m);
```

Calcola il *duty cycle* in percentuale basandosi sul voltaggio da applicare al motore in questione :

$$\delta = \frac{volt\_signal}{max\_volt} 100$$

e setta i valori dei registri corrispondenti alla generazione del *PWM* mediante la funzione :

```
void DutyCycle(float duty_cycle, unsigned char channel);
```

Per quanto riguarda la direzione di rotazione del motore

```
void motor_direction(Motor* m);
```

controlla la positività o meno del voltaggio da applicare al motore e assegna alla variabile *direction\_signal* il valore 0 per il verso orario o 1 per il verso antiorario:

```
if (m->volt_signal < 0) {
    m->volt_signal *= -1;
    m->direction_signal = 1;
} else m->direction_signal = 0;
```

consecutivamente, in base al canale passato alla funzione (*channel*) assegna al registro corrispondente il valore contenuto all'interno della variabile di direzione (*m->direction\_signal*).

Il voltaggio da applicare ai motori verrà calcolato dall'algoritmo di implementazione del **PID** che verrà discusso in seguito.

### 3.4.2 PID : PID.h/PID.c

Il file PID.h presenta al suo interno la definizione della struttura dati relativa al PID, i prototipi delle funzioni che saranno utilizzate e le definizioni di tutti i parametri per l'assegnamento del valore dei guadagni dei PID per le varie tipologie di controllo dei motori.

```
typedef struct {
    float Kp;                      //Guadagno proporzionale
    float Ki;                      //Guadagno integrale
    float Kd;                      //Guadagno derivativo
    float ti;                      //Tempo di integrazione
    float td;                      //Tempo di derivazione
    float tt;                      //Reset time
    float n;                        //Massimo guadagno derivativo
    float b;                        //Set point in k
    float uhigh;                   //Limite superiore uscita
    float ulow;                    //Limite inferiore uscita
    float h;                        //Periodo di campionamento

    float partI;                   //Termine integrale
    float partD;                   //Termine derivativo
    float yold;                     //Uscita precedente del PID
    float uc;                       //Ingresso di RIFERIMENTO
    float y;                        //Ingresso della RETROAZIONE
    float output;                  //Uscita del PID
} PIDSt_Type;
```

Oltre ai vari guadagni, che verranno settati tramite le apposite funzioni, troviamo altri parametri che verranno calcolati sempre all'interno delle funzioni adibite al settaggio.

Una volta creata la variabile di tipo **PIDSt\_Type** essa verrà inizializzata tramite la funzione:

```
int init_pid(PIDSt_Type* PID_P, int Set_param);
```

dove l'argomento *Set\_param* verrà scelto tra le definizioni seguenti in base alle necessità:

```
#define SET_PARAM_PID_TORQUE_2 0
#define SET_PARAM_PID_PITCH 1
#define SET_PARAM_PID_YAW 2
#define SET_PARAM_PID_TORQUE_1 3
#define SET_PARAM_PID_SPEED 4
```

Per quanto riguarda il calcolo dell'uscita del PID, la funzione seguente prende in ingresso una variabile di tipo **PIDSt\_Type**, che conterrà al suo interno il riferimento ed il valore della variabile da controllare, e restituirà in uscita lo sforzo di controllo:

```
int calcPID (PIDSt_Type * PID_P);
```

### 3.4.3 Main program : algoritmo di controllo motore

In questa sezione viene discussa la successione delle funzioni da utilizzare nel main per il controllo di un singolo motore (applicabile ad ogni motore):

1. creazione delle variabili :

```
Motor m1;           //motore TRAZIONE
PIDSt_Type PID_1;   //PID trazione
```

2. inizializzazione :

```
PWM_Init(3, 12, &m1);    //Inizializzazione PWM
Init_Port_dir();         //inizializzazione porte digitali
init_pid(&PID_1, SET_PARAM_PID_TORQUE_1); //controllo Trazione
PID_1->uc = 3;          //Settaggio riferimento (Ampere)
```

*nota: il controllo in coppia avviene tramite la corrente.*

3. loop di controllo :

```
while(1) {
    PID_1->y = SENS->sens_curr_2;    //Lettura corrente
    calcPID(&PID_1);
    m1->volt_signal = PID_1->output;
    DutyCycle_to_Motor(&m1);
}
```

*nota: si necessita dei vari sensori per poter applicare il controllo*

### 3.4.4 Controllo in cascata

Per quanto riguarda il motore utilizzato per la trazione viene utilizzato una tipologia di controllo in cascata.

In questa modalità è possibile controllare la velocità angolare del motore e la corrente che vi scorre all'interno.

necessitiamo quindi di 2 controllori diversi, uno per il controllo della velocità a cui passeremo il riferimento in  $\frac{rad}{s}$  ed uno per il controllo della corrente il cui riferimento sarà l'uscita del pid precedente.

#### 1. creazione delle variabili :

```
Motor m1; //motore TRAZIONE  
PIDSt_Type PID_vel_1; //PID trazione velocita'  
PIDSt_Type PID_curr_1; //PID trazione corrente
```

Va anche dichiarata ed inizializzata la struttura relativa all'encoder per la lettura della velocità angolare dell'albero motore (per maggiori informazioni consultare la relazione del gruppo encoders):

```
encoder_data enc_1;  
parametri_inr_config par_encoder_1;  
identifica_enc enc_id_1;  
  
encoder_init_1();  
Init_Encoder_vars( &par_encoder_1 , &enc_id_1 , 1);
```

#### 2. inizializzazione :

```
PWM_Init(3, 12, &m1); //Inizializzazione PWM  
Init_Port_dir(); //inizializzazione porte digitali  
init_pid(&PID_curr_1, SET_PARAM_PID_TORQUE_1);  
init_pid(&PID_vel_1, SET_PARAM_PID_SPEED);  
PID_vel_1->uc = V_ms / R_bike; //Settaggio riferimento rad/s
```

Dove V\_ms è la velocità tangenziale desiderata in  $\frac{m}{s}$ , mentre R\_bike è il raggio della ruota della bicicletta.

### 3. loop di controllo :

```
Query_Enc(&enc_1, &par_encoder_1, &enc_id_1); // lettura vel  
PID_vel_1.y = enc_1.speed_in_rad_per_sec;  
  
calcPID(&PID_P_vel_1);  
  
PID_curr_1.y=SENS_S.sens_curr_1;  
PID_curr_1.uc = PID_vel_1.output;  
  
calcPID(&PID_P_curr_1);  
  
m1.volt_signal = PID_P_curr_1.output;  
DutyCycle_to_Motor(&m1);
```

Inizialmente si legge la velocità dell' albero motore, che viene utilizzata per il calcolo dell' errore dal pid, per poi calcolare l' uscita da assegnare come ingresso al pid successivo per il controllo della corrente. Il resto è come nel paragrafo precedente.

## 4 Modello matematico

### 4.1 Premesse

Per lo sviluppo del controllore per la stabilizzazione del moto della bicicletta analizzeremo un modello matematico della stessa che si basa sullo studio della bicicletta come pendolo inverso.

Il modello proposto presenta alcune ipotesi semplificative:

- **Velocità** si assume velocità costante  $V$ , in modo da avere modelli tempo-invarianti per il sistema;
- **Sterzata** si assumerà che il controllore possa esercitare controlli sull'angolo di rollio unicamente agendo sull'angolo di sterzata.

È importante evidenziare fin d'ora che le analisi svolte nel corso della trattazione saranno sempre basate su modelli del sistema linearizzati attorno alla posizione di equilibrio verticale.

Riportiamo inoltre la simbologia utilizzata per i parametri del sistema in esame:

Parametro	Simbolo	Valore
Velocità	$V$	$3 \frac{m}{s}$
Momento d'inerzia del sistema	$I$	$10Kgm^2$
Distanza tra il centro di gravità e la ruota posteriore	$A$	$0.4m$
Distanza tra il centro di gravità e la ruota anteriore	$B$	$0.7m$
Distanza tra le ruote	$L = A + B$	
Altezza del baricentro	$H$	$0.5m$
Massa bicicletta	$M$	$10Kg$

### 4.2 Cinematica

Esaminiamo il movimento orizzontale del sistema bicicletta-ciclista nel piano stradale. Fissata una retta di riferimento, definiamo le seguenti coordinate per descrivere lo spostamento laterale:

- $x(t)$ : posizione laterale del punto di contatto della ruota anteriore rispetto alla retta di riferimento;
- $y(t)$ : posizione laterale del punto di contatto della ruota posteriore rispetto alla retta di riferimento;
- $z(t)$ : posizione laterale della proiezione sul suolo del centro di massa del sistema rispetto alla retta di riferimento;
- $\Psi(t)$ : angolo di imbardata della bicicletta.
- $\Phi(t)$  : l'angolo di sterzo della ruota anteriore.

Per semplicità ometteremo le dipendenze temporali delle variabili sopra indicate ed indicheremo con  $\dot{x}$  la derivata temporale della variabile.  
Le equazioni cinematiche fondamentali che descrivono il movimento laterale sono:

$$\begin{cases} \dot{x} &= V \frac{\sin(\Phi + \Psi)}{\cos(\Phi)} \\ \dot{y} &= V \sin(\Psi) \\ z &= \frac{Ax + By}{L} \\ L \sin(\Psi) &= x - y \end{cases}$$

Linearizzando le equazioni attorno al punto di equilibrio  $(\bar{x}, \bar{y}, \bar{\Phi}) = (0, 0, 0)$  si ottiene il sistema di equazioni linearizzate :

$$\begin{cases} \dot{x} = V(\Phi + \Psi) &= \frac{V}{L}x + \frac{V}{L}y + V\Phi \\ \dot{y} = V\Psi &= \frac{V}{L}x - \frac{V}{L}y \\ z = \frac{A}{L}x + \frac{B}{L}y \end{cases}$$

Da cui, assumendo  $\Phi$  come ingresso,  $x, y$  come lo stato del sistema e  $z$  come uscita, si ottiene la funzione di trasferimento:

$$\frac{Z(s)}{\Phi(s)} = W_1(s) = \frac{V}{L} \frac{As + v}{s^2}$$

### 4.3 Dinamica laterale

Per piccoli angoli di imbardata  $\Psi$ , il movimento di inclinazione di una bicicletta può essere modellato come quello di un pendolo invertito in cui il movimento laterale della base del pendolo è quello determinato dalle equazioni cinematiche descritte precedentemente.

Sia  $\beta$  l'angolo della bicicletta rispetto all'asse verticale, si può scrivere l'equazione di equilibrio dei momenti rispetto alla base del pendolo:

$$(MH^2 + I)\ddot{\beta} = MgH \sin(\beta) - MH\ddot{z} \cos(\beta)$$

che linearizzate per piccole variazioni dell'angolo  $\beta$  e ponendo  $MH^2 + I = J$  si ottiene :

$$J\ddot{\beta} = MgH\beta - MH\ddot{z}$$

da cui si ricava la funzione di trasferimento che lega lo spostamento laterale  $z(t)$  all'inclinazione laterale della bicicletta :

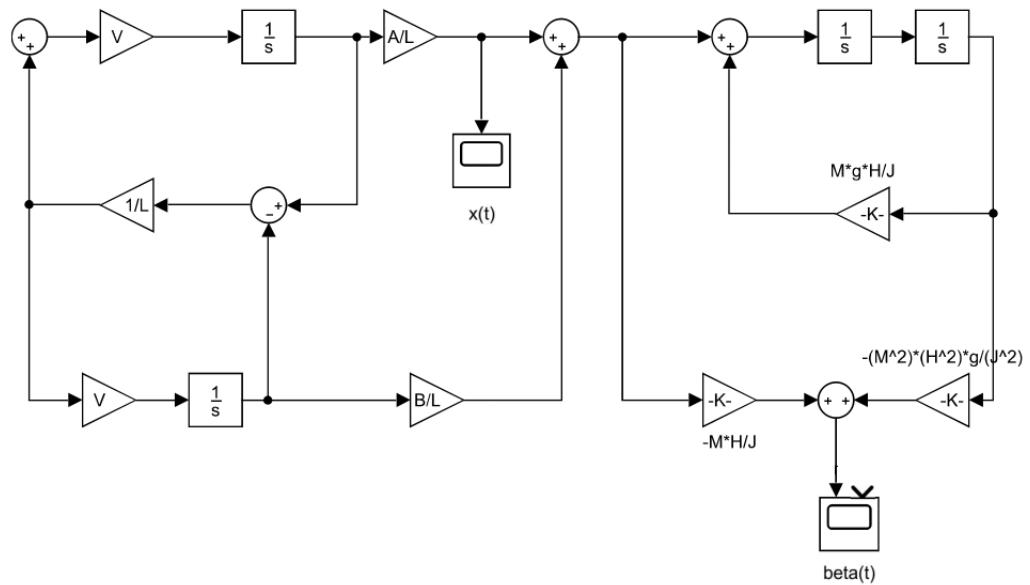
$$\frac{B(s)}{Z(s)} = W_2(s) = \frac{-MHS^2}{Js^2 - MgH}$$

#### 4.4 Funzione di trasferimento del sistema

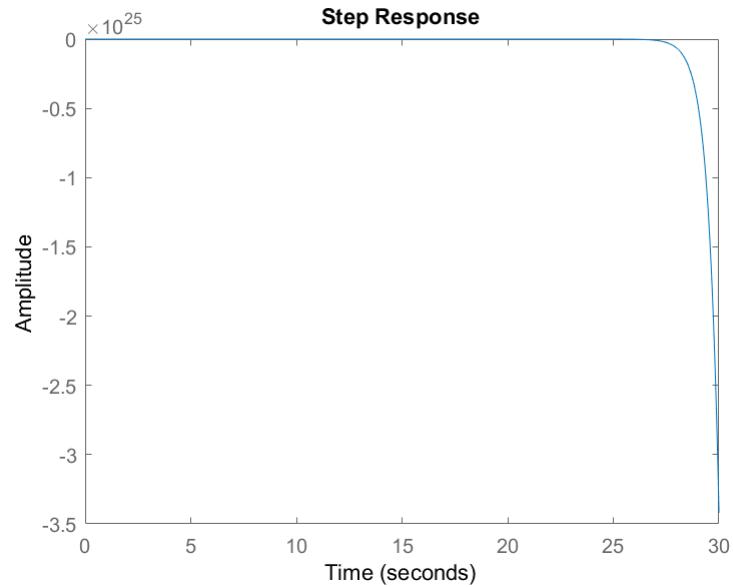
Mettendo in serie le due funzioni di trasferimento  $W_1(s)$  e  $W_2(s)$ , nel dominio delle trasformate, l'angolo di sterzo  $\Phi$  e la inclinazione laterale  $\beta$  :

$$W(s) = W_1(s)W_2(s) = \frac{B(s)}{\Phi(s)} = \frac{-\frac{V}{L}(As + V)MH}{Js^2 - MgH} = -\frac{VAMH}{LJ} \frac{s + \frac{V}{A}}{s^2 - \frac{MgH}{J}}$$

da cui ne deriva lo schema simulink :



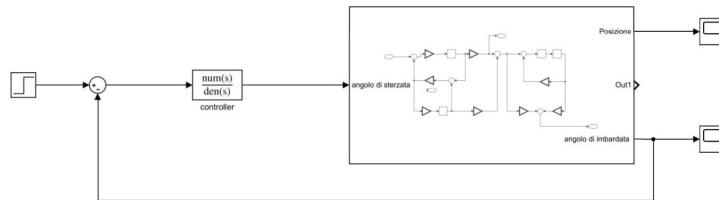
Il sistema risulta quindi instabile a causa del polo a parte reale positiva:  $p_1 = \sqrt{\frac{MgH}{J}}$ , e necessita pertanto di un controllore che possa stabilizzare il sistema ad anello chiuso. Per completezza riportiamo la risposta al gradino del sistema a ciclo chiuso:



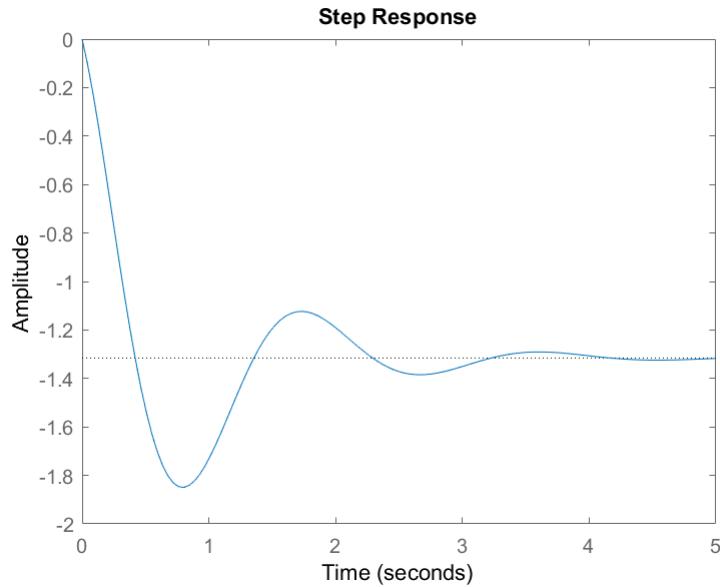
## 4.5 Controllore proporzionale

Come prima opzione si può pensare di utilizzare un controllore con guadagno costante in modo da stabilizzare il sistema; per determinare il valore del guadagno  $K$  necessario, si ricorre alla funzione di trasferimento ad anello chiuso  $W(s)$ :

$$W(s) = -K \frac{VAMH}{JL} \frac{s + \frac{V}{A}}{s^2 - K \frac{VAMH}{JL} s - k \frac{V^2 MH}{JL} - \frac{MgH}{J}}$$



da cui si dimostra che per  $K > \frac{gL}{V^2}$  il sistema risulta stabile:  
Simulazione:  $K = 5 > \frac{gL}{V^2} = 1.199$



Per mezzo di tale compensatore si ottengono un *sovravelongazione* ed un *tempo di assestamento* accettabili ma, a causa della mancanza di un integratore nel controllore, si ha un errore a regime permanente costante.

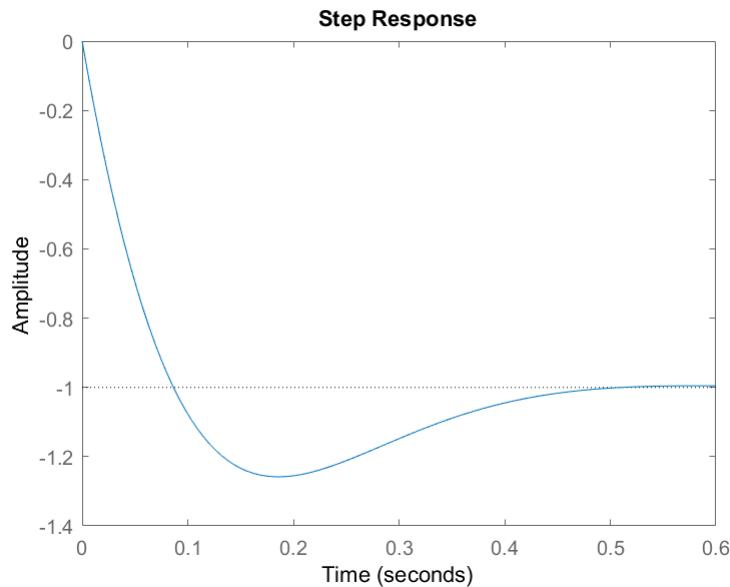
## 4.6 Controllore PI

Utilizzando un controllore di tipo **PI** si riesce quindi ad annullare l' errore a regime permanente e ad ottenere prestazioni migliori.

A tal proposito si è utilizzato un controllore del tipo :

$$C(s) = 60 * \frac{1 + 0.67s}{s}$$

ottenendo la seguente risposta al gradino :



## 4.7 La retroazione intrinseca del sistema

Il modello descritto precedentemente si basa su alcune semplificazioni, pertanto non è in grado di giustificare alcuni aspetti della dinamica della bicicletta, in particolare la sua capacità di autostabilizzarsi per valori di velocità efficienti.

La grandezza che gioca un ruolo fondamentale nella descrizione di questo fenomeno è l'avancorsa  $t$ , ovvero la distanza tra il punto di intersezione dell'asse di sterzo con il piano stradale e il punto di contatto della ruota sterzante.

Se indichiamo con  $d$  la distanza tra il centro della ruota e l'asse di sterzo,  $\lambda$  l'angolo di inclinazione dell'asse er  $R$  il raggio della ruota, otteniamo :

$$t = R \tan(\lambda) - \frac{d}{\cos(\lambda)}$$

Attraverso altre considerazioni di carattere fisico si giunge alla formulazione del modello matematico del sistema bicicletta-ciclista.

Nel complesso, il sistema viene suddiviso in due sottosistemi distinti: il telaio e la forcella anteriore. Per quanto riguarda il primo, esso viene descritto in modo analogo al modello descritto nel paragrafo precedente, mentre per la forcella viene usato un modello statico. Indicata infatti con  $T$  la coppia applicata dal ciclista sull'asse dello sterzo, la descrizione del funzionamento della forcella anteriore può essere fatta mediante un bilancio delle coppie che la caratterizzano:

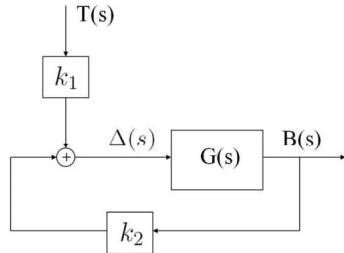
$$\delta = k_1(V)T + k_2(V)\beta$$

dove  $k_1$  e  $k_2$  sono parametri che dipendono dalla velocità e dalla geometria dello sterzo :

$$k_1(V) = \frac{L^2}{t M A \cos(\lambda)(V^2 \cos(\lambda) - g L \sin(\lambda))}$$

$$k_2(V) = \frac{Lg}{V^2 \cos(\lambda) - Lg \sin(\lambda)}$$

Il modello che ne deriva può dunque essere schematizzato mediante il seguente diagramma a blocchi:



la cui funzione di trasferimento lega l'angolo di imbardata  $\beta$  con la coppia  $T$  applicata al manubrio:

$$\frac{B(s)}{T(s)} = -\frac{k_1 VAMH}{LJ} \frac{s + \frac{V}{A}}{s^2 + k_2 \frac{VAMH}{LJ} s + \frac{MgH}{J} \left( \frac{k_2 V^2}{Lg} - 1 \right)}$$

da cui se ne deriva come per valori di  $V$  per cui vale che  $\frac{k_2 V^2}{Lg} > 1$  il sistema risulta stabile indipendentemente dall'azione della coppia applicata dal ciclista.