UNIVERSITÁ POLITECNICA DELLE MARCHE
FACOLTÁ DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica e dellAutomazione

(A.A. 2015/ 2016)

Laboratorio di automazione (Prof. A. Bonci)

SELF BALANCING BICYCLE

Studenti:
Matteo Camerlengo
Francesco di Girolamo
Mirko Simoni
Fulvio Tesei
Professore:
Andrea Bonci

SOMMARIO

1. INTRODUZIONE

Il presente elaborato ha lo scopo di studiare e sperimentare la dinamica e il controllo della stabilitá di una bicicletta automatizzata, la cui guida avvenga senza l'ausilio del ciclista.
Abbiamo inizialmente ripreso il lavoro fatto da degli studenti degli anni passati, in cui era istallato un microcontrollore Renesas SH7201. Lo abbiamo sostituito con un nuovo microcontrollore Renesas YRDKRX63N, abbiamo tradotto tutti i collegamenti della vecchia e della nuova scheda con tutte le periferiche hardware, ed abbiamo riscritto maggior parte del codice affinché il lavoro svolto dalla bicicletta sia il piú simile possibile a quello della bicicletta del vecchio progetto.
Grazie alle periferiche istallate, la bicicletta sará in grado, una volta alimentati i due motori, quello che fa girare la ruota posteriore, ed il servomotore, che fa ruotare lo sterzo, di calcolare la propria inclinazione rispetto all'asse verticale, e ruotare lo sterzo nel verso giusto affinché rimanga in equilibrio.
Dagli studi eseguiti dal primo gruppo che ha lavorato sulla bicicletta, é stato calcolato che la velocitá minima di stabilizzazione é di 1.8 m/s. Il motore attualmente istallato a bordo permette una velocitá massima di 1 m/s, perció non siamo riusciti a far rimanere la bicicletta perfettamente in equilibrio, ma abbiamo ottenuto buone risposte da parte del servomotore a variazioni dell'angolo di inclinazione.

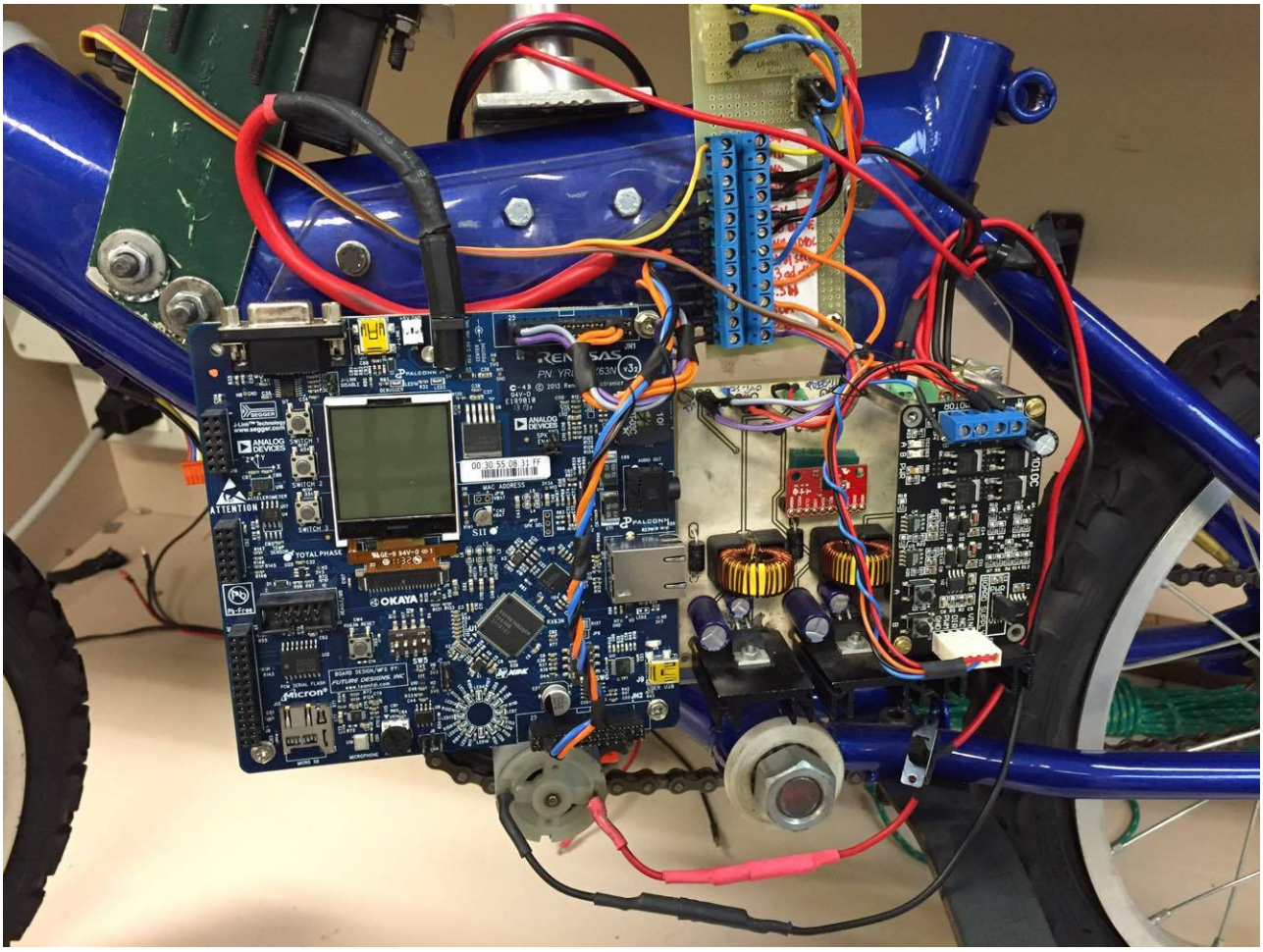


Figura 1. Hardware Bicycle

2. HARDWARE

2.1. Renesas YRDKRX63N

Il demontration kit YRDKRX63N è la scheda su cui è montato il micro-controller RX63N e una serie di periferiche di servizio.

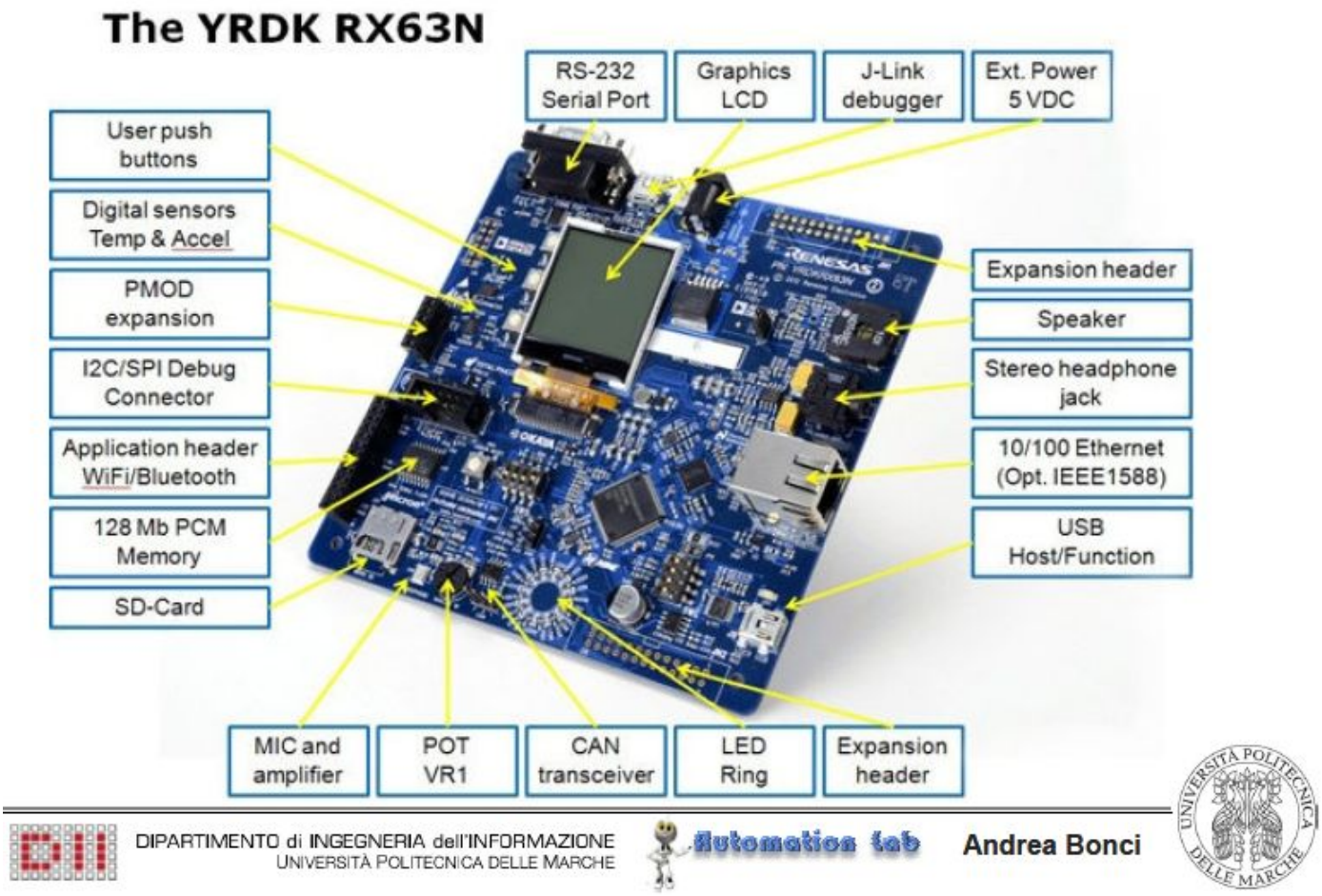


Figura 2. Scheda YRDKRX63N

L'YRDKRX63N incorpora un RX63N core della famiglia RX631 che include:

- 32-bit MCU capace di operare oltre i 100 MHz;
- FPU (Floating-PointUnit) per i calcoli aritmetici;
- Oltre 21 canali per ADC a 12-bit e oltre 2 canali per DAC;
- unit Timers MTU2 [Multi Timer Unit Function], con funzioni di:
  - input capture;
  - output compare;
  - counter clearing per generazione di segnali PWM;

  - controllo motori;

- watchdog timer Independente e funzione CRC per lo standard di applicazioni domestiche (IEC 60730) for Europe;
- molte funzioni di comunicazione: Ethernet, SCI, RSPI, CAN, and I2C.

Tabella 1. Principali caratteristiche della scheda YRDKRX63N

μ Chip: RX63N	PPG: Programmable Pulse Generator
Family: RX631 Renesas	I/O PORT: General I/O Port (GPIO)
family 32bit: RX [600] CISC architecture	PFC: Pin Function Controller
CPU: 32 bit, 100MHz	ADC: Analog/Digital Converter (12bit, 10bit)
FPU: Floating Point Unit	DAC: Digital/Analog Converter (10bit)
DMAC: Direct Memory Access controller	Ethernet: 10/100 MAC
CAN BUS: Controller Area Network	USB 2.0: Device/Host OTG
SCI: Serial Communication Interface	SPI: Serial Peripheral Interface
Syncronous and Asyncronous serial communication as	I2C BUS: InterIntegrated Circuit (IIC / I2C)
UART Universal asyncronous receiver/transmitter, etc ...	3-axis accelerometer
TIMERS:	Temperature Sensor
MTU, MTU2: multi-function Timer pulse unit	Micro SD card
(PWM signal generator, pulse signal generator, etc...)	Speaker
WDT: Watchdog Timers (also independent)	Microphone
(System Reset if the user program goes out of control, etc...)	
INTC: Interrupt Controller	
(Device for managing priority vectorized interrupts with priority for the MCU unit)	
RTC: Real Time Clock	
CMT: Compare Match Timer	

Nel nostro programma siamo andati ad utilizzare un MTU timer per la temporizzazione di un impulso PWM del motore, e per la temporizzazione dell'impulso del servomotore.
Abbiamo inoltre usufruito dell'I2C BUS per implementare il protocollo di comunicazione master-slave, e l'USB 2.0 per permettere il collegamento e la comunicazione tra PC e la scheda.
Attraverso il display LCD abbiamo potuto visualizzare dei dati prodotti dalla scheda, per poi riutilizzarli nella progettazione e il calcolo dell'angolo di sterzata che doveva produrre il servomotore.

2.2. Scheda di alimentazione

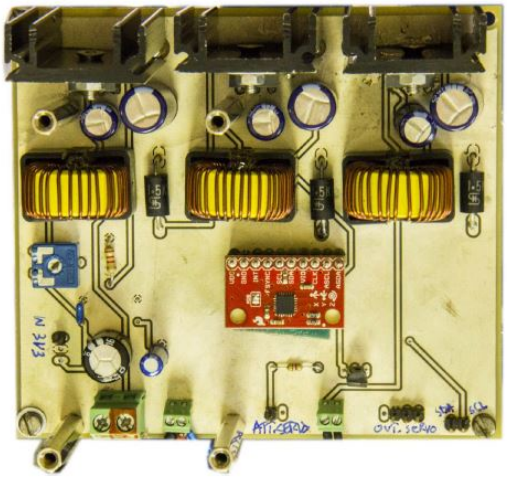


Figura 3. Scheda di alimentazione

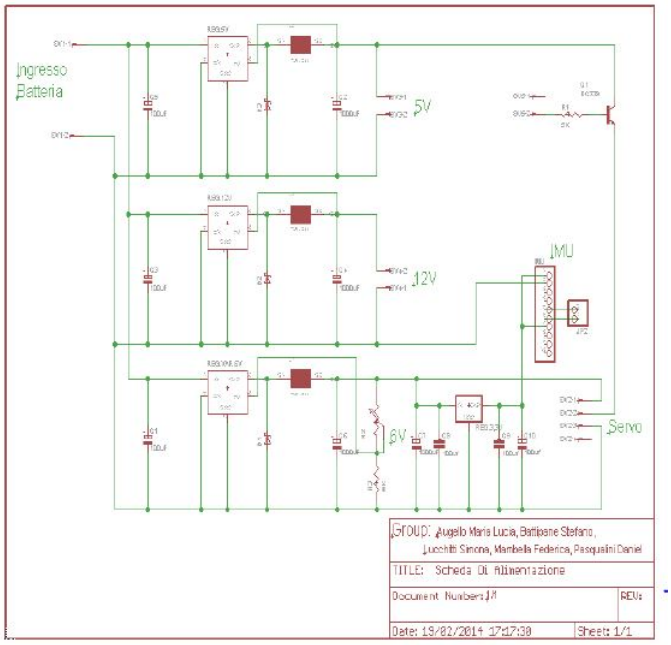


Figura 4. Schema della scheda di alimentazione

Per alimentare l'elettronica del dispositivo si è scelto di progettare un alimentatore a 4 uscite, utilizzando un regolatore a logica switching di tipo step-down, nel caso particolare un LM257.

2.3. Principio Di Funzionamento Del Regolatore Step-Down

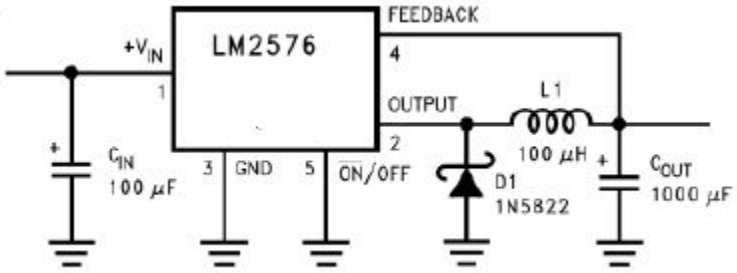


Figura 5. Circuito Regolatore Step-Down LM2756

Un regolatore di tensione di tipo Step-Down è un componente in grado di fornire in output una tensione predefinita e ridotta rispetto a quella fornita in ingresso (nel caso specifico si è usato lintegrato LM2576). La configurazione classica di tale circuito è quella mostrata in Figura 4.
Per mantenere la stabilitá sull'ingresso (pin 1 del LM2576), il regolatore deve essere bypassato con un condensatore elettrolitico di almeno 100F, al quale sarebbe anche utile, non in questo caso peró, affiancargli in parallelo un classico condensatore in ceramica per aumentare la stabilitá a basse temperature. L'induttanza e il condensatore in uscita (pin 2 del LM2576) sono usati come filtro sulla tensione di output e sono necessari per la stabilitá del ciclo, anche se in caso di cali di tensione su o sotto lo zero, l'induttanza creerá una controtensione e quindi una corrente inversa che viene scaricata grazie ad un percorso di ritorno fornito dal diodo Schottky. Il Feedback (pin 4 del LM2576) ha il compito di far diminuire il rumore in uscita e per questo viene collegato al potenziale positivo dell'otuput. Esiste anche un pin ON(negato)/OFF (pin 3 del LM2576) che puó essere messo o a massa o pilotato da un livello basso di tensione del tipo TTL(tipicamente sotto gli 1,6V), e in questo caso si potrebbe mettere in standby il componente applicando un livello alto di tensione sempre del tipo TTL o CMOS. In conclusione, per mantenere un buona stabilitá il potenziale di massa deve avere una bassa impedenza.

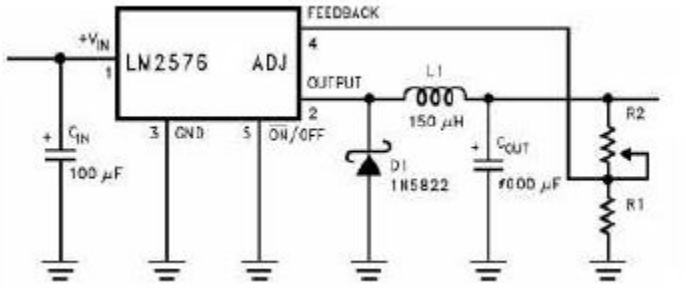


Figura 6. Circuito Regolatore Step-Down LM2756 ADJ

In Figura 5. é riportato sempre il circuito di un regolatore di tensione di tipo Step-Down e sempre realizzato con un LM2576 ma questa volta di tipo ADJ cioé adjustable (regolabile). Come notiamo l'unica effettiva differenza é che il Feedback é collegato tra le due resistenze in serie (R1 e R2) collegate in parallelo al condensatore di output, le quali in base al loro dimensionamento daranno un valore in output di tensione

pari a quello desiderato. L'utilizzo di questo circuito é stato necessario poiché il servomotore necessita di tensioni comprese tra 4.8V e 6V e in commercio non esistono LM2576 con output di 6V.
La formula per calcolare il valore di tale resistenze in base al valore di uscita desiderato viene fornita direttamente dal datasheet ed é:

$$Vout = Vref(1 + \frac{R2}{R1}) \tag{1}$$

$$R2 = R1(\frac{Vout}{Vref} - 1) \tag{2}$$

Dove: Vref=1,23V e R1 compresa tra 1KΩ e 5KΩ.

2.4. LM2576

Esistono vari componenti della serie LM2576 che forniscono tensioni particolari in uscita, uno di essi è regolabile attraverso il dimensionamento di alcuni componenti montati esternamente all'integrato.
In tal caso è necessario regolare la tensione della batteria (14,8V) a:

- 12V per l'alimentazione del motore posteriore;
- 6V per l'alimentazione del servo motore;
- 5V per l'alimentazione della scheda Renesas;
- 3V3 per l'alimentazione della IMU.

Si è scelto quindi di utilizzare due LM2576, uno con output a 12V e l'altro a 5V . Per i 6V è stato utilizzato un LM2576 ADJ, dimensionando le resistenze al fine di ottenere la tensione di alimentazione desiderata; nel caso dei 3V3 l'alimentazione è stata prelevata direttamente dalla scheda Renesas.

2.5. Scheda

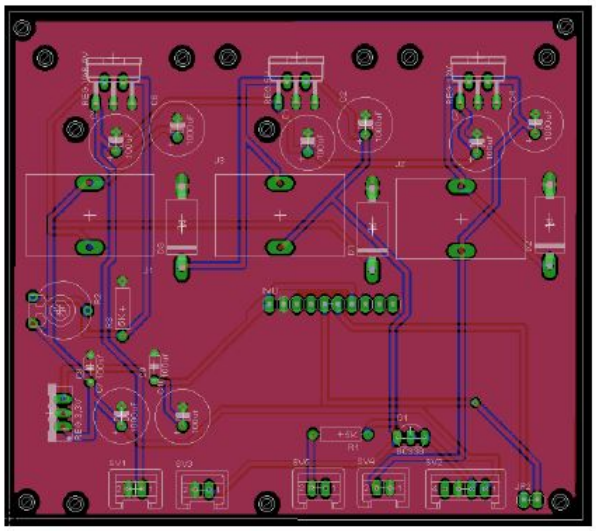


Figura 7. Scheda PCB

La scheda PCB è stata progettata per raccogliere i regolatori di tensione, i collegamenti tra IMU e microcontrollore e tra servo e microcontrollore in un'unica soluzione stampata, al fine di ridurre lo spazio necessario, unificare la massa e rendere il circuito facilmente fruibile. La vicinanza dei componenti non ha solo lo scopo di ridurre l'ingombro; è infatti indispensabile collegare un condensatore elettrolitico di bypass tra il pin d'ingresso dell'integrato e il pin di massa GND il piú vicino possibile ai regolatori per prevenire la comparsa di transitori di tensione sulla sorgente di alimentazione e per garantire un funzionamento stabile del componente. Le dimensioni notevoli delle piste e dello spazio di isolamento tra esse e il piano di massa servono a gestire meglio il flusso di possibili correnti elevate. Tutti i regolatori LM2576 sono stati montati con il dorso rivolto verso il bordo della scheda per favorire il posizionamento degli indispensabili dissipatori che limitano l'aumento di temperatura dovuto al discreto drop-out di tensione. Sono stati inoltre posizionati i fori per il montaggio del driver del motore posteriore e il socket di alloggiamento dell'IMU.
È predisposto inoltre un ingresso per l'alimentazione 3v3 proveniente dalla scheda Renesas, indispensabile per alimentare l'IMU, il quale è connesso sempre alla scheda Renesas tramite i due pin predisposti per il protocollo I2C.

2.6. IMU(MPU6050)

Figura 8. IMU

Un'unitá di misura inerziale (nota anche come Inertial Measurement Unit, o IMU), è un sistema elettronico basato su sensori inerziali, come accelerometri e giroscopi. Serve dunque per la misurazione diretta di accelerazioni e velocitá angolari rispetto a tre assi e per il calcolo di velocitá e posizione attraverso l'integrazione delle quantitá misurate da parte di un processore.
Nello specifico, l'MPU6050 composto da un giroscopio e un accelerometro entrambi a tre assi montati sullo stesso silicio insieme al processore digitale di movimento (DMP Digital Motion Processor) in grado di elaborare algoritmi su 9 assi complessi. Viene alimentato con una tensione (VDD) compresa tra i 2,375V ai 3,46V e lavora con una tensione logica (VLOGIC) compresa tra 1,71 fino alla VDD. È compreso di un bus I2C che permette alla MPU-6050 di accedere a magnetometri esterni, a altri sensori o, come nel nostro caso a comunicare con il microprocessore.

2.6.1. Giroscopio. Il giroscopio misura la velocitá angolare sui tre assi. É possibile accedere a questi valori tramite gli appositi registri dell'MPU6050. Il fondoscala del sensore é regolabile e puó essere settato su valori di $\pm$ 250, $\pm$ 500, $\pm$ 1000 e $\pm$ 2000 ^{o} /sec

2.6.2. Accelerometro. L'accelerometro misura le accelerazioni presenti sui tre assi, è possibile utilizzare queste per ricavare l'inclinazione della scheda sfruttando l'accelerazione gravitazionale terrestre, è inoltre possibile percepire urti, frenate, o in generale variazioni di accelerazione fino a 16g. Il fondoscala del sensore è regolabile e puó essere settato su valori di $\pm$ 2g, $\pm$ 4g, $\pm$ 8g e $\pm$16g

2.7. Driver Motore MD10C

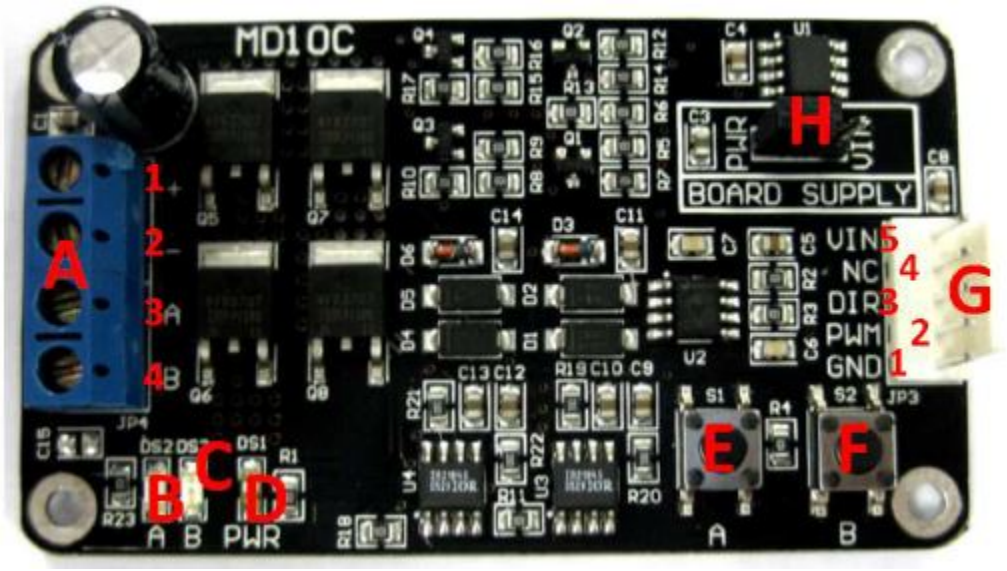


Figura 9. Scheda Driver Motore MD10C

La scheda Driver Motore MD10C è progettata per guidare correnti e tensioni elevate su motori DC usando come input per la regolazione della velocitá un segnale PWM proveniente dal microcontrollore:

- Controllo bidirezionale per il motore DC;
- Tensione di alimentazione va da 3V a 25V;
- Sopporta fino a 10A continui e 15A di picco massimo di corrente (10 secondi);
- 3,3 V e 5V ingresso livello logico;
- I componenti a stato solido forniscono il tempo di risposta pi veloci;
- Frequenza di controllo PWM fino a 10KHz;
- Gestisce modulo e segno PWM;
- Dimensioni: 75mm x 43 millimetri;

A. Blocco terminale: connessione tra motore e alimentazione;

1. + : Alimentazione positiva;
2. - : Alimentazione negativa;
3. A: Connessione al terminale A del motore;

4. B: Connessione al terminale B del motore;

B. LED A(rosso): si accende quando l'output di A è alto e quello di B è basso. La corrente circola da A a B;

C. LED B(rosso): si accende quando l'output di A è basso e quello di B alto. La corrente circola da B ad A;

D. POWER LED (verde): si accende quando la scheda è alimentata;

E. TEST BUTTON A: quando viene premuto la corrente circola da A a B e il motore girerá in senso orario (o antiorario in base alla connessione);

F. TEST BUTTON B: quando viene premuto la corrente circola da B ad A e il motore girerá in senso antiorario(o orario in base alla connessione);

G. INPUT

1. GND: massa logica;
2. PWM: ingresso per controllo velocit;
3. DIR: controllo direzionale;
4. NC: non connesso. Questo pin non usato;
5. VIN: pin di alimentazione scheda;

H. JUMPER: Selettore di alimentazione scheda

- PWR: La scheda è alimentata dallalimentazione del motore. Si pu utilizzare solo quando il motore alimentato con un ingresso maggiore di 14V;
- VIN: La scheda alimentata da VIN. Bisogna avere 12V di ingresso su VIN per far si che l'alimentazione dingresso del motore possa essere compresa tra i 3V ai 25V.

2.8. Adattatore di livello di tensione

Per quanto riguarda il collegamento tra la scheda Renesas e il servomotore, si è riscontrato una discordanza tra i livelli di tensione, in quanto il primo lavora con un bus ad una tensione di 3.3V mentre il secondo ad un livello di 5V. Questo problema è stato risolto realizzando un circuito di adattamento composto principalmente da due transistor discreti BS170 e da resistenze, collegati come mostrato in figura:

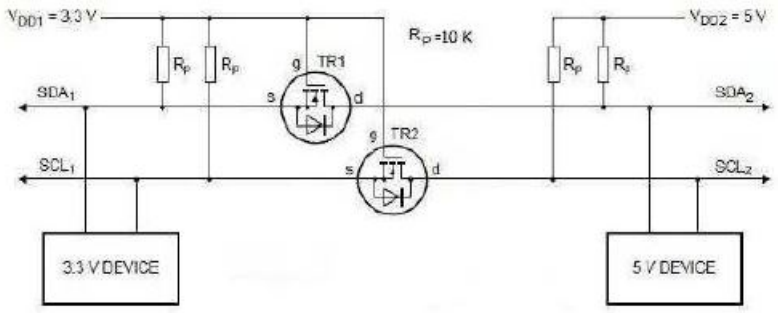


Figura 10. Schema adattatore di livello di tensione

Nella foto riportata qui di seguito, viene mostrato il circuito realizzato in laboratorio. La parte di sinistra va quindi collegata ai due pin della scheda Renesas, quella di destra ai 5 volt della scheda di alimentazione.

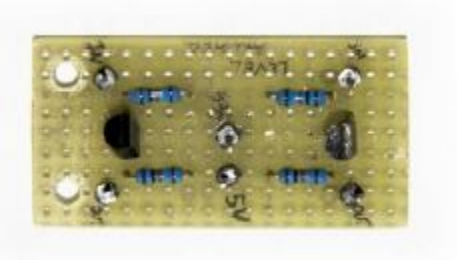


Figura 11. Adattatore di livello di tensione

2.8.1. Transistor BS170(Valori massimi).

- Vdss (Tensione tra il pin di Drain e il pin di Source): 60V;
- Vdgr (Tensione tra il pin di Drain e il pin di Gate): 60V;
- Vgss (Tensione tra il pin di Gate e il pin di Source): $\pm$ 20;
- Id (corrente sul Drain): 500 mA (DC), 1200 mA (AC);
- Tj (Temperatura in funzione);
- Pd (Potenza dissipata): 830 mW (a 25C).

2.9. Alimentazione di batteria

Per l'alimentazione è stata utilizzata una batteria Li-Po a 3250 mAh. Le batterie Li-Po sono piú stabili e piú affidabili rispetto alle tradizionali batterie stilo, come ad esempio le NiMH, e riescono ad erogare la stessa quantit di corrente per l'intero periodo di carica.

Tipo	LI-PO
Voltaggio	14.8 (4S) Volt
Capacitá	mAh 3250
Scarica Continua	25C
Scarica Picco	50C
Carica	2C
Tipo Balancer	RCS/GreatPlanes
Dimensioni	24x45x136 mm
Peso	316 Grammi

Tabella 2. Caratteristiche tecniche della batteria

Figura 12. LI-PO 3250 mAh

2.10. Schema di collegamento

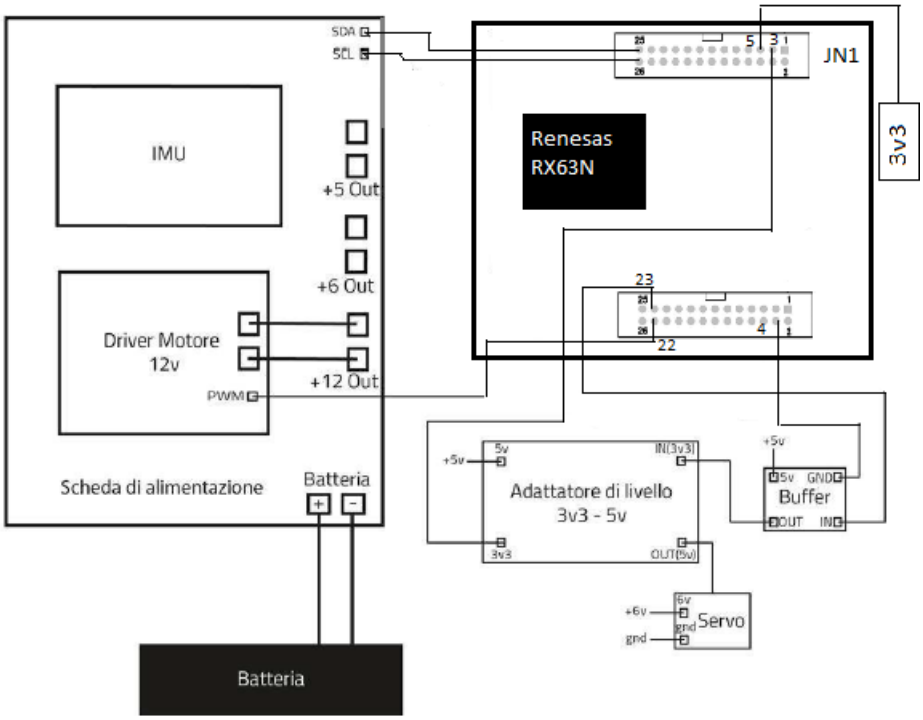


Figura 13. Schema dei collegamenti

3. MOTORI

I motori utilizzati sono:

- un motore DC con motoriduttore, inserito sulla corona anteriore, il quale genera una propulsione posteriore;
- un servomotore, montato in asse con lo sterzo, che regola l'angolo di sterzata.

3.1. Motore DC con Motoriduttore

DC	12 V
Output Power	3.4 W
Rated Speed	34 rpm
Rated Current	0.9 A
Rated Torque	980 mN m

Tabella 3. Specifiche del Motore DC con Motoriduttore

Figura 14. Motore DC con Motoriduttore

Il motore genera una coppia massima di 0.98 Nm, con una frequenza di 0.57 Hz (=34rpm); la velocitá angolare sull'albero motore sará dunque $\omega_M = 3.58 \frac{rad}{s}$

La velocitá tangenziale della corona é data da (indicando con R_A il raggio della corona):

$$V_A = \omega_M R_A \tag{3}$$

Sul pignone si avrá quindi una velocitá tangenziale $V_B = V_A$.

Poiché il rapporto dei raggi $\frac{corona}{pignone}$ é $\frac{18}{28}$ si deduce che:

$$\frac{\omega_M}{\omega_B} = \frac{18}{28} \tag{4}$$

dove R corrisponde al raggio della ruota.

Con tale espressione si determina la velocitá massima che il motore riesce a generare cioé circa $1\frac{m}{s}$. Poiché questa velocitá non é sufficiente ad autostabilizzare la bicicletta (velocitá minima per autostabilizzarla $1.8\frac{m}{s}$), occorre effettuare un controllo sul manubrio mediante attuazione dello stesso con un servomotore.

3.2. Servomotore

Figura 15. Servomotore

Il servomotore é costituito da:

- un motore elettrico completo di motoriduttore meccanico, che ha lo scopo di ridurre il numero di giri del motore cosí da aumentare la coppia motrice;
- un sistema di feedback per il controllo della posizione, realizzato con un potenziometro la cui resistenza varia alla posizione dellasse;
- l'elettronica di controllo che, tramite un opportuno sistema di comando, fa ruotare l'asse del servo fino a quando la posizione misurata dal potenziometro é uguale al valore imposto dal pin di controllo.

Il servomotore é dotato di tre poli, uno per l'alimentazione, uno per la massa ed uno per il controllo. Per indicare l'angolo al quale si deve portare il servomotore occorre inviare un segnale di questo tipo:

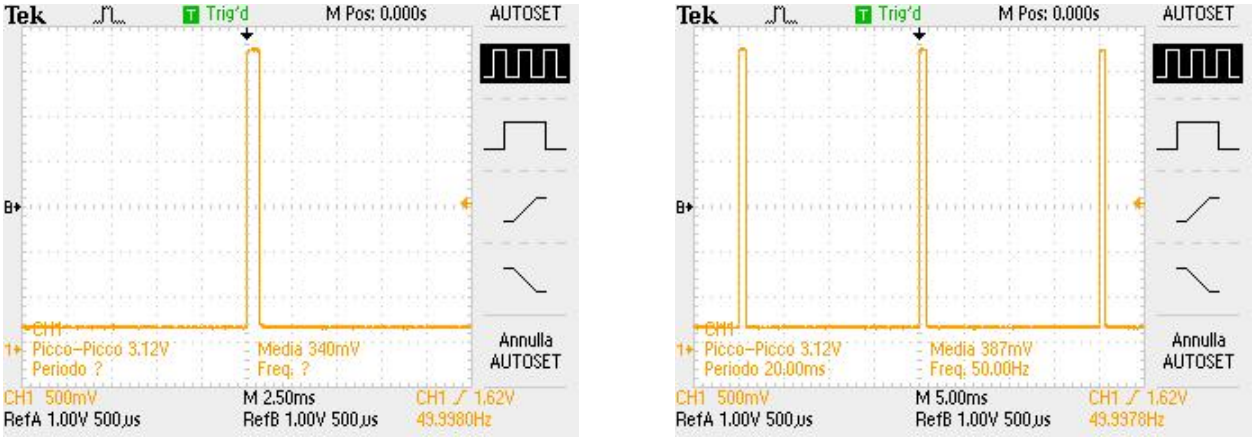


Figura 16. Grafici del duty cycle

Questo segnale é un'onda quadra TTL di frequenza 50 Hz che ha il duty cycle variabile e la cui durata indica il valore dell'angolo. Per esempio, se su un periodo di 20 ms l'angolo 0 gradi corrisponde a un duty cycle di 700 e 180 gradi corrisponde a 2062, facendo una proporzione é possibile ricavare la lunghezza dell'impulso corrispondente ad ogni angolo. I costruttori di servomotori forniscono sempre i valori delle durate minime e massime degli impulsi riconoscibili dal servo e i corrispondenti angoli. Di seguito viene riportato il datasheet del servomotore:

Control System	Pulse Width Control 1900 μs
Operating Voltage	4.8 - 7.2 V
Operating Speed (4.8 V)	0.20 sec / 60 degree
Operating Speed (6 V)	0.16 sec / 60 degree
Stall Torque (6 V)	28 kg.cm
Temperature Range	0 degree to 55 degree
Dead band width	5 μs
Gear type	metal
Size	66x30.2x64.4 mm
Weight	160 g
Wire lenght	32 cm

Tabella 4. Datasheet Servomotore

3.3. Problematica riscontrata

Uno dei problemi principali che abbiamo riscontrato si basa sul motore.

Come dicevamo nel paragrafo 5.1 (Motore DC con motoriduttore) a causa della bassa velocitá che il motore riesce a generare la bici stenta a tenersi in equilibrio. Perció abbiamo dedotto che una soluzione potrebbe essere quella di cambiare il motore con un altro la cui coppia massima e potenza sono piú elevate.

4. STRATEGIA DI CONTROLLO

La strategia di controllo utilizzata é stata di tipo proporzionale: abbiamo calcolato l'inclinazione massima a destra e a sinistra lungo l'asse verticale della bicicletta e abbiamo rapportato la scala dei valori con l'angolo di sterzata della ruota anteriore (manovrata dal servosterzo). Di seguito daremo una spiegazione dettagliata del procedimento che abbiamo seguito.

4.1. Spiegazione teorica

Per prima cosa abbiamo fatto stampare sul display della scheda i valori della variabile z (nel programma indicata con "zeta" e presa dall'accelerometro) che indica il valore dell'inclinazione verticale della bicicletta (che in gergo possiamo chiamare angolo di piegatura). Abbiamo osservato che il valore é pari a 0 quando la bicicletta é in posizione verticale e aumenta in valore assoluto man mano che l'inclinazione aumenta. Inoltre, un'inclinazione verso destra fa sí che la variabile zeta abbia segno negativo, mentre il segno é positivo nell'altro verso. A questo punto abbiamo fatto stampare, sempre sul display della scheda, i valori di rollk, che rappresenta il valore dell'angolo di inclinazione della bicicletta calcolato dal terreno (rollk = $0 \rightarrow$ bicicletta orizzontale) e abbiamo verificato che la bicicletta in posizione verticale ha rollk massimo (90 gradi), che diminuisce, in modo simmetrico, con l'aumentare dell'inclinazione, fino a un minimo di 83,65 gradi. Per quanto riguarda l'angolo di sterzata (nel programma indicato con "angolo"), esso varia da un minimo di 0 gradi (sterzo completamente ruotato verso destra) a un massimo di 120 gradi (sterzo completamente ruotato verso sinistra). Abbiamo quindi dovuto mappare i valori di "rollk" con quelli di "angolo". Come ultimo accorgimento, poiché rollk viene restituito dal codice in radianti, abbiamo dovuto operare una conversione preliminare da radianti a gradi per effettuare i calcoli.

4.2. Passaggi Matematici

Avendo a disposizione tutti i dati necessari, siamo stati in grado di convertire "rollk" in "angolo" utilizzando il seguente procedimento:

```
if(zeta>0){ //inclinazione verso destra
    angolo = (rollk*180/M_PI-83.65)*60/(2*M_PI); //calcolo dell'angolo di sterzata
}
else{ //inclinazione verso sinistra
    angolo = ((-(rollk*180/M_PI)+90)*60/(2*M_PI))+60; //calcolo dell'angolo di sterzata
}
```

Figura 17. Strategia di controllo (main)

Innanzitutto abbiamo dovuto dividere il problema in due parti: poiché "rollk" varia in modo identico a prescindere dal fatto che l'inclinazione sia verso destra o verso sinistra, abbiamo utilizzato "zeta" per distinguere i due casi:

- inclinazione verso destra ($z \leq 0$): l'angolo di sterzata varia da 0 a 60 gradi. Per operare la conversione della scala di valori si utilizza la seguente formula: (indicando con PI = 3,14)

$$angolo = \frac{60(\frac{180 rollk}{PI} - 83,65)}{2 * PI} \tag{5}$$

- inclinazione verso sinistra ($z \geq 0$): l'angolo di sterzata varia da 60 a 120 gradi. Per operare la conversione della scala di valori si utilizza la seguente formula: (indicando con PI = 3,14)

$$angolo = \frac{60(\frac{-180 rollk}{PI} + 90)}{2 * PI} + 60 \tag{6}$$

5. SOFTWARE

5.1. Main

Nel “Main” del programma, chiamato *riic_master_main.c*, una volta incluse tutte le librerie e gli header, dichiarate tutte le variabili, abbiamo cominciato ad inizializzare tutte le periferiche usate. L’LCD per la stampa dei valori sullo schermo della scheda; il servomotore per il movimento dello sterzo della bicicletta; un timer per la temporizzazione di alcuni eventi; il motore per il movimento della ruota posteriore; l’MPU per la lettura dell’angolo di inclinazione della bicicletta; ed il filtro di Kalman per la pulizia di alcuni dati dai rumori.
All’interno di un *while(1)*, codice quindi eseguito fino a quando la scheda sará alimentata, ogni 10 ms, attraverso *if(timer_10ms)*, con il quale si valuta la variabile *timer_10ms*, che ogni 10 ms diventa positiva (sezione 5.2), si salva nella variabile *zeta* il valore dell’inclinazione sull’asse verticale della bicicletta, calcolata dall’MPU: positiva se inclinata verso destra, negativa se verso sinistra. Successivamente si calcola *angolo* attraverso l’uso del filtro di Kalman (sezione 5.3) e questo valore viene passato alla funzione *Servo_Write_deg* (sezione 5.5) che ruota lo sterzo per posizionarsi all’angolo giusto.
Fuori dal *main* abbiamo inoltre inserito la funzione *get_rollk*, che verrá poi discussa nella sezione 5.3.

```
147  void main(void)
148  {
149
150      /* Variabile che stampa il valore di rollk in gradi */
151      float rollGRADI;
152
153      /* Declare and initialize the IIC result code. */
154      riic_ret_t iic_ret = RIIC_OK;
155
156      /* Initialize LCD */
157      lcd_initialize();
158
159      /* Clear LCD */
160      lcd_clear();
161
162      /* Inizializzazione del servomotore */
163      Servo_Init();
164
165
166      /* Prepare an RIIC channel for master mode communications. */
167      iic_ret |= riic_master_init();
168      while (RIIC_OK != iic_ret)
169      {
170          nop(); /* Failure to initialize here means demo can not proceed. */
171      }
172
173         /* Got a NACK. */
174
175         iic_ret = R_RIIC_Reset(RIIC_CHANNEL); /* Do soft reset to clean up bus states. */
176
177         if (RIIC_RESET_ERR & iic_ret) /* Check for successful IIC soft-reset. */
178         {   /* Soft-reset failed. Need to do complete re-initialization. */
179             /* Need to release the channel first berfore it can be re-initialized. */
180             R_RIIC_ReleaseChannel(RIIC_CHANNEL);
181             iic_ret = riic_master_init();
182         }
183
184         while (RIIC_OK != iic_ret)
185         {
186             nop(); /* Failure to initialize here means demo can not proceed. */
187         }
188
189      /* Inizializzazione del timer */
190      CMT_init();
191
192      /* Inizializzazione del motore */
193      PWMmotor_Init(1);
194
195      /* Inizializzazione dell'MPU */
196      MPU_init();
197
198      /* Inizializzazione del filtro di Kalman */
199      init_Kalman();
200
201      while (1)
202      {
203          /* I calcoli vengono svolti una volta ogni 10 ms, questo viene fatto con l'uso del timer_10ms */
204          if(timer_10mS){  //da provare anche 100mS
205              timer_10mS = 0;
206
207              /* la variabile zeta ci da l'inclinazione sull'asse verticale
208               * della bicicletta; l'inclinazone verso destra nel verso della
209               * bicicletta ha valore positivo, quella verso sinistra ha valore negativo */
210              zeta = return_z();
211
212
213              /* la variabile roll ci da l'angolo di inclinazione sull'asse verticale
214               * della bicicletta; l'inclinazone nulla ha valore di 90°, che scendono
215               * fino a 83° quando l'inclinazione è massima */
216              rollk=get_rollk();
217
218              /* Il calcolo dell'angolo di sterzata è stato svolto misurando empiricamente l'inclinazione massima
219               * della bicicletta visualizzata sul display della scheda */
220              if(zeta>0){  //inclinazione verso destra
221                  angolo = (rollk*180/M_PI-83.65)*60/(2*M_PI);  //calcolo dell'angolo di sterzata
222              }
223              else{ // inclinazione verso sinistra
224                  angolo = ((-(rollk*180/M_PI)+90)*60/(2*M_PI))+60; //calcolo dell'angolo di sterzata
225              }
226
227          lcd_display(LCD_LINE1, "    SELF    ");
228          lcd_display(LCD_LINE2, "  BALANCING  ");
229          lcd_display(LCD_LINE3, "   BICYCLE   ");
230
231          /* valore di rollk in gradi */
232          rollGRADI=rollk*180/M_PI;
233
234          /* stampa il valore dell'angolo di inclinazione passato al filtro di kalman in gradi*/
235          sprintf((char *)lcd_bufferProva, "roll=%.4f" , rollGRADI);
236          lcd_display(LCD_LINE7, lcd_bufferProva);
237
238          /* stampa il valore dell'angolo di sterzata */
239          sprintf((char *)lcd_bufferProva, "ang=%.1f" , angolo);
240          lcd_display(LCD_LINE8, lcd_bufferProva);
241
242
243          /* scrive nei registri del servomotore il valore dell'angolo di sterzata */
244          Servo_Write_deg(1,angolo);
245          }
246      }
247  }  /* End of function main() */
```

Figura 18. Main del programma

5.2. Timer

All'interno del file *CMT.c* ci sono delle funzioni per la gestione del clock. Attraverso la linea di comando 126 della Fig. 2, si istruisce il compilatore a considerare la routine *CMT_isr* come una ruotine speciale di interrupt. La variabile che a noi interessa, perché usata nel *Main*, é *timer_10ms*. Inizializzata a 0, questa, ogni volta che *timer_1ms* é multipla di 10, viene incrementata di 1. Poi all'interno del *Main* questa viene riportata a 0. *timer_10ms* oscilla quindi tra 0 e 1 (quando il timer é multiplo di 10 ms), ogni volta che é 1 vengono fatti tutti i calcoli per l'angolo di rotazione del servo, e viene fatto ruotare il servo. Questo timer é stato inserito cosí da 'rallentare' il servo per una migliore stabilizzazione.

```
126  #pragma interrupt (CMT_isr(vect = VECT(CMT0, CMI0)))
127  static void CMT_isr(void) {
128      static unsigned int timer_1mS = 0;
129
130      general_timer_mS++;
131      timer_1mS++;
132      if(!(timer_1mS % 10))
133        {
134          timer_10mS++;
135          if(!(timer_1mS % 100))
136            {
137              timer_100mS++;
138              if(!(timer_1mS % 500))
139                {
140                  timer_500mS++;
141                  if(!(timer_1mS % 1000))
142                    {
143                      timer_1S++;
144                      timer_1mS = 0;
145                    }
146                }
147            }
148        }
149
150
151  } /* End of CMT_isr() */
```

Figura 19. Funzione all'interno di CMT.c

```
128      while (1)
129      {
130          //I calcoli vengono svolti una volta ogni 10 ms, questo viene fatto con l'uso del timer_10ms
131
132          if(timer_10mS){ //timer (provare anche 100 mS)
133              timer_10mS = 0;
134
135              /* la variabile zeta ci da l'inclinazione sull'asse verticale
136               * della bicicletta; l'inclinazone verso destra nel verso della
137               * bicicletta ha valore positivo, quella verso sinistra ha valore negativo */
138              zeta = return_z();
```

Figura 20. Timer all'interno del Main

5.3. Servo

Questo tipo di motori viene alimentato separatamente e presenta un ingresso per la regolazione dell'angolo. É stato quindi necessario manipolare la periferica cosí da ottenere un segnale con frequenza 50hz e un periodo di 20ms.
Per comandare un servomotore occorre, tramite il filo di controllo, inviare allo stesso una serie di impulsi TTL. La durata del singolo impulso determina la posizione dell'asse di uscita. Il tempo di pausa, tra un impulso ed il successivo, puó variare entro ampi limiti senza che si abbia una perdita di controllo del servomotore. Dopo avere data una descrizione qualitativa cominciamo a dare delle indicazioni quantitative stabilendo dei precisi valori numerici. In funzione della durata di questo impulso il servomotore fará ruotare il perno di uscita, solidale con il potenziometro, fino al raggiungimento del completo equilibrio. Con una semplice proporzione, si ottiene l'angolo che si vuole mandare allo sterzo in microsecondi : dunque, a 0° corrispondono 700 us e a 120° corrispondono 2062 us (la durata minima e massima dell'impulso corrisponde ai due estremi dell'escursione del servomotore, che puó ruotare da 0 a 120 gradi).

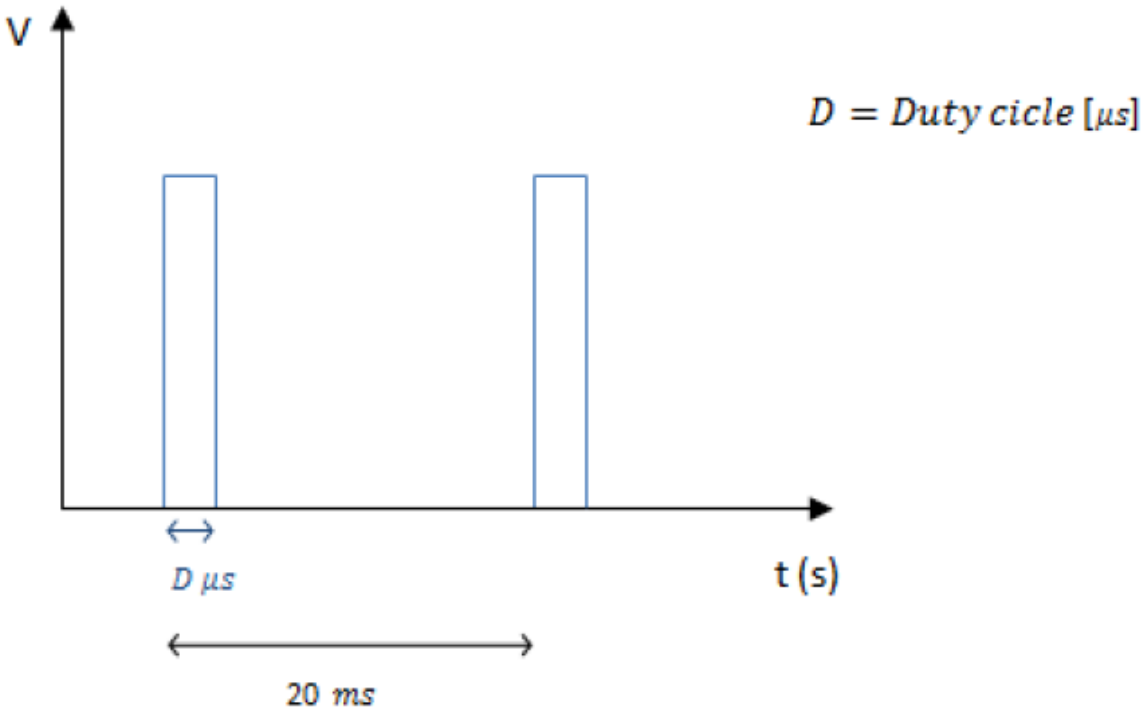


Figura 21. Duty cycle

Una volta inizializzata la periferica dal *Main*, ad ogni lettura dei dati dall'MPU e dopo aver tradotto l'angolo di inclinazione in angolo di rotazione del servo, questo angolo viene passato alla funzione *Servo_Write_deg*, che oltre a questo riceve il canale da utilizzare, il primo. In questa funzione si controlla se l'angolo che é stato ricevuto é un angolo appartenente all'intervallo accettabile, ossia compreso tra 0 e 120 gradi. Se l'angolo é al di fuori dei limiti, questo viene posto uguale al limite piú vicino. Successivamente la funzione richiama *Servo_Write_us*, a cui viene passato un valore dalla funzione *map* presente in *support_function.h*, che é la traduzione dell'angolo in microsecondi. *Servo_Write_us* richiama a sua volta *Servo_Write_PWM*, a cui invece viene passata la percentuale di microsecondi calcolati prima (duty cycle), sul periodo totale di 20 ms, che sará poi scritta nel registro del PWM del servomotore, cosí da farlo ruotare.

```c
/*****************************************************************************
* Function name: Servo_Init
* Description  : Inizializza i registri del servomotore
* Arguments    : none
* Return value : none
*****************************************************************************/
void Servo_Init()
{
    //Set all parameter needed by MTU4 unit and start count
    Set_MTU_U0_C4();
    StartCount_MTU_U0_C4();
}
/*****************************************************************************
* Function name: Servo_Write_PWM
* Description  : Scrive il valore value nel registro del PWM del servomotore
* Arguments    : channel, value
* Return value : none
*****************************************************************************/
void Servo_Write_PWM(int channel, float value)
{
    //channel value must be 1 or 2
    if(channel < 1 || channel > 2)  return;

    //check if value is a percentage, if not in range 1-99 return
    if(value<=1 || value>=99)   return;

    uint16_t tgr_a_val, tgr_b_val, tgr_d_val;
    tgr_a_val = GetTGR_A_MTU_U0_C4();

    switch(channel){
    case 1:
        tgr_b_val = (tgr_a_val*value)/100.0;
        SetTGR_B_MTU_U0_C4(tgr_b_val);
        break;
    case 2:
        tgr_d_val = (tgr_a_val*value)/100.0;
        SetTGR_D_MTU_U0_C4(tgr_d_val);
        break;
    }

}

void Servo_Write_us(int channel, float us)
{
    //channel value must be 1 or 2
    if(channel < 1 || channel > 2)  return;

    Servo_Write_PWM(channel, (us*100)/SERVO_PWM_SIGNAL_PERIOD_US);
}

void Servo_Write_deg(int channel, float deg)
{
    //channel value must be 1 or 2
    if(channel < 1 || channel > 2)  return;

    if(deg < SERVO_MIN_BOUND_DEG) deg = SERVO_MIN_BOUND_DEG;
    else if(deg > SERVO_MAX_BOUND_DEG) deg = SERVO_MAX_BOUND_DEG;

    //Servo_Write_us(channel, (((SERVO_MAX_US-SERVO_MIN_US)/(SERVO_MAX_DEG-SERVO_MIN_DEG))*deg-SERVO_MIN_DEG)+SERVO_MIN_
    Servo_Write_us(channel, map(deg, SERVO_MIN_DEG, SERVO_MAX_DEG, SERVO_MIN_US, SERVO_MAX_US));
}
```

Figura 22. Servo.c

5.4. MPU_driver

Dopo una prima inizializzazione, sará la funzione di riic_master_main.c *get_rollk*, che ogni 10 ms, richiamando la funzione *final_result_prova*, che a sua volta richiama una funzione di *MPU_driver*, ossia *MPU_read_raw*, aggiornerá i valori di accelerometro e giroscopio captati dall'MPU, da utilizzare poi per calcolare l'angolo di Roll (sezione 5.5). Tra questi valori, viene utilizzato *az* (ottenuto dalla funzione *return_z*), per capire se la bicicletta é inclinata verso destra o verso sinistra.

```
//Legge i valori grezzi di accelerazione dalla MPU6050
//ed eventualmente li stampa
riic_ret_t MPU_read_raw(void)
{
    //int16_t slope = 0;

    uint8_t    data[14];
    riic_ret_t ret;         /* Result code from the RIIC API functions. */
    uint8_t retry_count;

    /* Uses RIIC to read the accelerometer axis data. */
    ret = MPU_read(RIIC_CHANNEL, MPU_ADDRESS, INV_MPU6050_REG_RAW_ACCEL, data, 14);

    retry_count = 0;
    while (RIIC_OK != ret && retry_count < 4)
    {
        retry_count++;
        ret = MPU_read(RIIC_CHANNEL, MPU_ADDRESS, INV_MPU6050_REG_RAW_ACCEL, data, 14);
    }

    if(RIIC_OK != ret && retry_count >= 4)
     {
        ret = R_RIIC_Reset(RIIC_CHANNEL); /* Do soft reset to clean up bus states. */

        if (RIIC_RESET_ERR & ret) /* Check for successful IIC soft-reset. */
        {   /* Soft-reset failed. Need to do complete re-initialization. */
            /* Need to release the channel first before it can be re-initialized. */
            R_RIIC_ReleaseChannel(RIIC_CHANNEL);
        }

        ret = riic_master_init();

        ret = MPU_init();

        return ret;
     }

    a_x    = (int16_t)((((uint16_t)data[0]) << 8) & 0xFF00);
    a_x   |= data[1];

    a_y    = (int16_t)((((uint16_t)data[2]) << 8) & 0xFF00);
    a_y   |= data[3];

    a_z    = (int16_t)((((uint16_t)data[4]) << 8) & 0xFF00);
    a_z   |= data[5];

    g_x    = (int16_t)((((uint16_t)data[8]) << 8) & 0xFF00);
    g_x   |= data[9];

    g_y    = (int16_t)((((uint16_t)data[10]) << 8) & 0xFF00);
    g_y   |= data[11];

    g_z    = (int16_t)((((uint16_t)data[12]) << 8) & 0xFF00);
    g_z   |= data[13];

    /* Scostamenti dalla accelerazione iniziale                    */
    /* NOTA                                                        */
    /* Verificare che il cast da int32 a int16 dia risultati corretti */
    adjusted_x = a_x - (int16_t)ax_zero;
    adjusted_y = a_y - (int16_t)ay_zero;
    adjusted_z = a_z - (int16_t)az_zero;

    return ret;
}
```

```
//Legge i registri di accelerometro e giroscopio simultaneamente
void final_result_prova(float *gxp,float *gyp,float *gzp,float *axp,float *ayp,float *azp)
{
    //dati grezzi dalla MPU
    MPU_read_raw();

    //Conversione dei dati come da datasheet della MPU
    //calcolo valori finali per il giroscopio
    *gxp=(((float)g_x)/65.4);
    *gyp=(((float)g_y)/65.4);
    *gzp=(((float)g_z)/65.4);
    //valori di accelerazione a meno dello scostamento iniziale
    //Quali devo usare?
    //*axp=(((float)adjusted_x)/16384);
    //*ayp=(((float)adjusted_y)/16384);
    //*azp=(((float)adjusted_z)/16384);
    *axp=(((float)a_x)/8192);
    *ayp=(((float)a_y)/8192);
    *azp=(((float)a_z)/8192);

    return;
}
```

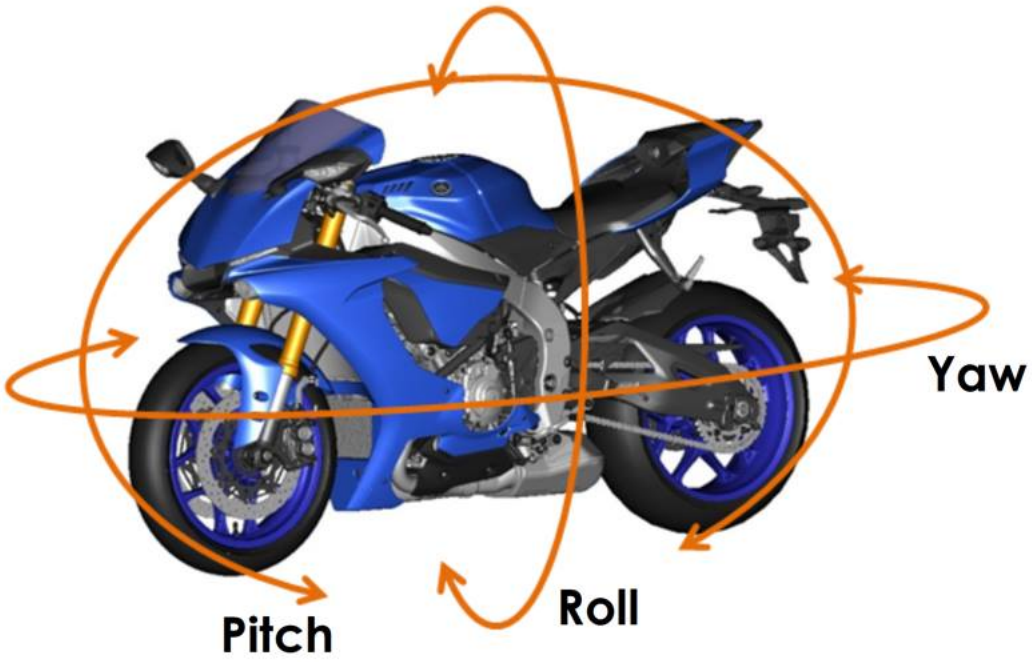
Figura 23. Funzioni MPU_read_raw e final_result_prova all'interno di MPU_driver.c

5.5. Filtro di Kalman

Il filtro di Kalman é un efficiente filtro ricorsivo che valuta lo stato di un sistema dinamico a partire da una serie di misure soggette a rumore. Per le sue caratteristiche intrinseche é un filtro ottimo per rumori e disturbi agenti su sistemi gaussiani a media nulla. Trova utilizzo come osservatore dello stato, come loop transfer recovery (LTR) e come sistema di identificazione parametrica.

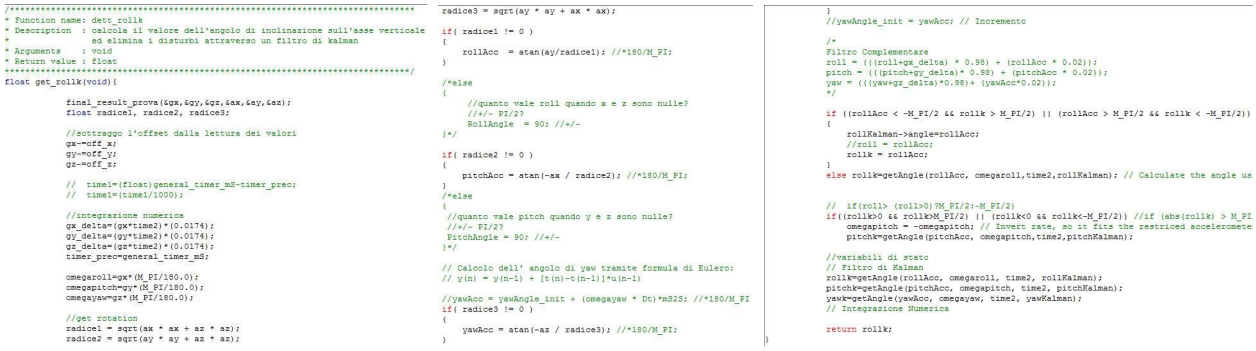
Ci siamo serviti del filtro di Kalman per eliminare disturbi e rumori nel calcolo dell'angolo di inclinazione della bicicletta sull'asse verticale. Abbiamo fatto dei test con l'angolo roll (senza filtro di Kalman) e abbiamo confrontato i risultati con l'angolo rollk (passato al filtro di Kalman): abbiamo cosí potuto osservare che la bicicletta risponde molto meglio nel secondo caso. Nel codice é presente l'header file Kalman.h e il file sorgente Kalman.c; in quest'ultimo sono presenti l'inizializzazione del filtro e le funzioni getAngle() e setAngle(), che sono funzioni di supporto. Nel file sorgente riic_master_main.c é presente invece, nonostante essa sia parte integrante del filtro di Kalman, la funzione get_rollk() che restituisce l'angolo rollk come float. Essa costituisce la parte piú interessante del filtro: calcola i valori dell'inclinazione della bicicletta lungo i tre assi e svolge i calcoli necessari per ricavare tre parametri:

- rollk: angolo di inclinazione (quello che ci serve)
- pitchk: angolo di impennata
- yaw: angolo di rotazione orizzontale



(La funzione get_rollk() insieme all'inizializzazione delle variabili utilizzate in essa, sono state inserite nello stesso sorgente del main poiché abbiamo riscontrato dei problemi nei calcoli del valore di rollk quando essa era stata inserita nel sorgente Kalman.c e poi richiamata nel main)

Nonostante i valori di pitchk e yaw non ci servano, li abbiamo lasciati per completezza, nel caso in cui qualcuno possa essere interessato al driver del filtro di Kalman per un altro progetto.



```
/**************************************************************************
* Function name: dett_rollk
* Description   : calcola il valore dell'angolo di inclinazione sull'asse verticale
*                 ed elimina i disturbi attraverso un filtro di kalman
* Arguments     : void
* Return value : float
**************************************************************************/
float get_rollk(void){

          final_result_prova(&gx,&gy,&gz,&ax,&ay,&az);
          float radice1, radice2, radice3;

          //sottraggo l'offset dalla lettura dei valori
          gx-=off_x;
          gy-=off_y;
          gz-=off_z;

          //  time1=(float)general_timer_mS-timer_prec;
          //  time1=(time1/1000);

          //integrazione numerica
          gx_delta=(gx*time2)*(0.0174);
          gy_delta=(gy*time2)*(0.0174);
          gz_delta=(gz*time2)*(0.0174);
          timer_prec=general_timer_mS;

          omegaroll=gx*(M_PI/180.0);
          omegapitch=gy*(M_PI/180.0);
          omegayaw=gz*(M_PI/180.0);

          //get rotation
          radice1 = sqrt(ax * ax + az * az);
          radice2 = sqrt(ay * ay + az * az);
          radice3 = sqrt(ay * ay + ax * ax);

          if( radice1 != 0 )
          {
              rollAcc  = atan(ay/radice1); //*180/M_PI;
          }

          /*else
          {
               //quanto vale roll quando x e z sono nulle?
               //+/- PI/2?
               RollAngle  = 90; //+/-
          }*/

          if( radice2 != 0 )
          {
              pitchAcc = atan(-ax / radice2); //*180/M_PI;
          }
          /*else
          {
           //quanto vale pitch quando y e z sono nulle?
           //+/- PI/2?
           PitchAngle = 90; //+/-
          }*/

          // Calcolo dell' angolo di yaw tramite formula di Eulero:
          // y(n) = y(n-1) + [t(n)-t(n-1)]*u(n-1)

          //yawAcc = yawAngle_init + (omegayaw * Dt)*mS2S; //*180/M_PI
          if( radice3 != 0 )
          {
              yawAcc = atan(-az / radice3); //*180/M_PI;
          }
          //yawAngle_init = yawAcc; // Incremento

          /*
          Filtro Complementare
          roll = (((roll+gx_delta) * 0.98) + (rollAcc * 0.02));
          pitch = (((pitch+gy_delta)* 0.98) + (pitchAcc * 0.02));
          yaw = (((yaw+gz_delta)*0.98)+ (yawAcc*0.02));
          */

          if ((rollAcc < -M_PI/2 && rollk > M_PI/2) || (rollAcc > M_PI/2 && rollk < -M_PI/2))
          {
              rollKalman->angle=rollAcc;
              //roll = rollAcc;
              rollk = rollAcc;
          }
          else rollk=getAngle(rollAcc, omegaroll,time2,rollKalman); // Calculate the angle us


          //  if(roll> (roll>0)?M_PI/2:-M_PI/2)
          if((rollk>0 && rollk>M_PI/2) || (rollk<0 && rollk<-M_PI/2)) //if (abs(rollk) > M_PI
              omegapitch = -omegapitch; // Invert rate, so it fits the restriced accelerometer
              pitchk=getAngle(pitchAcc, omegapitch,time2,pitchKalman);

          //variabili di stato
          // Filtro di Kalman
          rollk=getAngle(rollAcc, omegaroll, time2, rollKalman);
          pitchk=getAngle(pitchAcc, omegapitch, time2, pitchKalman);
          yawk=getAngle(yawAcc, omegayaw, time2, yawKalman);
          // Integrazione Numerica

          return rollk;
}
```

Figura 24. Funzione: get_rollk()

Per quanto riguarda il main, viene eseguita l'inizializzazione del filtro di Kalman:

```
/* Inizializzazione del filtro di Kalman */
init_Kalman();
```

e in seguito il valore restituito da get_rollk() viene messo nella variabile float rollk:

```
/* la variabile roll ci da l'angolo di inclinazione sull'asse verticale
 * della bicicletta; l'inclinazone nulla ha valore di 90°, che scendono
 * fino a 83° quando l'inclinazione è massima */
rollk=get_rollk();
```

Quindi rollk vieni utilizzato per eseguire il controllo come spiegato nel capitolo 4: Strategia di controllo.

6. CONCLUSIONI

La prima cosa che abbiamo fatto é stata quella di sostituire la vecchia scheda con la nuova (RENESAS RX63N) e convertire tutti i collegamenti tra questa e le periferiche attraverso gli opportuni pin descritti nella sezione 2.10. A questo punto abbiamo riscritto da capo il codice, utilizzando in un primo momento l'accelerometro della scheda, e in seguito l'MPU per calcolare l'inclinazione dell'asse verticale della bicicletta. Abbiamo scritto tutti i driver (MPU, Motore, Servomotore, CMT, Kalman) e, una volta testati singolarmente, li abbiamo uniti fino a creare il progetto definitivo che rende possibile un parziale controllo della bicicletta.

Come suggerito dal gruppo che ha precedentemente lavorato su questo progetto, abbiamo utilizzato un filtro di Kalman per migliorare la stabilitá della bicicletta ed eliminare i disturbi. Nonostante dei miglioramenti, le modifiche apportate non sono state sufficienti a garantire l'equilibrio a causa di diversi problemi riscontrati.

Per migliorare ulteriormente il progetto e permettere il perfetto equilibrio della bicicletta é necessario:

- aumentare la velocitá, utilizzando un motore DC piú performante, come suggerito nella sezione 3.1;
- utilizzare, oltre al controllo proporzionale giá implementato nel codice, un controllore PID, che garantisce la stabilitá necessaria.

Si potrebbe inoltre pensare di sostituire la bicicletta con una di dimensione maggiore, che dovrebbe essere piú controllabile, ed aggiungere un joystick wireless (con relativo driver) per controllare la bicicletta a distanza e avere una maggiore libertá di utilizzo.

Riferimenti bibliografici

[1] Relazione "Self Balancing Bicycle" - Professore: Andrea Bonci; Studenti: M.L.Augello, S. Battipane, S. Lucchitti, F. Mambella, D. Pasqualini - A.A 2012/2013

[2] RX63N631 HARDWARE MANUAL

[3] Schematics YRDKRX63N