UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Integrazione di Hardware e Firmware per il controllo di una e-bike a guida autonoma

Hardware and Firmware integration for the control of a self-driving e-bike

Candidato:
Andrea Perugini

Relatore:
Andrea Bonci

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Integrazione di Hardware e Firmware per il controllo di una e-bike a guida autonoma

Hardware and Firmware integration for the control of a self-driving e-bike

Candidato:
Andrea Perugini

Relatore:
Andrea Bonci

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'obiettivo di questa tesi è progettare un sistema di controllo per una e-Bike a guida autonoma, quindi senza l'intervento umano. Per il suo sviluppo, verranno utilizzati progetti precedentemente realizzati da studenti del corso di Ingegneria Informatica e dell'Automazione. Il sistema di controllo si concentrerà sul mantenimento dell'equilibrio della bicicletta, garantendo una velocità costante e un'andatura rettilinea.

Per permettere alla bicicletta di stare in equilibrio si hanno a disposizione lo sterzo della ruota anteriore e la trazione della ruota posteriore. Per entrambi sono stati realizzati dei regolatori digitali e in questa tesi si cercherà di applicarli in modo da mantenere l'equilibrio della bicicletta.

Un particolare controllore digitale è il PID (Proporzionale Integrativo Derivativo), il cui funzionamento è basato sulla somma di tre termini, rispettivamente uno proporzionale al segnale, uno alla sua derivata e uno alla sua integrazione.

Sia il PID dedicato allo sterzo che il PID dedicato alla trazione saranno tarati nuovamente dal momento che non erano funzionanti.

Nel capitolo 2 viene esaminato il sistema fisico, con un'introduzione ai regolatori PID e agli angoli di Eulero, oltre ad una descrizione del sistema di controllo proposto.

Il capitolo 3 illustra le componenti hardware utilizzate, fornendo dettagli sulle loro caratteristiche e sui collegamenti.

Nel capitolo 4 si analizza il diagramma di flusso del controllore, con particolare attenzione alle periferiche e alle librerie impiegate.

Il capitolo 5 approfondisce la catena di controllo, con un'analisi dettagliata di ciascun PID.

Il capitolo 6 descrive i metodi di comunicazione utilizzati nel progetto e i codici Matlab e C che ne gestiscono l'implementazione.

Infine, nel capitolo 7, vengono discussi i test eseguiti, le problematiche incontrate e le possibili direzioni per sviluppi futuri.

Capitolo 2

Sistema

L'e-Bike presa in considerazione è costituita da una bicicletta di piccole dimensioni a cui sono stati applicati dei motori e delle componenti hardware per controllarli. Un motore è dedicato alla rotazione dela corona centrale e serve per la trazione della bicicletta. L'altro motore è stato posizionato nella parte superiore della bicicletta come sostituto del manubrio, in modo tale che la sua rotazione permetta lo sterzo. Sono stati montati due pannelli, uno sul lato destro e uno sul lato sinistro della bicicletta, sui quali sono state installate le componenti hardware, come mostrato nelle figure 2.1 e 2.2.

Per il controllo dell'assetto della bicicletta, utilizzeremo gli "angoli di Eulero" e i "PID". Pertanto, verranno dedicate due sezioni all'introduzione di questi concetti.

Figure 2.1: *Foto bicicletta del lato destro*

Figure 2.2: *Foto bicicletta del lato sinistro*

2.1 Angoli di Eulero

Per un corpo rigido che si trova nello spazio tridimensionale è possibile definire i cosiddetti angoli di eulero. Gli angoli sono:

1. Angolo di **Roll** ϕ:
   descrive la rotazione attorno all'asse longitudinale del corpo. Per una bicicletta, il rollio si verifica quando la bicicletta si inclina a destra o a sinistra rispetto alla sua direzione di marcia.

2. L'angolo di **Yaw** ψ:
   descrive la rotazione di un veicolo attorno al suo asse verticale, che passa per il centro del veicolo dall'alto verso il basso. Per una bicicletta, l'imbardata rappresenta l'orientamento della bicicletta rispetto alla direzione del movimento su un piano orizzontale. Questo angolo si modifica quando la bicicletta cambia direzione verso destra o sinistra.

3. Angolo di **Pitch** θ:
   descrive la rotazione attorno all'asse trasversale del corpo. È l'inclinazione verso l'alto o verso il basso, nel caso della bicicletta, è l'angolo ottenuto quando la bicicletta è in salita o discesa.

Figure 2.3: *Angoli di eulero*

2.2 PID

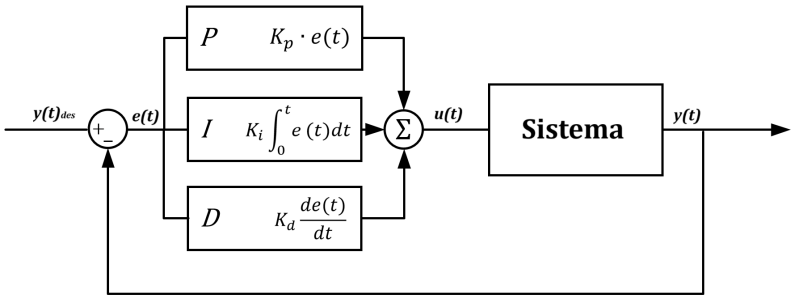

Figure 2.4: *Schema a blocchi di un PID*

Un controllore PID è un meccanismo di controllo in catena chiusa con retroazione dall'uscita (feedback) ampiamente utilizzato nei sistemi di controllo industriali. L'idea alla base del funzionamento di questo tipo di controllori è di calcolare continuamente il valore di errore che per definizione è data dalla differenza tra il setpoint desiderato (uscita desiderata) e l'uscita reale del processo (variabile da controllare). In particolare, nei regolatori PID, il controllo è dato dalla somma dei termini Proporziale , Integrale e Derivativo. L'equazione associata ad un PID è di questo tipo:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)\, d\tau + K_D \frac{de(t)}{dt} \tag{2.1}$$

dove:

- $u(t)$ è l'ingresso da fornire al sistema.
- $e(t) = y_{des}(t) - y(t)$ è l'errore tra l'uscita desiderata del sistema e il suo effettivo valore.
- K_P, K_I, K_D sono delle costanti che vanno determinate.

L'ingresso $u(t)$ dell'eq 2.1 può essere visto come la somma di 3 contributi:

- $u_P = K_P e(t)$ l'azione proporzionale.
- $u_I = K_I \int_0^t e(\tau)\, d\tau$ l'azione integrale.
- $u_D = K_D \frac{de(t)}{dt}$ l'azione derivativa.

quindi

$$u(t) = u_P(t) + u_I(t) + u_D(t) \tag{2.2}$$

2.2.1 Azione Proporzionale

L'azione proporzionale moltiplica l'errore $e(t)$ per una costante K_P. All'aumentare di quest'ultimo valore, lo sforzo di controllo diventa maggiore e il sistema diventerà più reattivo, ovvero seguirà il riferimento (la risposta desiderata) più velocemente. Tuttavia un valore di K_P troppo grande può portare il sistema ad oscillazioni e quindi indurre l'instabilità.

Generalmente l'azione proporzionale non è sufficiente al controllo dell'uscita del sistema. Ciò dipende come varia $y(t)_{des}$ e dal sistema. Se $y(t)_{des}$ è una costante, a regime si otterrà una risposta cosi fatta:

$$y(t) = y(t)_{des} + offset \tag{2.3}$$

Per questo motivo l'azione proporzionale solitamente non basta e si combina con l'azione integrale.

2.2.2 Azione Integrale

L'azione integrale moltiplica l'integrale $\int_0^t e(\tau)\, d\tau$ per una costante K_I. L'integrale può essere visto come la somma degli errori per ogni istante di tempo, ciò significa che u_I tende ad un valore finito solo quando l'errore tende a 0. Quando l'integrale raggiunge lo 0 e l'errore viene mantenuto 0, l'offset dell'Eq 2.3 viene compensato e a regime si ottiene:

$$y(t) = y(t)_{des} \tag{2.4}$$

2.2.3 Azione Derivativa

L'azione derivativa è data dal prodotto tra $\frac{de(t)}{dt}$ e K_D. La derivata considera il tasso di variazione dell'errore, ciò indica che quando l'errore varia velocemente, l'azione derivata è grande, mentre quando l'errore è costante (possibilmente 0) l'azione derivata è nulla.

Il suo comportamento è quello di predirre il comportamento dell'errore e cercare di renderlo costante più rapidamente. Questa azione richiede che la misura $y(t)$ del sistema sia senza rumore, altrimenti la derivata $\frac{de(t)}{dt}$ può assumere valori molto grandi e il controllo può indurre instabilità.

2.3 Sistema di controllo

In questa tesi non è stato considerato il modello della bicicletta, data la complessità del sistema. Nell'articolo del professor Bonci [7], è stato esaminato il funzionamento di un sistema di controllo a doppio anello progettato per garantire il mantenimento di una traiettoria desiderata di una motocicletta. Le simulazioni del modello della motocicletta hanno permesso la corretta progettazione dei PID che permettessero al sistema di percorrere una traiettoria mantenendo l'equilibrio.

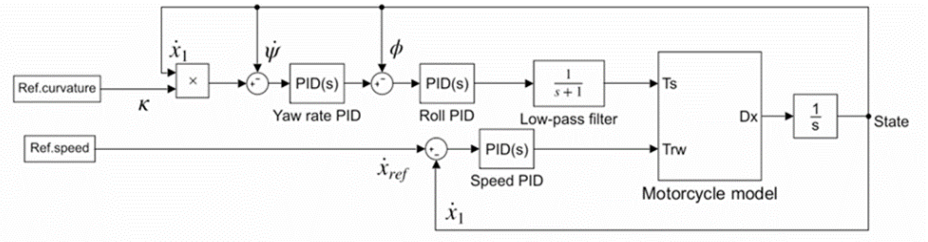

Figure 2.5: *Schema di controllo dell'articolo*

Lo schema di controllo esaminato è rappresentato nella figura 2.5

Se in questo articolo l'obiettivo era quello di far percorrere al veicolo una traiettoria desiderata, nella presente tesi verrà invece garantito il primo sotto-obiettivo essenziale, ovvero mantenere la bicicletta in equilibrio, cioè $\phi = 0$. Inoltre nell'articolo si tiene conto di applicare direttamente una coppia allo sterzo della motocicletta, mentre a noi non è nota la dinamica tra la tensione applicata al motore dedicato allo sterzo e la coppia generata, e quindi si introduce un PID per generare la coppia desiderata. Il nuovo schema di controllo è mostrato nella figura 2.6

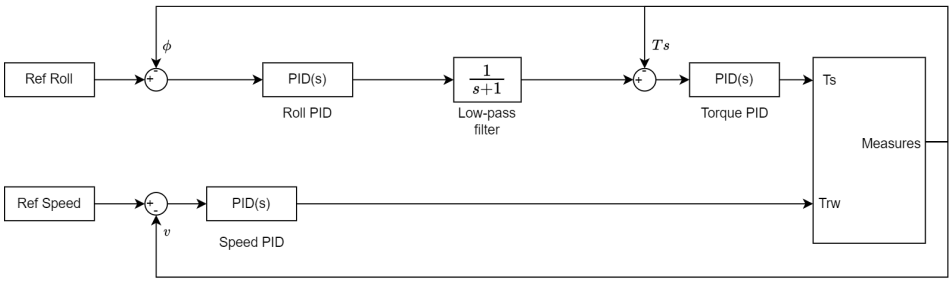

Figure 2.6: *Schema di controllo della tesi*

I blocchi mostrati nello schema possono essere divisi in 2 sottogruppi, uno riguardante lo sterzo e uno riguardante la trazione.

Per il controllo della trazione, viene impostata una velocità di riferimento per la ruota posteriore, definita manualmente. La differenza tra questa velocità e quella effettivamente rilevata dai sensori della bicicletta viene utilizzata come ingresso per lo "Speed PID", che calcola la tensione da applicare al motore dedicato alla trazione.

Il controllo dello sterzo è più complesso. L'obiettivo è applicare una coppia alla ruota anteriore per mantenere un angolo di rollio di riferimento. La differenza tra l'angolo di rollio desiderato e quello misurato costituisce l'ingresso del "Roll PID", che determina la coppia da applicare alla e-Bike. Questa coppia viene poi filtrata attraverso un filtro passa basso, che consente il passaggio dei segnali a bassa frequenza, attenuando quelli ad alta frequenza.

L'applicazione della coppia filtrata alla bicicletta richiede un'ulteriore gestione, poiché l'unica ingresso accessibile è la tensione ai capi del motore. Non disponendo di un modello matematico che colleghi direttamente la tensione del motore alla coppia che genera, si utilizza il "PID Torque". Questo PID calcola la tensione da applicare al motore in base all'errore tra la coppia filtrata desiderata e la coppia misurata dai sensori, garantendo il controllo della coppia trasmessa.

In seguito nella tesi:

1. Il PID Torque sarà denominato PID Sterzo.
2. Il PID Speed sarà denominato PID Trazione.

Capitolo 3

Hardware

In questo capitolo verranno descritte le componenti hardware usate nella e-Bike. In particolare sono presentate:

- scheda microcontroller
- motore
- driver
- encoder
- sensore di corrente
- IMU
- Esp32
- regolatori
- batteria

3.1 Scheda microcontroller

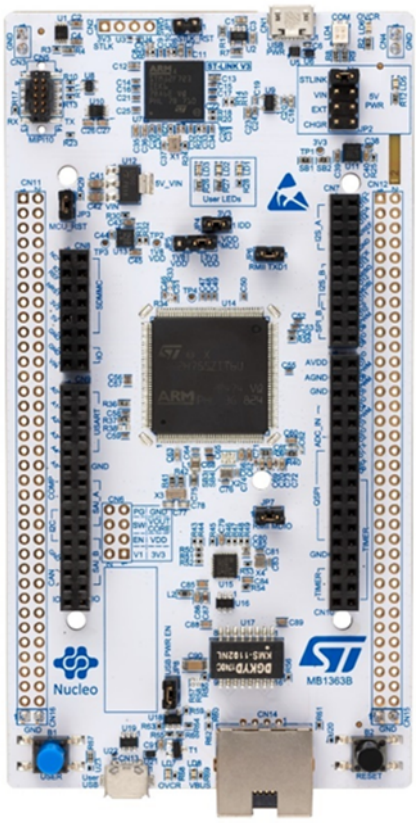

Figure 3.1: *Microcontrollore STM32*

Un microcontrollore (dall'inglese microcontroller, o MCU - MicroController Unit), è un dispositivo elettronico integrato su singolo circuito elettronico, nato come evoluzione alternativa al microprocessore e utilizzato generalmente in sistemi embedded, ovvero per applicazioni specifiche di controllo digitale [9].

Il microcontrollore utilizzato è NUCLEO-144 STM32H7 45 zi-q *(fig. 3.1)*, avente 2 processori a 32-bit:

1. ARM Cortex M4
2. ARM Cortex M7 (che non verrà utilizzato)

La scheda dispone anche di periferiche per permettere al microcontroller di interagire con dispositivi esterni. Alcune periferiche da evidenziare:

1. l'ADC (Analog-to-digital converter)
2. l'interfaccia USB

3. i TIMER
4. porte usate per generare segnali PWM (Pulse Width Modulation)
5. un User-Button (bottone blu)

3.2 Motore

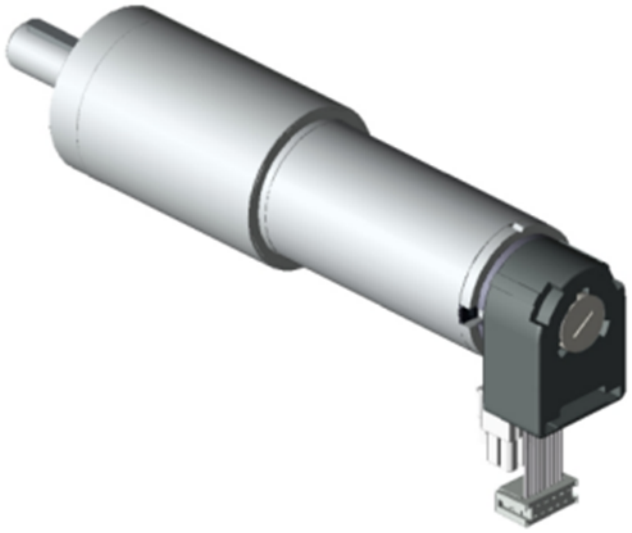

Figure 3.2: *Motor - DCX35L GB KL 18V*

Nella bicicletta sono stati montati 2 motori, uno per lo sterzo e uno per la trazione. Le caratteristiche principali sono [6]:

1. Costante di coppia = 23,4 $mNmA^{-1}$;
2. Resistenza terminale = 0,212 Ω;
3. Inerzia del rotore = 102 gcm^2;
4. Induttanza terminale = 0,0774 mH;
5. Tensione nominale = 18 V;
6. Riduzione = 1:66;

3.3 Driver

Il driver del motore è un componente elettronico che regola la potenza fornita al motore. In pratica, controlla la tensione e quindi la potenza che il motore riceve. La

sua funzione principale è quella di tradurre il segnale di controllo (in questo caso un segnale PWM) in un voltaggio e corrente che il motore può utilizzare.

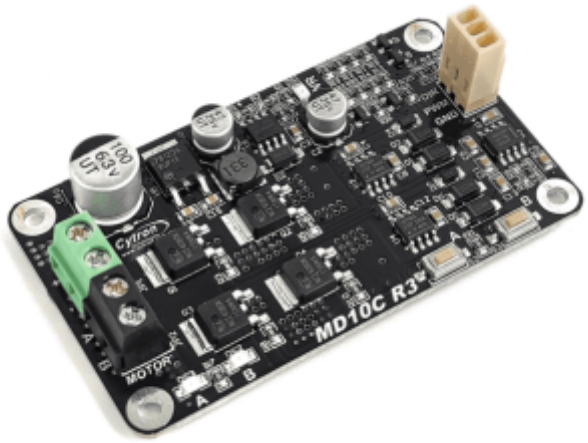

Figure 3.3: *Driver MD10C*

Specifiche [5]	
tensione motore	da 5V a 40V DC
corrente continua max	13Amps
corrente di picco	30Amps (10s)
canale di output	Motore mono-spazzola
input logico	da 3.3V a 5.0 V DC
segnale di controllo	PWM

Table 3.1: *Specifiche Driver*

3.4 Encoder

Figure 3.4: *Encoder HEDL5540 500IMP*

L'encoder è un componente elettronico che traduce un movimento fisico (in questo caso la rotazione dell'albero del motore) in impulsi elettrici. Questi impulsi vengono poi elaborati per calcolare parametri come la velocità, la distanza percorsa, o la direzione del movimento. L'encoder che è stato utilizzato è un ENC30 HEDL 5540 [4]. Esso è dotato di 10 pin, ma per la e-Bike ne vengono utilizzati solo 4:

1. l'alimentazione a 5V.
2. il GND.
3. il channel A.
4. il channel B.

Entrambi i motori sono provvisti di un encoder, nella tabella 3.2 vengono presentati i collegamenti di ognuno di questi.

Pin Encoder	**Encoder sterzo**	**Encoder trazione**
5V	5V della STM	5V generale della e-Bike
GND	GND generale della e-Bike	GND della STM
CHANNEL A	PD13	PC7
CHANNEL B	PD12	PC6

Table 3.2: *Tabella collegamenti dei 2 encoder*

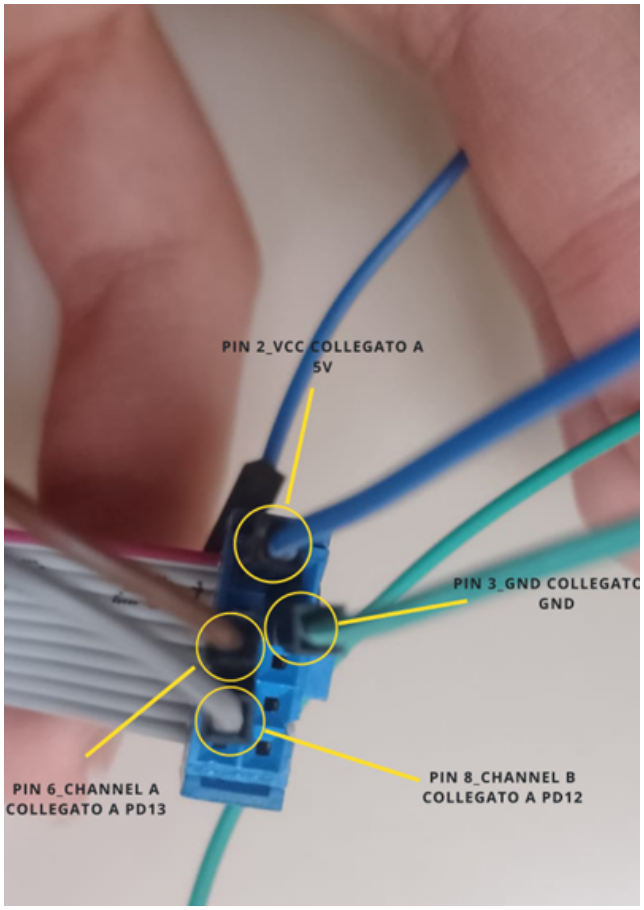

Figure 3.5: *Pin usati per l'Encoder della trazione. Sono gli stessi anche nello sterzo*

3.5 Sensore di corrente

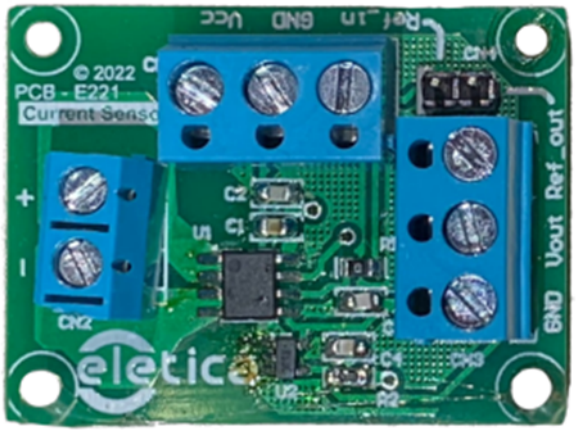

Figure 3.6: *Sensore di corrente TMCS1101A3B*

Il sensore di corrente è un componente elettronico che misura la corrente che passa attraverso un circuito. Il sensore rileva la corrente e produce un segnale in tensione proporzionale alla quantita di corrente misurata. Il sensore usato è il TMCS1101A3B. La relazione tra la corrente misurata e la tensione generata è lineare: $I = 4.6V - 11.425$
La corrente del circuito entra dal pin corrispondente al + e esce dal -. Il sensore viene alimentato con 5V dalla scheda STM. I valori massimi di corrente misurabili sono $\pm 11.25A$. L'obbiettivo è quello di misurare la coppia che viene generata dal motore dedicato allo sterzo. Sapendo che la coppia $C = K \cdot I$, dove $K = 0.0234Nm$ è una costante del motore, possiamo ottenere facilmente la coppia tramite il sensore di corrente.

Purtroppo i valori in output del sensore sono soggetti a rumore quindi è stato necessario applicare un filtro di Kalman introducendo dei ritardi nel sistema.

3.6 IMU

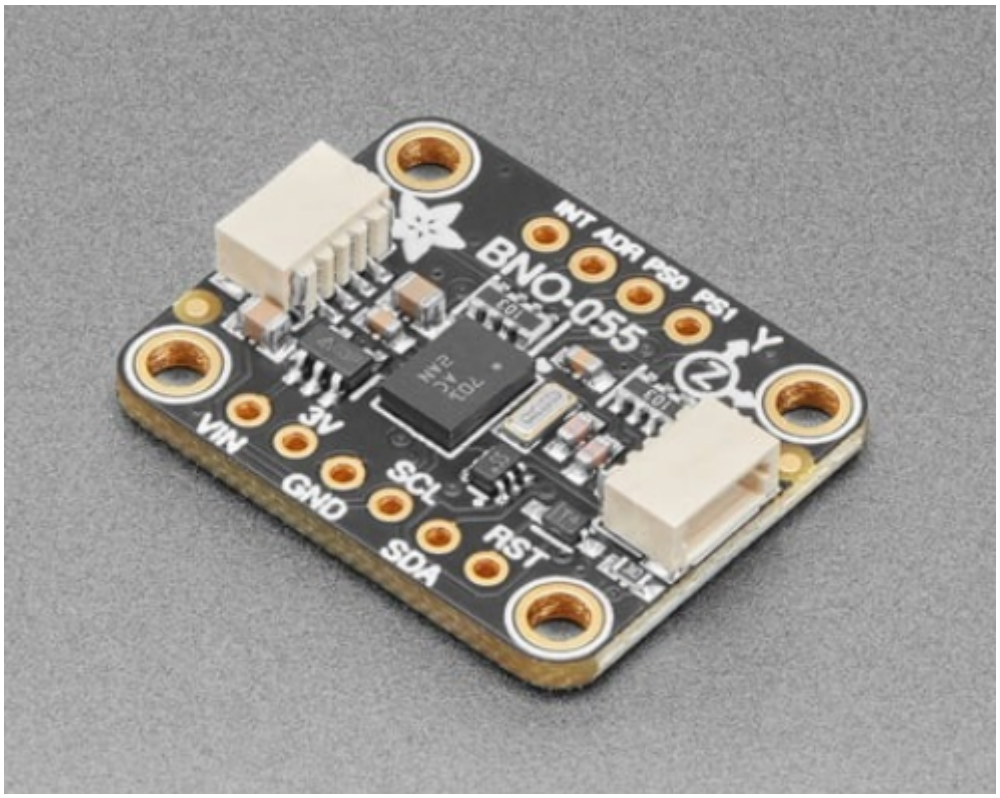

Figure 3.7: *Imu BNO055*

Una IMU (Inertial Measurement Unit, in italiano "Unità di Misura Inerziale") è un dispositivo elettronico che misura e riporta l'accelerazione specifica, la velocità angolare, e talvolta l'orientamento di un oggetto. L'IMU utilizzato è il sensore BNO055, il quale integra un accelerometro triassiale, un giroscopio triassiale e un

magnetometro triassiale, offrendo una soluzione completa per le misurazioni di orientamento e movimento in tre dimensioni.

In questo progetto l'IMU è stato usato per misurare gli angoli di eulero della bicicletta

3.7 Esp32

L'Esp32 è un microcontrolore che è stato utilizzato per gestire la comunicazione wireless tra microcontrollore e computer. Esso viene alimentato con 3.3V e offre la possibilita di stabilire una connessione bluetooth o wifi.

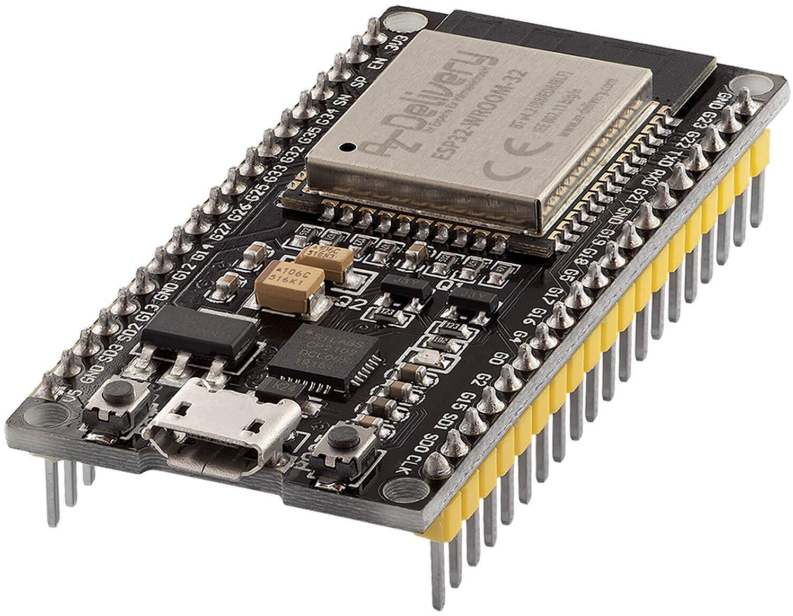

Figure 3.8: *Enter Caption*

I pin utilizzati sono quelli riportati nella tabella 3.3

Pin Esp32	Collegamento
3.3V	3.3V generale della e-Bike
GND	GND generale della e-Bike
G16	PA3
G17	PD5

Table 3.3: *Tabella collegamenti dell'Esp32*

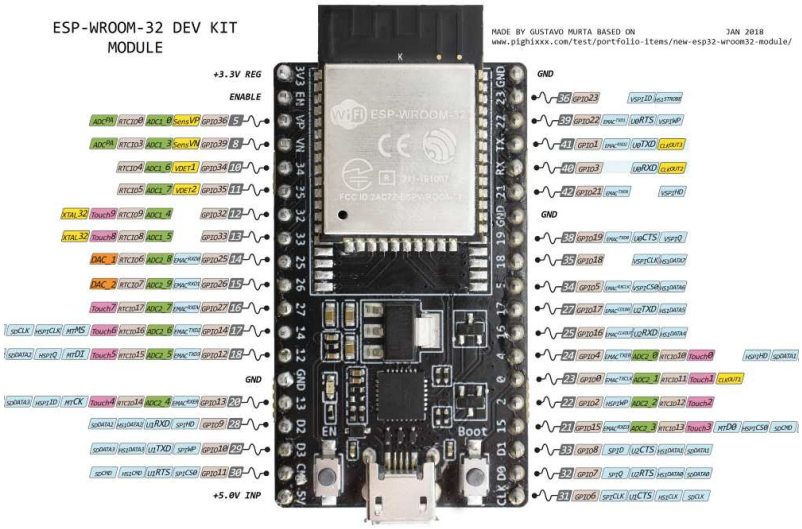

Figure 3.9: *Pinout Esp32*

3.8 Regolatori

I 3 regolatori presenti sull'e-Bike servono per convertire la tensione di alimentazione a 16.8V, proveniente dalla batteria, in tensione in uscita a 3.3V, 5V e 12V.

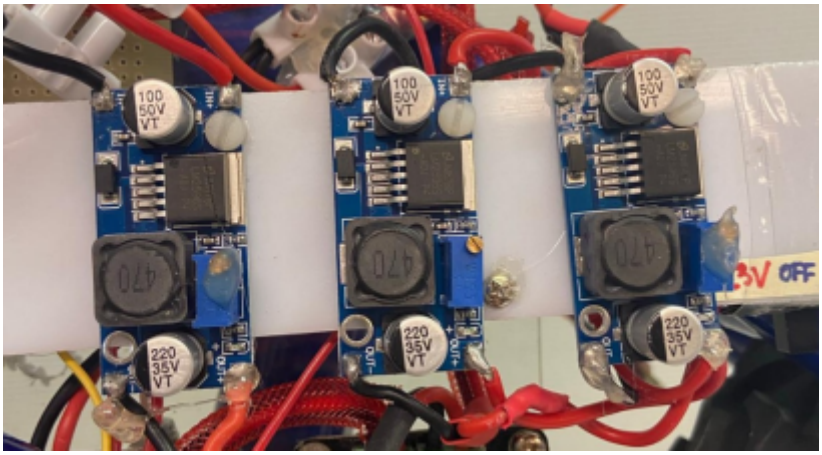

Figure 3.10: *Regolatori Buck LM2596[4]*

Inizialmente il regolatore che produceva una tensione di 12V alimentava entrambi i motori. Tuttavia, per garantire un'alimentazione più stabile e affidabile, si è deciso di aggiungere un quarto regolatore. Questa scelta ha permesso di dedicare un regolatore a ciascun motore, migliorando così le prestazioni complessive del sistema e riducendo il rischio di sovraccarichi o malfunzionamenti.

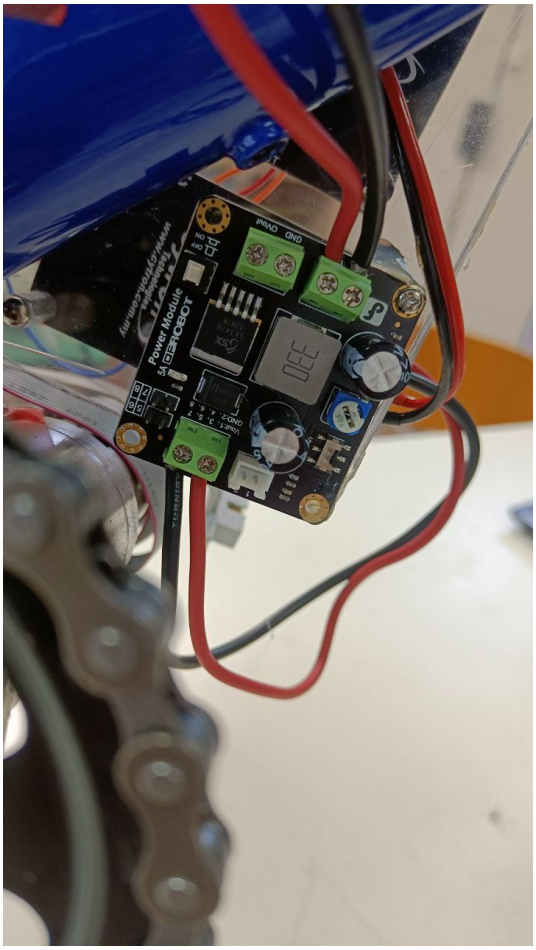

Figure 3.11: *Regolatore aggiunto successivamente*

3.9 Batteria

Le varie componenti vengono alimentate da una unica batteria. In particolare, nel corso del progetto sono state utilizzati due modelli di batteria a 4 celle, una Turnigy nanotech A-Spec G2[3] e una FullPower Silver Edition v2[2].

Figure 3.12: *Turnigy nanotech A-Spec G2*

Caratteristiche della Turnigy nanotech A-Spec G2:

1. Capacità: 2600 mAh;
2. Tensione: 4S1P / 4 celle / 14.8V
3. Scarico: 65C Costante / 130C Burst

Figure 3.13: *FullPower Silver Edition v2*

Caratteristiche della FullPower Silver Edition v2:

1. Capacità: 3300 mAH;
2. Tensione: 4S1P / 4 celle / 16.8V
3. Scarico: 35C Costante / 55C Burst

3.10 Schema dei collegamenti elettrici

È qui rappresentato lo schema dei collegamenti, dove sono indicate le connessioni di tutti i dispositivi introdotti nelle precedenti sezioni.

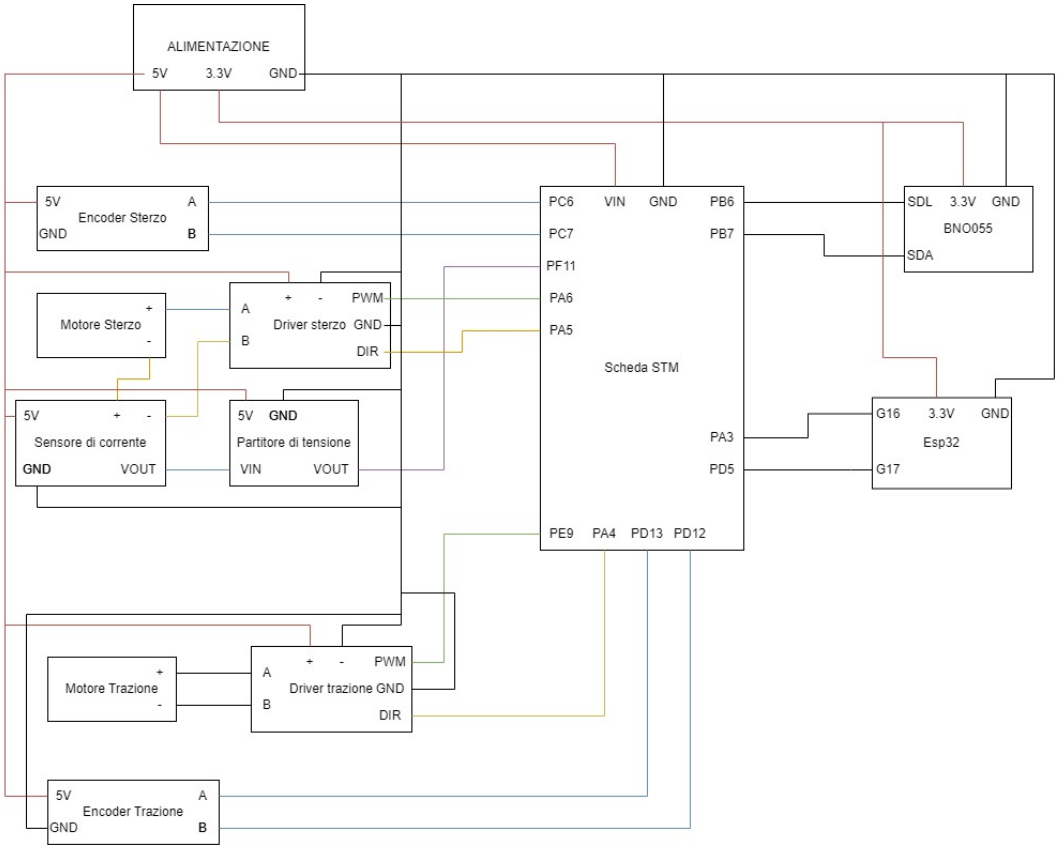

Figure 3.14: *Collegamenti del sistema*

Capitolo 4

Software

4.1 Diagramma di flusso

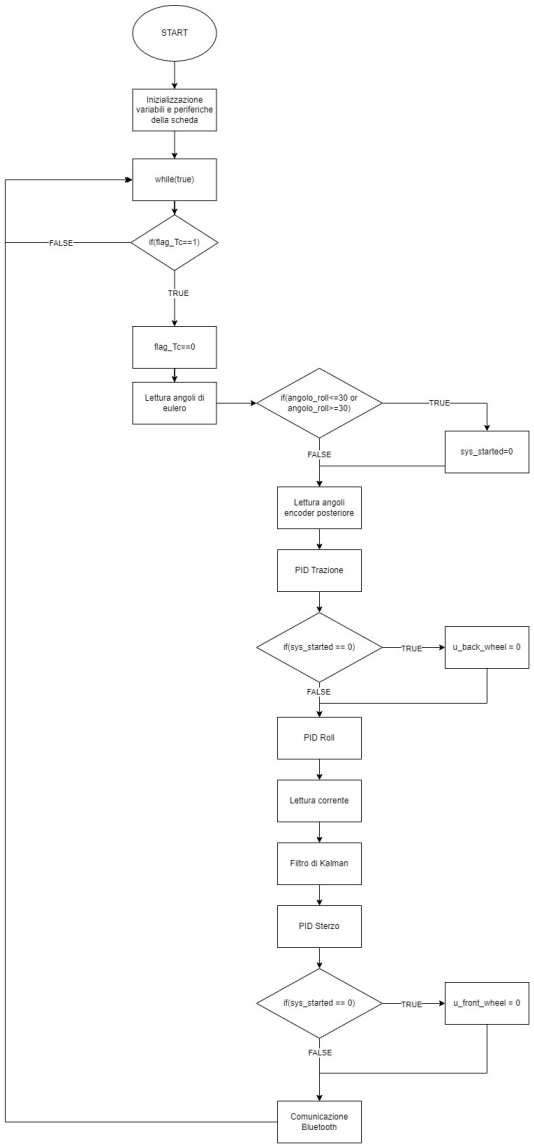

Figure 4.1: *Diagramma di flusso del codice implementato su microcontroller STM*

La figura precedente mostra il diagramma di flusso seguito dalla scheda STM32. Il processo inizia con l'inizializzazione di tutte le variabili utilizzate nel codice, inclusa la variabile `sys_started`, che può assumere tre valori:

- `sys_started = 0`: I PID di trazione e sterzo non alimentano i motori.
- `sys_started = 1`: Fase intermedia in cui vengono resettate alcune variabili necessarie per il controllo. Al termine del reset, `sys_started` assume il valore 2.
- `sys_started = 2`: I PID di trazione e sterzo sono attivi e funzionanti.

Questa variabile è cruciale per avviare la bicicletta tramite la pressione di un pulsante e per fermarla in caso di situazioni critiche. Quando viene premuto lo user button del microcontrollore, un timer attende 3 secondi prima di incrementare di 1 il valore di `sys_started`.

Dopo aver inizializzato le periferiche, nel ciclo `while` viene monitorata la variabile `flag_Tc`, che viene impostata a 1 ogni 0,01s da un timer gestito tramite interrupt. Ogni 0,01s vengono eseguite le seguenti operazioni:

1. Vengono letti gli angoli della bicicletta, verificando se l'angolo di rollio supera i 30 gradi. Se l'angolo eccede questo limite, la variabile `sys_started` viene impostata a 0, arrestando la bicicletta.
2. Viene letta la misura dell'encoder posteriore per calcolare la velocità della ruota posteriore e attivare il controllo della trazione Se `sys_started` è a 0 il PID non alimenta il motore;
3. Il PID genera una coppia desiderata per mantenere l'angolo di rollio vicino allo zero.
4. Viene letta la corrente e, dopo l'applicazione di un filtro di Kalman per pulire il segnale, viene attivato il PID per il controllo dello sterzo. Se `sys_started` è a 0 il PID non alimenta il motore;

Infine, tutti i dati raccolti vengono inviati tramite USART all'Esp32, che li trasmette via Bluetooth a un programma MATLAB su PC. Questo permette di visualizzare e graficare i dati in tempo reale.

4.2 STM32CubeIDE 1.13.2

4.2.1 Periferiche utilizzate

Timer

I timer sono delle periferiche del microcontrollore con i quali è possibile misurare il tempo. Non sono altro che contatori che contano con una frequenza uguale a quella

del processore del microcontrollore o un suo sottomultiplo. Il "Prescaler" rappresenta l'intero che divide la frequenza del processore. Iniziato il conteggio, il timer continua a contare fino ad un valore chiamato "Period", ed è possibile far generare un segnale di interrupt al suo raggiungimento.

La frequenza del segnale di interrupt è data quindi da

$$Frequenzadiaggioramento = \frac{FrequenzadiClock}{Prescaler \cdot Period}$$

dove Prescaler e Period devono essere interi maggiori o uguali a 1.

Sono stati usati 2 timer in modalità interrupt per conteggiare il tempo:

- il TIM2 è stato settato ad una frequenza di 100hz. Ogni 0.01 secondi la flag_Tc viene impostata a 1.

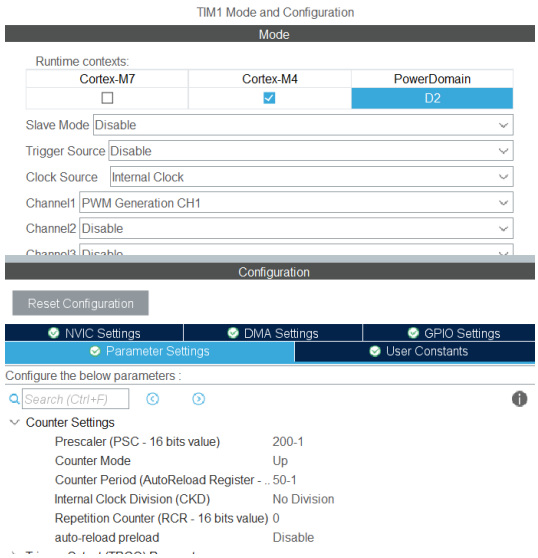

Figure 4.2: *Impostazioni del TIM2*

- il TIM6 è stato settato ad una frequenza di 0.33hz. Ogni 3 secondi vengono trasmessi i dati via bluetooth al computer se la comunicazione è attiva.

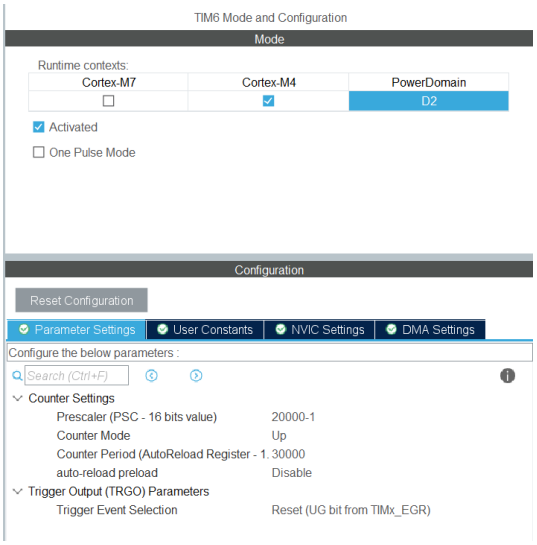

Figure 4.3: *Impostazioni del TIM6*

ADC

Gli ADC (Analog-to-Digital Converter) nei microcontrollori STM32 sono utilizzati per convertire segnali analogici in segnali digitali. Ogni segnale analogico in tensione può essere campionato dall'ADC e trasformato in un valore numerico che rappresenta quel segnale. Nelle STM32, gli ADC sono generalmente a 16 bit.

L'oversampling è una tecnica che permette di migliorare la risoluzione effettiva dell'ADC al di là della sua risoluzione fisica. Questo si ottiene campionando un segnale analogico più volte e poi facendo una media di quei campioni.

Il suo funzionamento è il seguente: L'ADC esegue un numero elevato di campionamenti dello stesso segnale. Questi campioni vengono sommati, e successivamente divisi per ridurre il rumore casuale.

Nella scheda è stato abilitato l'ADC1 con la funzione di oversampling, impostato su un rapporto di 1024.

Aumentando il ratio dell'oversampling aumenta anche la risoluzione dell'adc, e nel caso di un oversampling di 1024 i bit dedicati per la conversione sono 16 a cui vanno aggiunti i 10 dell'oversampling ($2^{10} = 1024$). La risoluzione risulta perciò $2^{26} = 67108864$.

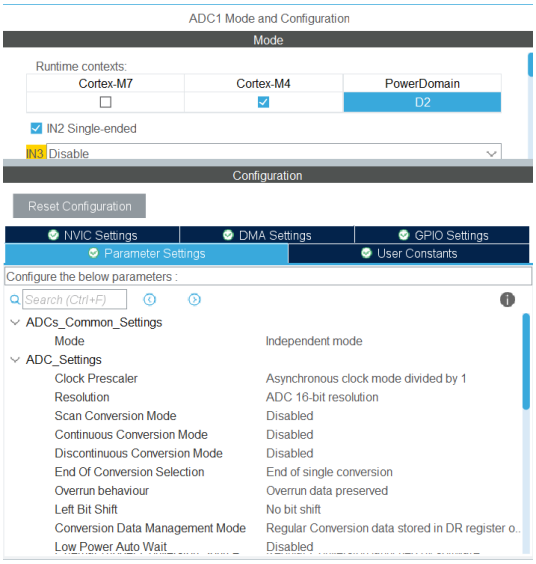

Figure 4.4: *Impostazioni dell' ADC1*

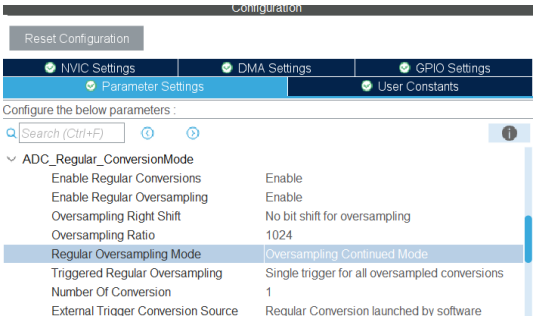

Figure 4.5: *Impostazioni dell'oversampling*

4.2.2 PWM

Un segnale pwm è un'onda quadra dove il segnale può assumere solamente 2 valori, che nella scheda STM sono 3.3V e 0V. Ci si avvale di questa tecnica poichè i microcontrollori lavorano con valori booleani, quindi possono generare una tensione alta o bassa, non intermedia. Tramite questa tecnica che consiste nell'alternare con una alta frequenza i 2 segnali alti e bassi, è possibile generare un segnale intermedio ad intensità voluta.

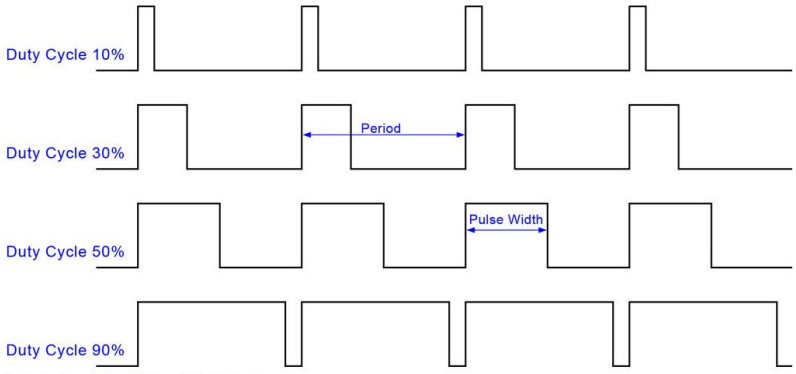

Figure 4.6: *Esempio di segnale pwm*

Se il tempo per il quale il segnale assume il valore alto viene chiamato t e se il periodo totale del segnale viene chiamato T, allora possiamo definire il duty cicle come $duty = \frac{t}{T}$, il quale rappresenta la percentuale di tempo per cui la tensione è alta rispetto al tempo totale.

Ripetendo ad alta frequenza questa onda quadra, dal momento che ogni sistema fisico è un filtro passa-basso, la potenza assorbita risultante sul carico esterno equivale alla potenza generata dalla tensione media sul carico.

Nel microcontrollore è possibile generare un segnale pwm tramite un timer, settandolo in modalita pwm.

I timer usati per generare i segnali pwm per lo sterzo e per la trazione sono:

- TIM1

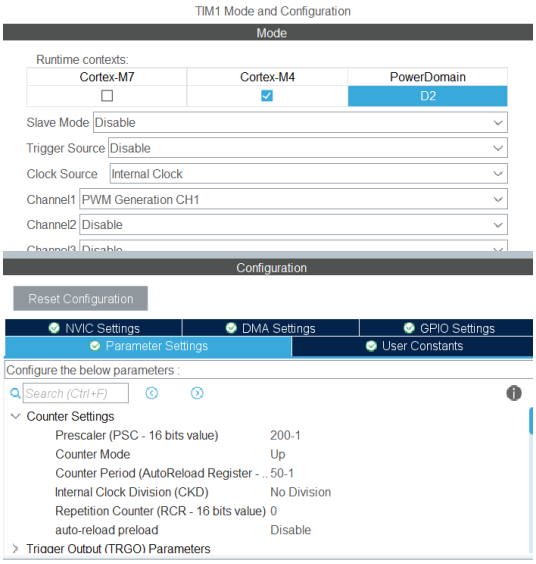

Figure 4.7: *Impostazioni del TIM1*

- TIM3

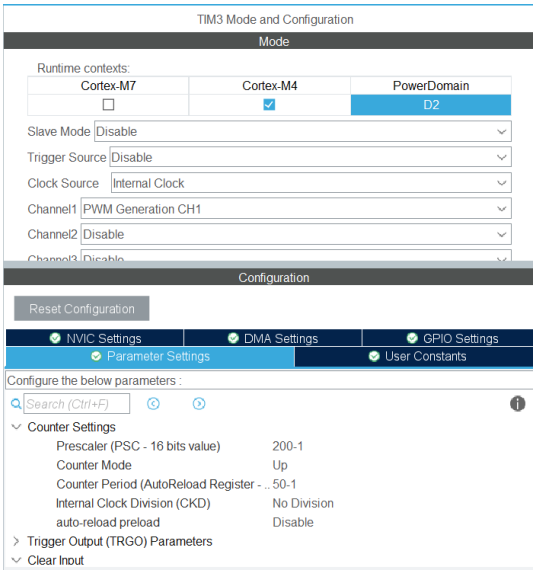

Figure 4.8: *Impostazioni del TIM3*

La frequenza di entrambi i timer è la massima consentita dai driver, ovvero 20000hz.

4.3 Libreria Bno055

La libreria bno055.c permette di interfacciarsi con il sensore IMU semplicemente. Infatti la libreria presenta numerose funzioni già codificate per acquisire e gestire le misurazioni del sensore. In questo contesto è stata utilizzata solo la funzione $bno055\_getVectorEuler()$, che restituisce i tre angoli di Eulero in una variabile di tipo $bno055\_vector\_t$. Da questa variabile vengono poi estratti esclusivamente gli angoli di rollio (roll) e imbardata (yaw).

4.4 Libreria Kalman

Il filtro di Kalman è un algoritmo ricorsivo che fornisce stime ottimali dello stato di un sistema utilizzando misure che possono essere imprecise e rumorose. Questo utilizza un modello matematico del sistema per prevedere lo stato futuro e corregge questa previsione utilizzando le misure effettive. Il processo si ripete iterativamente, migliorando la stima dello stato con ogni nuova misura. Nel nostro caso, il filtro verrà utilizzato per stimare la corrente reale a partire dai dati rumorosi forniti dal sensore.

Consideriamo un sistema dinamico che è rappresentato in spazio di stato:

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

Dove:

- $x(k)$ è il vettore di stato (all'istante k)
- $u(k)$ è il vettore degli ingressi (all'istante k)
- $w(k)$ è il rumore di processo
- A è la matrice di transizione di stato
- B è la matrice di controllo

Definiamo $z(k)$ come il vettore delle misure, e varrà quindi la relazione:

$$z(k) = Hx(k) + v(k)$$

Dove:

- H è la matrice di osservazione
- $v(k)$ è il rumore di misura

Da qui definiamo:

- $P_{k|k}$ la matrice di covarianza che misura l'incertezza della stima
- $P_{k|k-1}$ la matrice di covarianza che misura l'incertezza di una predizione
- R_k la matrice di covarianza che misura l'incertezza della misura
- Q la matrice di covarianza che misura il rumore di processo

Il filtro di Kalman procede in due fasi principali:

1. Fase di previsione:

   In questa fase cerchiamo di prevedere lo stato del sistema senza conoscere la misura.

   Conosciamo lo stato all'istante k-1 e deduciamo quale dovrebbe essere lo stato k secondo il modello dinamico. Dal momento che il vero x non lo conosciamo, utilizziamo la notazione $\widehat{x}$ essendo una stima.

   $$\widehat{x}_{k|k-1} = A\widehat{x}_{k-1|k-1} + Bu_k$$

   Troviamo poi $P_{k|k-1}$. È possibile dimostrare che vale la seguente uguaglianza:

   $P_{k|k-1} = AP_{k-1|k-1}A^T + Q$

2. Fase di aggiornamento:

Introduciamo la matrice K, ovvero il guadagno di Kalman, il quale determina il peso dato alle misurazioni nella stima dello stato.

Lo stato corrente si può stimare conoscendo la stima passata e la misura corrente:

$$\widehat{x}_{k|k} = \widehat{x}_{k|k-1} + K_k \left(z_k - H\widehat{x}_{k|k-1} \right) \tag{4.1}$$

Notiamo che nel caso in cui $H = I$ (cioè tutte le variabili di stato hanno una misura) l'equazione diventa:

$$\widehat{x}_{k|k} = \widehat{x}_{k|k-1} + K_k z_k - K_k \widehat{x}_{k|k-1} = (I - K_k)\, \widehat{x}_{k|k-1} + K_k z_k$$

Cioè la stima è una media pesata tra stato previsto e misura. Se $K = I$ allora lo stato previsto non ha peso e la stima corrisponde alle misure, mentre se $K = 0$ le misure non vengono considerate e la stima è uguale alla previsione.

Prima di applicare L'Eq 4.1 dobbiamo trovare K_k che è data da:

$$K_k = P_{k|k-1} H^T (H P_{k|k-1} H^T + R_k)^{-1}$$

E infine troviamo la matrice della covarianza:

$P_{k|k} = (I - K_k H)\, P_{k|k-1}$

Si noti che questa equazione è facile da ricordare ma numericamente instabile: in seguito a arrotondamenti possono generarsi grandi errori di calcolo, per questo è preferibile usare:

$$P_{k|k} = P_{k|k-1} - P_{k|k-1} H^T K_k^T - K_k H P_{k|k-1} + K_k \left(H P_{k|k-1} H^T + R_k \right) K_k^T$$

Queste sono le 2 fasi del filtro di Kalman, conclusa la seconda si ritorna alla prima e poi alla seconda e cosi via.

Le matrici usate per il filtro sono quelle che sono state scelte precedentemente nel progetto [8]:

$$x = \begin{bmatrix} V \\ I \end{bmatrix} \qquad u = [2.4812] \qquad z = \begin{bmatrix} misura\ V \\ misura\ I \end{bmatrix}$$

$$A = \begin{bmatrix} 0.0010 & 0.2172 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad Q = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.4 \end{bmatrix} \cdot 10^{-4}$$

$$P = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix} \qquad R = \begin{bmatrix} 0.0013 & 0 \\ 0 & 0.0279 \end{bmatrix}$$

Nel nostro contesto, la "misura V" corrisponde alla lettura effettuata dall'ADC della scheda STM, mentre la "misura I" rappresenta il valore della corrente ottenuto indirettamente dall'equazione

$$I = mV + q$$

dove m e q sono costanti del sensore.

4.5 Libreria DC_Motor

La seguente libreria serve a modulare la tensione applicata al motore e il verso di rotazione, tramite segnale PWM e porte GPIO. Le funzioni definite sono:

- `Voltage2Duty`, che converte il valore di tensione prodotto da un PID in un duty cicle.
- `Ref2Direction`, che restituisce:
  - 0 se il parametro passato alla funzione è maggiore o uguale a 0
  - 1 altrimenti

  Questa funzione viene usata per determinare il verso della rotazione del motore a partire dal valore di tensione prodotto dai PID.
- `set_PWM_and_dir_back_wheel`, la quale a partire dal duty cicle e dalla direzione, crea un segnale pwm e alimenta il driver della ruota posteriore
- `set_PWM_and_dir_front_wheel`. funzione analoga alla precedente, ma dedicata alla ruota anteriore.

```
1 #include "DC_motor.h"
2 #include "main.h"
3 #include "stdint.h"
4
5 float Voltage2Duty(float u){
6     if(u <= 0){
```

```
7           u = -u;
8       }
9       float duty = 100 * u/V_MAX;
10
11      if (duty > 100){
12          duty = 100;
13      }else if(duty < 0){
14          duty = 0;
15      }
16      return duty;
17  }
18
19  uint8_t Ref2Direction(float y_ref){
20      uint8_t direction;
21      if(y_ref >= 0){
22          direction = 0;
23      } else {
24          direction = 1;
25      }
26      return direction;
27  }
28
29  void set_PWM_and_dir_back_wheel(float duty, uint8_t direction){
30      TIM1 ->CCR1 = (duty/100.0)*TIM1->ARR;
31
32
33
34      if(direction == 0){
35          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
36      }else if(direction == 1){
37          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
38      }
39  }
40
41  void set_PWM_and_dir_front_wheel (float duty, uint8_t dir){
42      TIM3 -> CCR1 = (duty/100)*TIM3->ARR;
43
44      if (dir == 0){
45          HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
46      }else if (dir == 1){
47          HAL_GPIO_WritePin (GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
48      }
49  }
```

4.6 Libreria PID

Per implementare i PID sul microcontrollore è stata usata la libreria "PID.c", nella quale vengono definite le funzioni:

- `init_PID`, per inizializzare il PID con il tempo di campionamento usato e definire il valore di output massimo e minimo.
- `tune_PID`, per assegnare le costanti K_P, K_I, K_D.
- `PID_Controller`, che implementa l'algoritmo per ottenere l'uscita del PID dando come input l'uscita del sistema e il riferimento.

```
1
2  /*
3   * PID.c
4   *
5   *  Created on: Nov 20, 2023
6   *      Author: andre
7   */
8
9  #include <PID.h>
10
11 void init_PID (PID* p, float Tc, float u_max, float u_min){
12
13     p->Tc = Tc;
14     p->u_max = u_max;
15     p->u_min = u_min;
16     p->e_old=0;
17     p->Iterm=0;
18
19 }
20
21
22 void tune_PID (PID* p, float Kp, float Ki, float Kd){
23
24     p->Kp = Kp;
25     p->Kd = Kd;
26     p->Ki = Ki;
27 }
28
29 float PID_controller (PID* p, float y, float r){
30
31     float u;
32     float newIterm;
33     float e = r-y;
34     float Pterm = p-> Kp * e;
35
36     newIterm = p->Iterm + (p->Ki)* p->Tc * p->e_old;
37     p->e_old = e;
38     u = Pterm + newIterm;
```

Code/PID.c

Capitolo 5

Catena di controllo

5.1 PID Trazione

Il controllo PID per la trazione sviluppato precedentemente non ha fornito i risultati desiderati, per cui è stato rielaborato partendo da zero.

Il nuovo approccio è suddiviso nelle seguenti fasi:

1. Si acquisisce la velocità angolare della corona anteriore (che aziona il motore) tramite le misurazioni dell'encoder.
2. La velocità angolare viene filtrata utilizzando un filtro a media mobile, per attenuare il rumore presente nei dati.
3. Si rileva la velocità angolare della ruota posteriore e da questa si ricava la velocità tangenziale al punto di contatto con il terreno.
4. Si determina la velocità desiderata attraverso la funzione `getSpeed()`, che genera un riferimento a rampa fino al valore della variabile `speed`, mantenendolo costante una volta raggiunto.
5. Il PID calcola la tensione da applicare al motore per raggiungere la velocità desiderata.
6. Si calcola il duty cycle e la direzione di rotazione corrispondente, che vengono successivamente inviati al driver per il controllo del motore.

```
//#####################################
//##          RUOTA DIETRO           ##
//#####################################
//*******************************
//Ottengo velocit  ruota dietro
counts = (double) TIM4->CNT - (TIM4->ARR) / 2;
TIM4->CNT = (TIM4->ARR) / 2;
//velocit  angolare
//encoder ha risoluzione cpr 500, non 66
delta_angle_degree = (counts * 360)
        / (ppr * gear_ratio * encoder_resolution); //del motore(
            davanti) 18 denti dietro (raggio)3.8cm ,28 denti davanti (
            raggio) 5.7cm
```

```
12 speed_degsec = -1 * delta_angle_degree / dt;
13
14 angle_degree += delta_angle_degree;
15 //filtro media mobile
16 speed_degsec_filtrata = filtro_media_mobile(velocit vecchia,
17         velocit nuova, speed_degsec, 30);
18 //rapporto ruota posteriore
19 speed_degsec_back = speed_degsec_filtrata
20         * raggio_corona_anteriore
21         / raggio_corona_posteriore; //rapporto velcoita angolare tra
               ruota dietro e avanti???
22
23 //velocit  ruota dietro
24 speed_metsec = speed_degsec_back / 180 * 3.14 * radius;
25
26 //angolo dietro
27 angle_back_wheel += speed_degsec_back * dt;
28 //******************************
29
30 //******************************+
31 //PID ruota dietro
32 desired_speed_metsec = getSpeed(desired_speed_metsec); //funzione che
       crea un riferimento a rampa e poi costante per la velocit  della
       ruota dietro
33 desired_speed_rpm = DegreeSec2RPM(desired_speed_metsec / radius); //
       inutile per ora
34 u_back_wheel = PID_controller(&pid_speed, speed_metsec,
35         desired_speed_metsec);
36 //******************************
```

5.2 PID Roll

Il PID per l'angolo di roll ha il ruolo di regolare la coppia da applicare allo sterzo della bicicletta per mantenere l'angolo di rollio a 0.

5.2.1 Funzionamento

Il funzionamento è il seguente:

1. L'angolo di rollio viene misurato e corretto aggiungendo un offset, poiché non è perfettamente parallelo al piano dove si muove la bicicletta.

```
1
2 //#####################################
3 //##             BNO055              ##
4 //#####################################
5 //******************************
6
```

```
7 eul = bno055_getVectorEuler();
8
9 roll = -eul.y + 1.6; //ottengo angolo di eulero (il sensore
      leggermente inclinato rispetto al piano in cui giace la
      bicicletta)
```

2. Viene calcolata la coppia desiderata tramite il PID.

```
1 //#####################################
2 //##            PID ROLL             ##
3 //#####################################
4
5 //*****************************
6
7 desired_roll = 0; //l'angolo di equilibrio sono 2 gradi
8 desired_torque = PID_controller(&pid_roll, roll, desired_roll);
9 //*****************************
```

5.2.2 Esempio di funzionamento

Il PID dell'angolo di roll migliore ha i seguenti parametri:

- $K_P = 0.000150$
- $K_I = 0$
- $K_D = 0.000600$

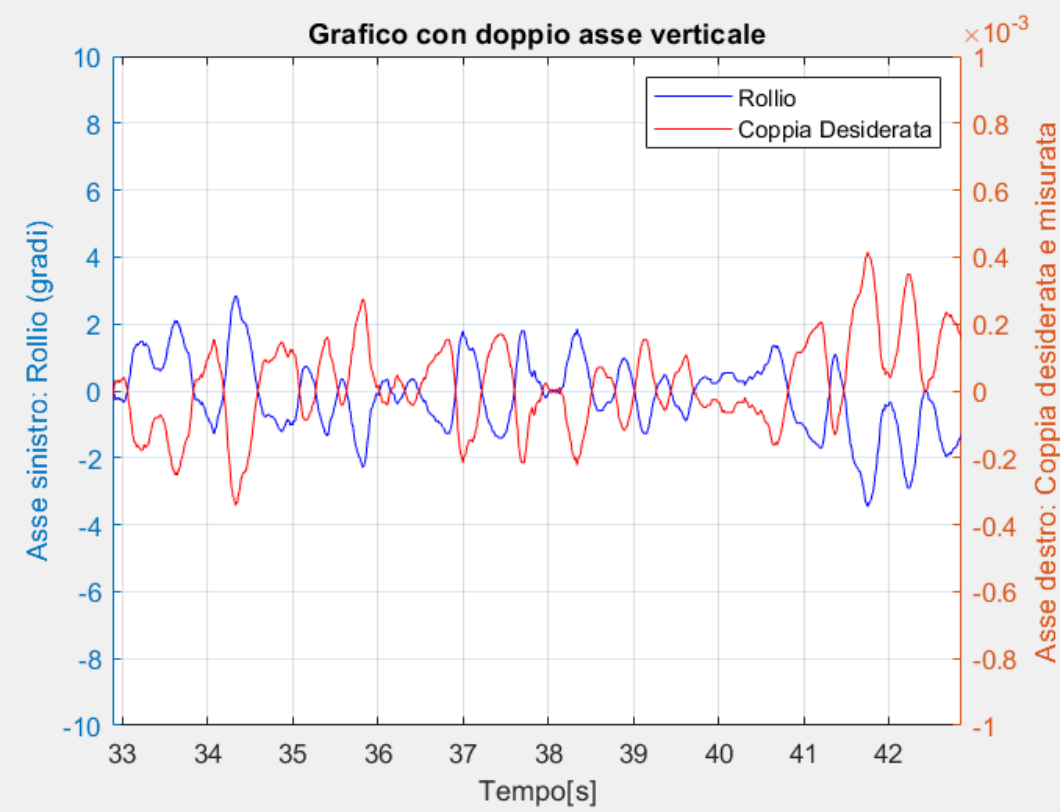

Figure 5.1: *Grafico che mostra l'angolo di rollio e la coppia desiderata prodotta dal PID*

5.3 Filtro passa basso

Il filtro passa basso nel sistema di controllo rappresentato nella Figura 2.6 viene introdotto per simulare il comportamento umano durante la guida di una bicicletta. In particolare, gli esseri umani non eseguono mai movimenti del manubrio troppo rapidi, poiché ciò comprometterebbe la stabilità del veicolo.

Per implementare il filtro passa basso $\frac{1}{s+1}$ sulla scheda STM, è necessario sviluppare un algoritmo che, ad ogni iterazione (ogni 0,01 secondi), riproduca il comportamento del filtro a tempo continuo.

Utilizzando MATLAB, è possibile discretizzare il filtro ottenendo la sua rappresentazione nel dominio zeta, che può essere direttamente utilizzata per il controllo digitale.

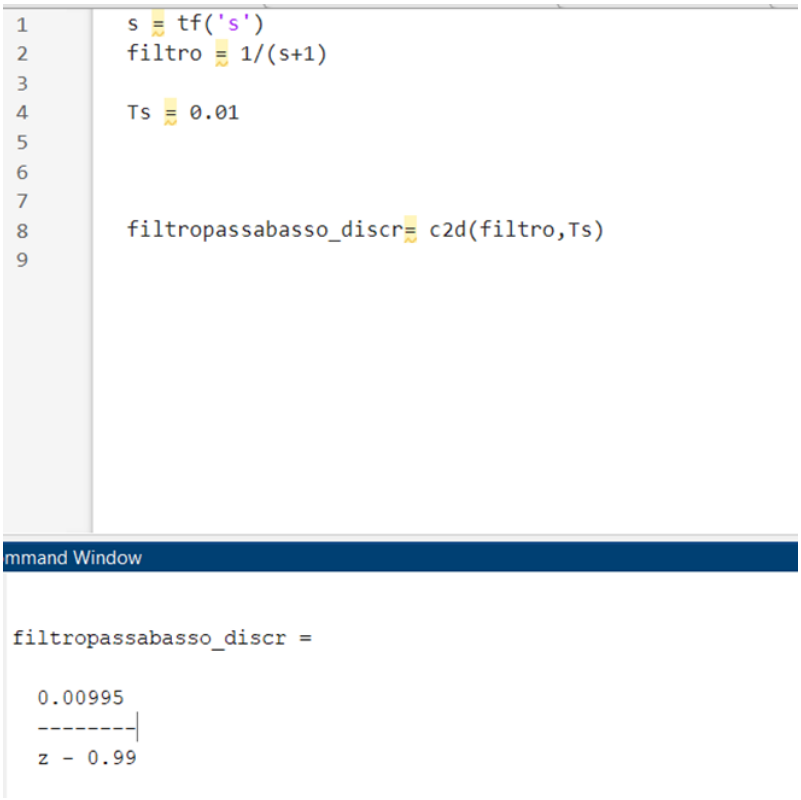

Figure 5.2: *Funzione Matlab con il quale viene discretizzato il filtro*

Se chiamiamo $y(k)$ l'uscita del filtro e $u(k)$ l'ingresso del filtro, rimodelliamo la funzione di matlab in modo da ottenere il filtro nel dominio del tempo discreto.

$$F(z) = \frac{0.00995}{z - 0.99} = \frac{Y(z)}{U(z)}$$

$$0.00995U(z) = Y(z)(z - 0.99) = zY(z) - 0.99Y(z)$$

$$0.00995\frac{U(z)}{z} = Y(z) - 0.99\frac{Y(z)}{z}$$

Che nel dominio del tempo diventa:

$$y(k) = 0.00995u(k-1) + 0.99y(k-1)$$

Trovata l'equazione, viene implementata nel while:

```
//#######################################
//##         LOW-PASS FILTER           ##
//#######################################
//*****************************
//filtro passa basso 1/s+1 discretizzato con Matlab

desired_filtered_torque = 0.99 * old_desired_filtered_torque
        + 0.00995 * old_desired_torque;
old_desired_torque = desired_torque;
```

```
10 old_desired_filtered_torque = desired_filtered_torque;
```

Dopo numerosi tentativi, si è constatato che l'uso del filtro passa-basso comprometteva l'efficacia del controllo, impedendo alla bicicletta di mantenere l'equilibrio. È stato testato anche un filtro passa basso $\frac{1}{s+10}$, meno "lento", ma i risultati migliori sono stati ottenuti eliminando completamente il filtro.

Sotto viene rappresentato l'effetto del filtro passa-basso $\frac{1}{s+1}$, che induce un ritardo non trascurabile per il mantenimento dell'equilibrio della e-Bike.

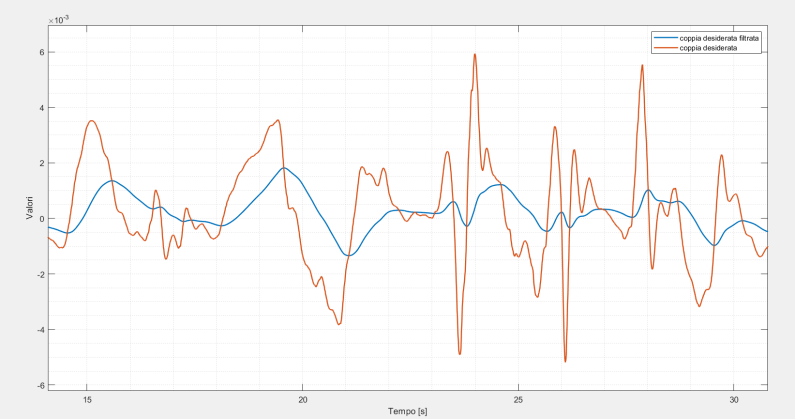

Figure 5.3: *Grafico che mostra l'effetto del filtro passa basso*

5.4 PID Sterzo

Il PID dello sterzo ha il ruolo di regolare la tensione per generare la coppia calcolata con il PID di roll.

Per misurare la coppia è stato utilizzato il sensore di corrente, in quanto la coppia vale $C = KI$, dove la costante K vale $0.0234\frac{Nm}{A}$. Essendo la misura della corrente rumorosa, è stato utilizzato un filtro di Kalman.

5.4.1 Funzionamento

Il funzionamento è suddiviso in queste fasi:

1. L'adc misura il valore del partitore di tensione, viene aggiunto l'offset (D) e calcolato il valore prodotto dal sensore di corrente.
2. Si applica il filtro di Kalman al segnale.
3. Viene calcolata la coppia.
4. Si applica un controllo in modo tale che l'angolo dello sterzo non superi i 90 gradi in entrambe le direzioni.
5. viene calcolata la tensione da applicare al motore con il PID.

6. viene calcolato il duty-cicle corrispondente e il verso dell'alimentazione, e vengono impartiti al driver.

```
//#######################################
//##            STERZO                 ##
//#######################################
//*****************************
//Calcolo valore corrente
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, timeout);
CountValue = HAL_ADC_GetValue(&hadc1);
volt = ((float) CountValue) * Vref / (resolution);
HAL_ADC_Stop(&hadc1);
VoltSens = (volt + D) * 1.5059;
corrente_non_filtrata = voltToAmpere(VoltSens, a, b);


//Filtro di Kalman per corrente
//setta i valori di input e di misura per il filtro di kalman
z_data[0] = VoltSens; //misura del voltaggio del sensore di corrente
z_data[1] = corrente_non_filtrata; //per non usare la misura I
    aggiuntiva, volendo si pu  usare la formula V=IR del motore,
    adesso vedo come metterla
kalman_predict(&kf, &u);
kalman_update(&kf, &z);
filtered_current_kalman = x_data[1];

//calcolo coppia
torque = filtered_current_kalman * K;

//controllo angolo limite manubrio
//quando la u    negativa, l'angolo diminuisce
 if (angle_steer <= -90) {
 if (u_front_wheel < 0)
 u_front_wheel = 0;
 }
 if (angle_steer >= 90) //se
 {
 if (u_front_wheel > 0)
 u_front_wheel = 0;

 }

if (sys_started <= 1) {
    desired_speed_metsec = 0;
    u_back_wheel = 0;
    u_front_wheel = 0;

}
u_front_wheel = PID_controller(&pid_steering_torque, torque,
                    desired_filtered_torque);
```

```
47 duty_front_wheel = Voltage2Duty(u_front_wheel);
48 dir_front_wheel = Ref2Direction(u_front_wheel);
49 set_PWM_and_dir_front_wheel(duty_front_wheel, dir_front_wheel);
```

5.4.2 Oversampling ADC

Il segnale fornito dal sensore di corrente presentava un livello significativo di rumore, motivo per cui inizialmente è stato applicato un filtro di Kalman. Questo filtro ha permesso di trovare un compromesso accettabile tra la reattività del segnale e la riduzione del rumore, soprattutto nei test statici, in cui la e-Bike veniva lasciata sterzare a vuoto. Tuttavia, durante i test dinamici effettuati sul tapis roulant, si è osservato che la coppia applicata allo sterzo non veniva generata abbastanza velocemente, oppure risultava in movimenti della ruota troppo bruschi.

Dal momento che un controllo efficace richiede una misura accurata e affidabile, si è deciso di migliorare ulteriormente il segnale applicando la tecnica dell'oversampling all'ADC. Questa tecnica ha portato a risultati nettamente migliori.

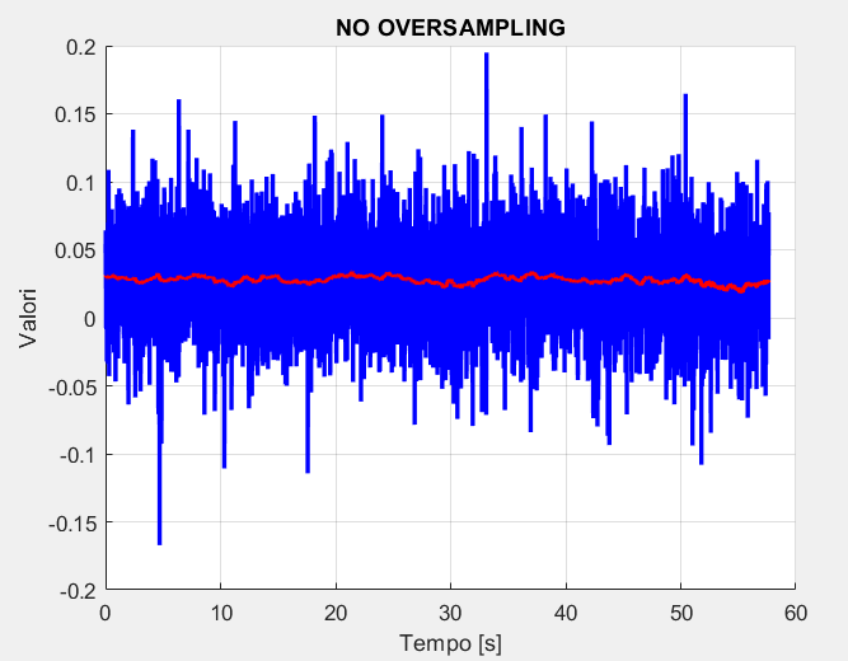

Figure 5.4: *ADC senza oversampling. La linea blu rappresentata la corrente misurata, la linea rossa rappresenta la corrente filtrata dal filtro di Kalman*

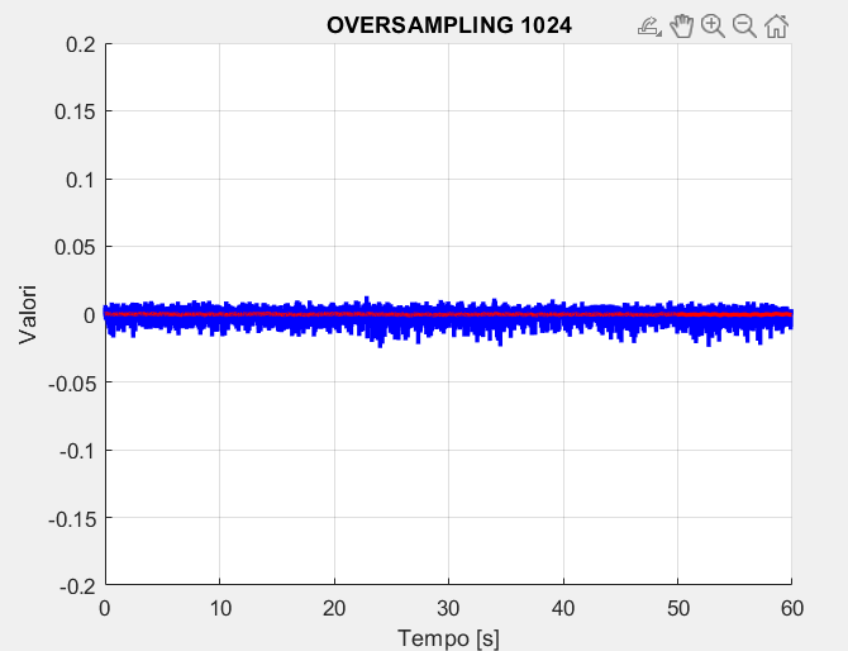

Figure 5.5: *ADC con oversampling abilitato. La linea blu rappresentata la corrente misurato dall'ADC, la linea rossa rappresenta la corrente filtrata dal filtro di Kalman*

Come mostrato nelle figure, l'oversampling ha migliorato notevolmente la qualità del segnale. In combinazione con il filtro di Kalman, il segnale risulta molto più preciso e accurato, consentendo al sistema di controllo di gestire coppie anche molto piccole, che in precedenza non potevano essere applicate con sufficiente precisione.

	Senza oversampling	**Con oversampling**
Range ADC	0.3622	0.0382
Range Filtro di Kalman	0.0154	0.0015

Table 5.1: *Tabella che mostra i range dell'ADC senza e con oversampling. L'unità di misura è Ampere (corrente)*

5.4.3 Correzione riferimento di tensione

Nel corso dei test si è verificato che a volte la misura della corrente non fosse a 0A nonostante essa fosse effettivamente a 0A.

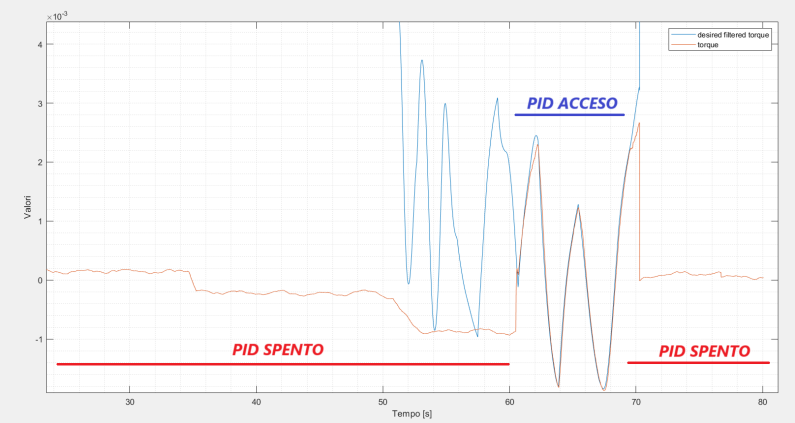

Figure 5.6: *Grafico che mostra il problema delle misurazioni della coppia. Le linee rosse e blu indicano i tratti di tempo in cui il PID è spento o acceso*

.

Dalla figura vediamo come la coppia misurata non rimanesse a 0Nm nonostante il PID fosse spento e la ruota anteriore fosse ferma. Nel momento in cui il PID era acceso, il riferimento era seguito bene, benchè girasse più in un verso che nell'altro.

In un primo momento era stato notato che quando la scheda veniva alimentata dal computer, la misura non presentava un offset, a differenza dell'alimentazione con batteria.

Oltretutto, è stata osservata anche un'altra peculiarità: l'offset variabile della tensione misurata era presente quando la scheda STM32 è alimentata solo dalla distribuzione di energia dal sistema ed era connessa al computer con il cavo USB. Di seguito sono riportate le immagini che mostrano la tensione letta, con PID disattivato e motore non alimentato in entrambe le casistiche

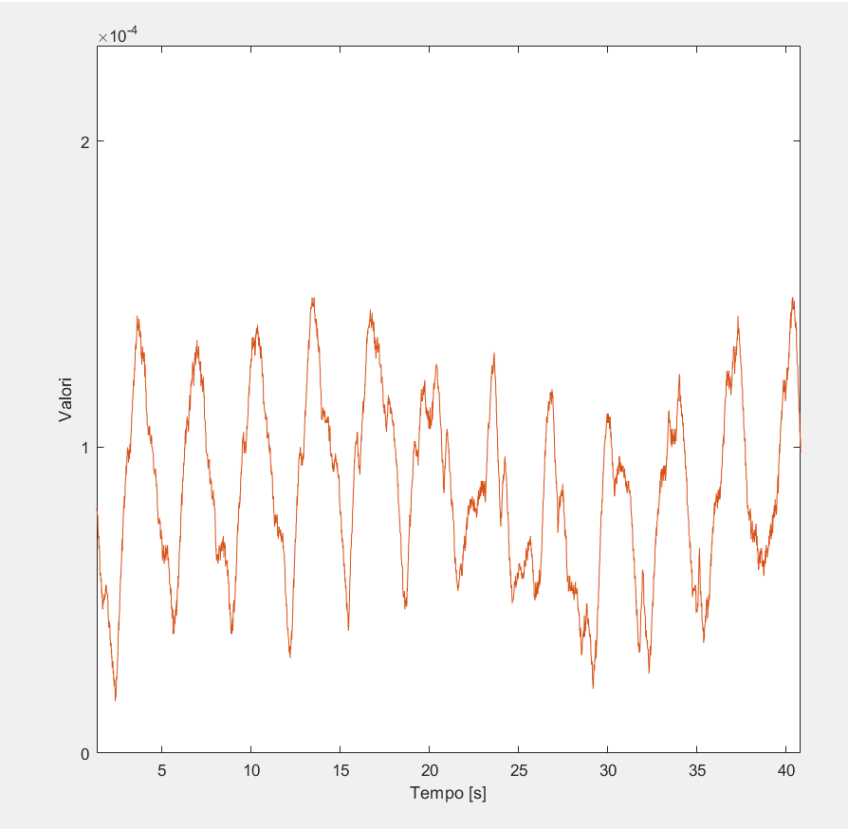

Figure 5.7: *Misura della coppia con scheda non collegata al computer tramite cavo USB*

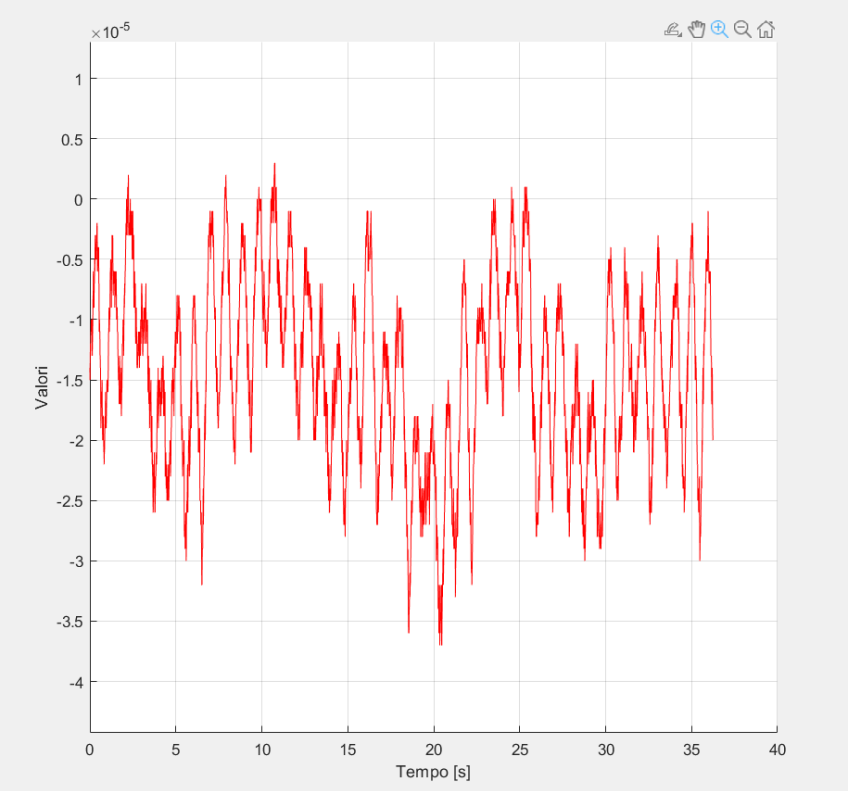

Figure 5.8: *Misura della coppia con scheda collegata al computer tramite cavo USB*

Nella prima figura c'è un errore di circa $6 \cdot 10^{-5}$, mentre nella seconda un errore di $1.5 \cdot 10^{-5}$.

Nonostante l'errore sembri piccolo, la coppia necessaria per far ruotare la ruota della bicicletta è dell'ordine di 10^{-3} e questo può influire sul controllo, come è stato verificato.

Con ulteriori test si è scoperto che il problema era dovuto al ground della scheda STM, che nonostante fosse collegato al ground del sistema, non era comunque a 0V, ma fluttuava tra 0V e 0.40V. Una volta sostituito il cavo di ground della scheda con un cavo di sezione maggiore, quindi con una resistenza minore, la tensione di riferimento risulta più stabile. Questa soluzione ha dunque mitigato notevolmente la problematica riscontrata, rendendo sufficientemente affidabile la misurazione acquisita.

5.4.4 Esempio di funzionamento

Inizialmente il PID per lo sterzo era stato settato a vuoto, lasciando la ruota anteriora sterzare in aria, con i seguenti parametri:

- $K_P = 30000$
- $K_I = 100000$

- $K_D = 0$

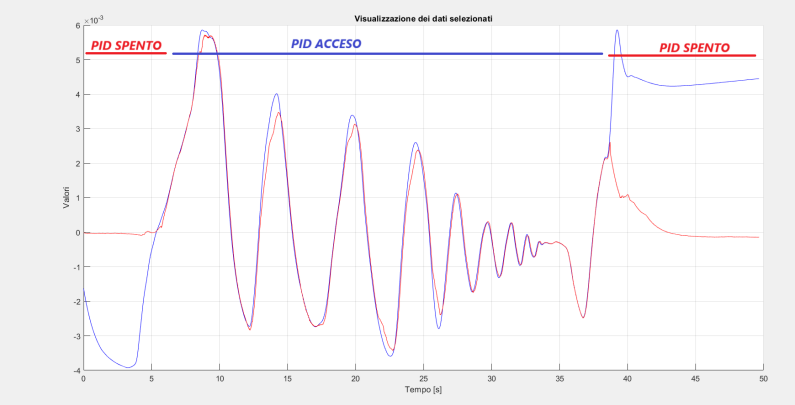

Figure 5.9: *Grafico dell'andamento della coppia misurata (rossa) e della coppia desiderata (blu). Qui la ruota anteriore della e-Bike viene sollevata*

In seguito si è notato che nella guida autonoma della e-Bike, la coppia misurata non raggiungeva la coppia desiderata abbastanza velocemente, come si vede dal grafico 5.10

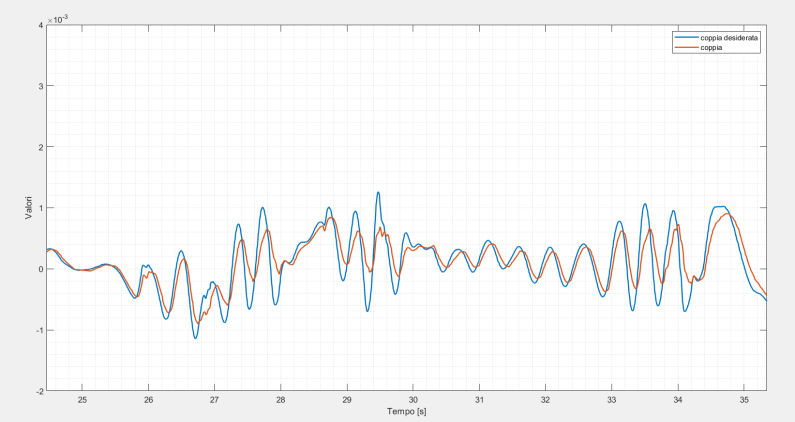

Figure 5.10: *Primo PID applicato durante il tentativo dell'e-Bike di mantenere l'equilibrio*

Per questo motivo si è alzato di molto il guadagno proporzionale e diminuito il guadagno integrale, rendendo più reattivo il controllo. I nuovi parametri sono:

- $K_P = 105000$
- $K_I = 30000$
- $K_D = 0$

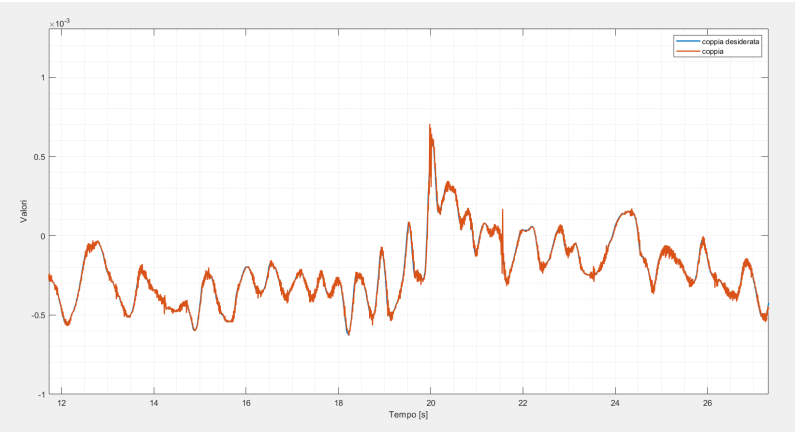

Figure 5.11: *Secondo PID applicato durante il tentativo dell'e-Bike di mantenere l'equilibrio.*

Capitolo 6

Comunicazione

6.1 Comunicazione seriali

I protocolli seriali, come I2C, SPI e UART, sono strumenti fondamentali per la comunicazione dati a breve distanza e via cavo all'interno dei sistemi embedded. Queste interfacce sono comunemente impiegate per collegare tra loro vari componenti elettronici, come microcontrollori (MCU) e sensori, garantendo l'interazione tra diversi dispositivi all'interno dello stesso sistema.

I microcontrollori (MCU) integrano una vasta gamma di periferiche di comunicazione basate su interfacce digitali standardizzate. Questi canali permettono lo scambio di dati tra più circuiti integrati (IC), i quali possono essere montati sullo stesso circuito stampato (PCB), favorendo un'efficiente comunicazione e sincronizzazione tra i vari moduli del sistema.

La trasmissione dei dati tra dispositivi digitali può avvenire tramite due tecniche fondamentali: seriale e parallela. Nella trasmissione seriale, i dati vengono inviati un bit alla volta attraverso un singolo canale, mentre nella trasmissione parallela più bit vengono trasmessi simultaneamente su più linee, consentendo un trasferimento più rapido, ma richiedendo una maggiore complessità.

Nei protocolli di comunicazione sincrona, come SPI, I2C e USART, il trasferimento di dati avviene seguendo lo stesso segnale di clock sia per il mittente che per il destinatario. Per garantire la sincronizzazione, è necessaria una linea dedicata per il clock, che coordina l'invio e la ricezione dei dati. Nei protocolli asincroni, come UART, invece, non vi è un clock condiviso tra le due parti. In questo caso, la velocità di trasmissione deve essere prestabilita da entrambi i dispositivi per garantire un trasferimento corretto.

La comunicazione half-duplex permette ai dispositivi di scambiarsi dati in entrambe le direzioni, ma solo un'operazione alla volta: un dispositivo può inviare o ricevere, ma non entrambe le cose simultaneamente. Questo tipo di comunicazione, utilizzato ad esempio da I2C, richiede un solo canale. Al contrario, la modalità full-duplex consente la trasmissione simultanea in entrambe le direzioni, come avviene con i protocolli SPI e UART, ma richiede due canali per garantire la trasmissione continua da e verso entrambi i dispositivi.

6.1.1 UART (Universal Asynchronous Receiver-Transmitter)

L'Universal Asynchronous Receiver-Transmitter (UART) è ampiamente riconosciuto come uno dei protocolli di comunicazione tra dispositivi più comuni. Non è solo un protocollo, ma un circuito fisico vero e proprio, il cui obiettivo principale è quello di convertire i dati paralleli in dati seriali e viceversa, facilitando così la trasmissione e la ricezione dei dati stessi. Nel sistema di comunicazione UART, due UART comunicano direttamente l'una con l'altra. Il processo inizia con l'UART trasmittente che converte i dati paralleli ricevuti da un dispositivo di controllo in formato seriale. Questi dati seriali vengono poi inviati all'UART ricevente, la quale riconverte i dati seriali in formato parallelo per il dispositivo destinatario. Questo metodo di conversione bidirezionale assicura un trasferimento dati efficace tra dispositivi con differenti formati di dati.

I protocolli UART operano con una trasmissione seriale, inviando dati un bit alla volta, e sono di tipo asincrono, poiché non utilizzano un segnale di clock per la sincronizzazione tra mittente e destinatario. Supportano diverse modalità di comunicazione: full-duplex, che permette lo scambio simultaneo di dati in entrambe le direzioni; half-duplex, che consente la trasmissione bidirezionale non contemporanea; e simplex, in cui la comunicazione dei dati è unidirezionale.

UART utilizza due cavi per la trasmissione dei dati tra dispositivi. I dati vengono inviati dal pin di trasmissione (Tx) dell'UART del dispositivo mittente al pin di ricezione (Rx) dell'UART del dispositivo destinatario.

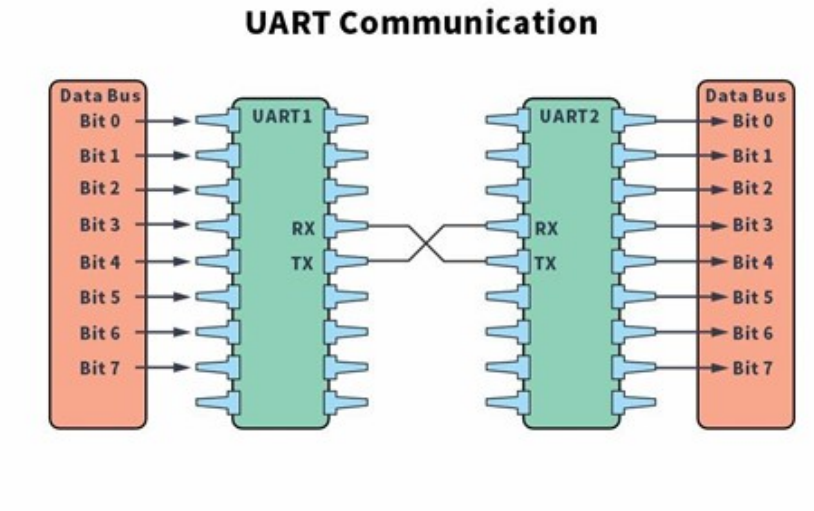

Figure 6.1: *Comunicazione UART*

UART non fa uso di un segnale di clock per sincronizzare il trasmettitore e il ricevitore. Il trasmettitore genera un flusso di bit basato sul proprio segnale di clock, mentre il ricevitore impiega il proprio segnale di clock interno per campionare i dati in ingresso. La sincronizzazione viene garantita configurando entrambi i dispositivi

con le stesse impostazioni, in particolare con il medesimo baud rate. La tolleranza per la differenza nella velocità di trasmissione può arrivare fino al 10% prima che la temporizzazione dei bit diventi eccessivamente disallineata, causando discrepanze nella gestione dei dati. I dati trasmessi attraverso UART sono strutturati in pacchetti. Ogni pacchetto include un bit di inizio, seguito da 5 a 9 bit di dati (a seconda della configurazione dell'UART, con un massimo di 8 bit se viene utilizzato il bit di parità), un bit di parità facoltativo, e 1 o 2 bit di stop:

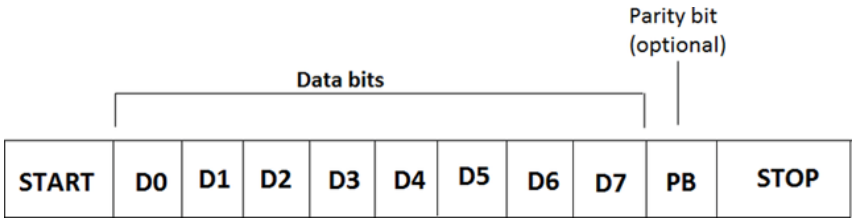

Figure 6.2: *Distribuzione dei bit*

Le impostazioni dell'UART includono le seguenti configurazioni (le impostazioni comuni sono indicate tra parentesi): [6]

- **Baud Rate (115200 bit/s)**: Rappresenta la velocità di trasmissione dei bit e può assumere vari valori come 9600, 14400, 19200, 38400, 57600, 115200, 128000 e 256000 bit al secondo;
- **Bit di parità (nessun bit di parità)**: Descrive la parità o la disparità dei dati trasmessi, permettendo all'UART ricevente di rilevare eventuali errori durante la trasmissione;
- **Dimensione dei bit di dati (8 bit)**: Indica il numero di bit nel frame dati trasmesso o ricevuto;
- **Dimensione bit di stop (1 bit)**: Specifica il numero di bit di stop utilizzati per segnalare la fine del pacchetto di dati;
- **Controllo di flusso (nessun controllo di flusso)**: Ad esempio, il controllo di flusso RTS/CTS fa parte dello standard RS232 e utilizza due pin aggiuntivi, RTS (Request to Send) e CTS (Clear to Send), per gestire il flusso di dati tra mittente e destinatario;

Queste impostazioni definiscono il modo in cui l'UART comunica e interagisce con altri dispositivi, garantendo una trasmissione affidabile e conforme agli standard di comunicazione seriale.

6.1.2 Codice Matlab per la Visualizzazione dei dati con comunicazione usart

Attraverso MATLAB è possibile gestire la comunicazione USART, permettendo la lettura, la modifica e la visualizzazione grafica dei dati acquisiti.

Di seguito è riportato il codice MATLAB utilizzato:

1. Vengono inizializzate le variabili e le linee animate usate per la rappresentazione grafica

```
%clc; clear; close all;

% Inizializzazione variabili
T = 60*10;                          % Durata totale dell'acquisizione
    in secondi
deltaTs = 0.01;                 % Intervallo di campionamento (10 ms
    )
N_campioni = T / deltaTs;       % Numero totale di campioni
dati = zeros(N_campioni, 2); % Preallocazione matrice per dati di
    interesse
figure;

% Inizializzazione grafica


misurato1 = animatedline('Color','b','LineWidth',2,'DisplayName',
    'angle_steer');  % Linea animata per il primo dato
%misurato2 = animatedline('Color','b','LineWidth',2,'DisplayName
    ', 'desired_filtered_torque');  % Linea animata per il
    secondo dato
misurato2 = animatedline('Color','b','LineWidth',1);  % Linea
    animata per il secondo dato
misurato3 = animatedline('Color','g','LineWidth',2,'DisplayName',
    'desired_speed_metsec');  % Linea animata per il terzo dato
misurato4 = animatedline('Color','y','LineWidth',2,'DisplayName',
    'roll');  % Linea animata per il quarto dato
misurato5 = animatedline('Color','b','LineWidth',2,'DisplayName',
    'speed_metsec');  % Linea animata per il primo dato
%misurato6 = animatedline('Color','r','LineWidth',2,'DisplayName
    ', 'torque');  % Linea animata per il secondo dato
misurato6 = animatedline('Color','r','LineWidth',1);  % Linea
    animata per il secondo dato
misurato7 = animatedline('Color','g','LineWidth',2,'DisplayName',
    'u_back_wheel');  % Linea animata per il terzo dato
misurato8 = animatedline('Color','y','LineWidth',2,'DisplayName',
    'u_front_wheel');  % Linea animata per il quarto dato
misurato9 = animatedline('Color','b','LineWidth',2,'DisplayName',
    'tempo');  % Linea animata per il primo dato
misurato10 = animatedline('Color','r','LineWidth',2);  % Linea
    animata per il secondo dato
misurato11 = animatedline('Color','g','LineWidth',2);  % Linea
    animata per il secondo dato

ax = gca;
xlabel('Tempo [s]');
ylabel('Valori');

```

```
31 title('Visualizzazione dei dati selezionati');
32 grid on;
```

2. Il programma prova ad eliminare la comunicazione seriale, se già presente; Il comando $s = serialport("COM5", 115200, 'DataBits', 8, 'StopBits', 1)$ apre la comunicazione USART sulla porta COM5 con un baud rate di 115200, utilizzando 8 bit di dati e 1 bit di stop. Ad ogni iterazione, una riga di dati viene letta tramite il comando $testo = readline(s)$. Il testo viene successivamente convertito in formato numerico, dopo di ché i diversi valori sono rappresentati graficamente in modo dinamico

```
33
34 try
35     delete(s);
36     clear s;
37 end
38 % Apertura porta seriale
39 s = serialport("COM5", 115200, 'DataBits', 8, 'StopBits', 1);
40 contatore = 0;
41 % Lettura dati
42 for i = 1 : N_campioni
43
44     tempo_corrente = (i - 1) * deltaTs; % Calcola il tempo
           corrente
45     while true % Ciclo che continua fino a quando non viene letta
           una stringa valida
46         testo = readline(s); % Legge una linea dalla porta
               seriale
47         testo = strtrim(testo); % Rimuove eventuali spazi bianchi
               e caratteri di ritorno a capo
48
49         % Controlla se la linea letta contiene il giusto numero
               di elementi
50         dati_letto = textscan(testo, "%f,%f"); % Prova a leggere
               5 numeri
51
52         % Verifica che siano stati letti 5 numeri
53         if numel(dati_letto) == 2 && all(~cellfun(@isempty,
               dati_letto))
54             % Estrai solo il primo, terzo e quinto dato
55             dati(i, :) = [dati_letto{1}, dati_letto{2}];
56
57             % Aggiungi i punti ai grafici animati
58             addpoints(misurato2, tempo_corrente, dati(i, 1)); %
                   Secondo dato
59             addpoints(misurato6, tempo_corrente, dati(i, 2)); %
                   Sesto dato
60             %Qua aggiungere le linee che si vogliono
                   rappresentare
```

```
61
62              drawnow limitrate; % Aggiorna il grafico con una
                    frequenza limitata
63              break; % Esci dal ciclo 'while' solo se la stringa
                     valida
64          else
65              fprintf("_stringa non valida o vuota: %s_\n", testo);
66              % Se la stringa    vuota o il formato non    corretto
                    , il ciclo 'while' continua
67          end
68      end
69  end
70
71  % Chiusura porta seriale
72  delete(s); clear s;
```

6.1.3 I2C

La comunicazione I2c è un protocollo di comunicazione seriale usato in genere per comunicazioni a corto raggio tra dispostivi su una scheda o tra componenti perfirici come sensori, display o altro. La comunicazione avviene su 2 canali, uno è il canale SCL (Serial Clock Line) che porta il segnale di clock per sincronizzare la comunicazione tra i dispostivi. L'altro canale è il SDA (Serial Data Line) che è usato per trasportare i dati. Il protocollo segue un'architettura master-slave, dove il master controlla il segnale di clock e il canale di comunicazione, mentre gli slave sono i dispositivi che rispondono alle richieste del master.

Nella bicicletta la comunicazione I2c viene usata per la comunicazione tra il sensore IMU e la scheda STM32.

6.2 Comunicazione Bluetooth

Per la comunicazione wireless, è stato scelto il microcontrollore Esp32, che offre sia connettività Bluetooth che Wi-Fi.

L'idea di base è stabilire una comunicazione tra la STM32 e l'Esp32 utilizzando l'interfaccia USART. L'Esp32, a sua volta, riceve i dati dalla STM32 e li trasmette via Bluetooth a MATLAB, che deve essere precedentemente collegato all'Esp32. La comunicazione avviene in entrambe le direzioni, con un canale dedicato alla ricezione e uno alla trasmissione dei dati.

6.2.1 Installazione Arduino IDE

Per poter utilizzare e programmare la scheda Esp32 con l'Arduino IDE, è necessario seguire alcuni passaggi fondamentali, descritti di seguito:

Installazione dei Driver

Per permettere al computer di riconoscere e comunicare correttamente con la scheda Esp32, è indispensabile installare i driver appropriati in base al tipo di chip presente sulla scheda. Nel caso specifico, il chip utilizzato è il CP210x [1], quindi è necessario installare i driver corrispondenti a questo modello.

Installazione della Libreria Esp32

Per configurare correttamente l'ambiente di sviluppo, è necessario aggiungere la scheda Esp32 all'Arduino IDE. A tal fine, segui questi passaggi:

1. Vai su "File" > "Preferences" e incolla il seguente URL nella casella "Additional Boards Manager URLs": `https://dl.espressif.com/dl/package_Esp32_index.json`.

2. Successivamente, apri il "Boards Manager" da "Tools" > "Board" > "Boards Manager".

3. Cerca e installa "Esp32 by Espressif Systems", in modo da includere il supporto per la scheda utilizzata, ovvero l'Esp32 Dev Module.

6.2.2 Flash codice su Esp32

Una volta installato e settato Arduino IDE, possiamo scrivere il seguente codice e flasharlo.

NOTA: per alcuni dispositivi Esp32 è necessario tenere premuto il tasto "Boot" durante il run del codice, in modo tale da far entrare il microcontrollore in modalita di flash.

```
1 void setup() {
2
3   delay(500);
4   Serial2.begin(115200,SERIAL_8N1,16,17);
5   serialBT.begin("Esp32BT");  //Bluetooth device name
6   //SerialBT.deleteAllBondedDevices(); // Uncomment this to delete
        paired devices; Must be called after begin
7
8 }
9
10 void loop() {
11
12  // Se ci sono dati disponibili sul canale Bluetooth, li legge e li
       invia al dispositivo STM32
13  if (serialBT.available()) {
14  char cmd = serialBT.read();
15  Serial2.print(cmd);
16  }
```

```
17  // Se ci sono dati disponibili dal dispositivo STM32, li invia
        attraverso il canale Bluetooth
18  if (Serial2.available())
19  {
20   serialBT.print(Serial2.readString());
21  }
22
23 }
```

6.2.3 Codice MATLAB che implementa la comunicazione

Una volta flashato il codice in Arduino IDE, aver alimentato l'Esp32 con 3.3Volt e aver connesso via bluetooth il pc con il microcontrollore, eseguendo il seguente codice matlab è possibile stabilire la connessione con l'Esp32.

Il codice è suddiviso in diverse funzioni che gestiscono la creazione di finestre grafiche, la selezione del tipo di connessione, l'invio di dati e la gestione della connessione stessa. Si utilizzano variabili globali per condividere dati tra le funzioni. Le sezioni principali sono:

1. Definizione delle variabili globali: All'inizio del codice, vengono definite varie variabili globali (fig1, fig2, connection, ecc.) utilizzate per gestire le finestre, la connessione, e i file di log.

2. Funzione `crea_file_di_testo`: Questa funzione chiede all'utente di selezionare un file di testo in cui salvare i dati. Se il file viene creato con successo, viene aperto per la scrittura.

3. Funzione `creazione_finestra`: Crea una finestra grafica (uifigure) che permette all'utente di scegliere il tipo di connessione (Bluetooth). Viene anche gestita la chiusura della finestra con un callback specifico. Funzione `selezionaConnessione`: Gestisce la selezione del tipo di connessione scelto dall'utente (Bluetooth) e stabilisce la connessione con il dispositivo (Esp32).

4. Funzione `trasmissioneDati`: In questa funzione si avvia un ciclo che monitora i dati in arrivo dalla connessione, leggendo i dati ricevuti e suddividendoli tramite la funzione splitstringa.

5. Funzioni per l'invio dei caratteri (`inviaCaratterev`, `inviaCarattereS`): Queste funzioni inviano specifici caratteri ('V' per avviare e 'S' per fermare) all'Esp32.

6. Funzione `inviaKpKiKd`: Gestisce l'invio dei parametri PID (Kp, Ki, Kd) al dispositivo tramite la connessione stabilita. L'utente inserisce i valori tramite un'interfaccia grafica e questi vengono inviati formattati.

7. Funzione `splitstringa`: Suddivide i dati ricevuti in una stringa separata da virgole, li converte in numeri e li stampa su un file di log.

8. Funzione `closeFig`: Chiude la finestra dell'interfaccia e la connessione, salvando il file di log se necessario.

Il codice è il seguente:

```
clear;
global tipoConnessione;
global connection;
global fig1;
global fig2;
global fileID;
global orario;
global descrizioni;
global filename;
global grafico_attivo;
grafico_attivo=0;
crea_file_di_testo;
creazione_finestra;


% Creazione della finestra  d e l l applicazione
function crea_file_di_testo()
global fileID;
global filename;
fileID=[];
% Chiedi  a l l utente  di selezionare il percorso e il nome del file
[filename, pathname] = uiputfile('*.txt', 'Seleziona la posizione e il
     nome del file');

% Controlla se  l utente  ha selezionato un file
if isequal(filename,0) || isequal(pathname,0)
    disp('Creazione del file annullata.');
else
    % Crea il percorso completo del file
    filepath = fullfile(pathname, filename);

    % Prova ad aprire il file in m o d a l i t a  scrittura
    try
        fileID = fopen(filepath, 'w');
        % Verifica se  l apertura  del file  e  riuscita
        if fileID ==-1
            error('Impossibile aprire il file.');
        else
            disp(['Il file ', filename, '    stato creato con successo
                .']);
        end
    catch ME
        % Visualizza eventuali errori
        disp(ME.message);
```

```
43      end
44  end
45  end
46
47
48
49
50
51  function creazione_finestra()
52  global fig1;
53  global connection;
54  fig1= uifigure('Name', 'Seleziona Connessione', 'Position', [100, 400,
        300, 200]);
55
56  % Creazione  d e l l etichetta  per il titolo
57  uilabel(fig1, 'Text', 'Seleziona la tua connessione', 'Position', [50,
        150, 400, 30], 'FontSize', 16, 'FontWeight','bold');
58
59  fig1.Resize = 'off';
60
61  % Creazione dei bottoni
62  %uicontrol(fig1, 'Style', 'push', 'String', 'WiFi', 'Position', [50,
      100, 100, 30], 'Callback', @(src, event)selezionaConnessione(src,
      event, 'wifi'));
63  uicontrol(fig1, 'Style', 'push', 'String', 'Bluetooth', 'Position',
      [100,100 , 100, 30], 'Callback', @(src, event)selezionaConnessione
      (src, event, 'bluetooth'));
64
65
66  % Assegna una funzione di callback per interrompere il programma
      quando la finestra viene chiusa
67  fig1.CloseRequestFcn = @(src, event) closeFig(src, event, connection);
68  end
69
70
71  % Funzione per la selezione del tipo di connessione
72  function selezionaConnessione(~, ~, tipo)
73  global tipoConnessione;
74  global fig2;
75  tipoConnessione = tipo;
76  global connection;
77  global fileID;
78  global descrizioni;
79  descrizioni=false;
80
81  disp('Tento la connessione...');
82  if exist('fig2', 'var')
83      delete(fig2);
84  end
85  if ~isempty(connection) && isvalid(connection)
86      delete(connection);
```

```
87      disp('Connessione chiusa.');
88      if(descrizioni==true)
89          fprintf(fileID, 'Connessione chiusa. \n');
90      end
91  end
92
93  if strcmp(tipoConnessione, 'wifi')
94      indirizzoIP = '192.168.4.1'; % Indirizzo IP  d e l l Esp32
95      porta = 80; % Porta su cui  l Esp32  sta ascoltando
96      try
97          connection = tcpclient(indirizzoIP, porta);
98          disp('Connessione wifi stabilita');
99          if(descrizioni==true)
100             fprintf(fileID,'Inizio connessione wifi. \n');
101         end
102     catch
103         % Se la connessione WiFi non riesce, mostra un messaggio  di
              errore
104         disp('Impossibile stabilire la connessione WiFi. Assicurati
              che il server sia in esecuzione.');
105
106     end
107 elseif strcmp(tipoConnessione, 'bluetooth')
108     try
109         connection = bluetooth("Esp32BT", 1);
110         disp('Connessione Bluetooth stabilita');
111         if(descrizioni==true)
112             fprintf(fileID,'Inizio connessione Bluetooth. \n');
113         end
114     catch
115         % Se la connessione WiFi non riesce, mostra un messaggio di
              errore
116         disp('Impossibile stabilire la connessione Bluetooth .
              Assicurati che il server sia in esecuzione.');
117     end
118
119 end
120 if ~isempty(connection) && isvalid(connection)
121     trasmissioneDati(connection);
122 end
123
124
125 end
126
127 function trasmissioneDati(connessione)
128 global fig1;
129 global filename;
130 global fig2;
131
132 fig2 = uifigure('Name', 'Bottoni', 'Position', [600, 400, 300, 200],'
      CloseRequestFcn', @closeRequestFcn);
```

```
133 fig2.Resize = 'off';
134
135
136 figKpKiKd = uifigure('Name', 'Invia Kp, Ki, Kd', 'Position', [600,
        400, 350, 250]);
137
138 % Campi di input per Kp, Ki e Kd
139 uilabel(figKpKiKd, 'Position', [20, 200, 50, 22], 'Text', 'Kp:');
140 kpField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 200, 100,
         22]);
141
142 uilabel(figKpKiKd, 'Position', [20, 150, 50, 22], 'Text', 'Ki:');
143 kiField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 150, 100,
         22]);
144
145 uilabel(figKpKiKd, 'Position', [20, 100, 50, 22], 'Text', 'Kd:');
146 kdField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 100, 100,
         22]);
147
148 % Pulsante per inviare i dati
149 uibutton(figKpKiKd, 'Position', [70, 50, 100, 30], 'Text', 'Invia',
        ...
150     'ButtonPushedFcn', @(src, event)inviaKpKiKd(src, event,
            connessione, kpField, kiField, kdField));
151
152
153
154
155 btnStart = uicontrol(fig2, 'Style', 'pushbutton', 'String', 'Start (V)
        ', 'Position', [50, 50, 100, 30], 'Callback', @(src, event)
        inviaCarattereV(src, event, connessione));
156 btnStop = uicontrol(fig2, 'Style', 'pushbutton', 'String', 'Stop (S)',
         'Position', [150, 50, 100, 30], 'Callback', @(src, event)
        inviaCarattereS(src, event, connessione));
157
158
159
160     function closeRequestFcn(~, ~)
161         disp("Spiacenti, non pu i chiudere questa finestra.");
162     end
163 while isvalid(fig1)
164     if isvalid(connessione) && connessione.BytesAvailable > 0
165         % Lettura di dati
166         %datiRicevuti = readline(connessione);
167         %Visualizzazione dei dati
168         %splitstringa(datiRicevuti);
169
170         % Lettura di dati
171         datiRicevuti = readline(connessione);
172
173         % Verifica che ci siano dati ricevuti
```

```
174         if ~isempty(datiRicevuti)
175             % Esegui la suddivisione solo se ci sono dati validi
176             splitstringa(datiRicevuti);
177         else
178             disp('Nessun dato ricevuto.');
179         end
180
181
182     end
183     % Pausa per evitare sovraccarico della CPU
184     pause(0.005);
185 end
186 end
187
188 % Funzione per  l invio  del carattere
189 function inviaCarattere(~, ~, connection, carattere)
190 write(connection, carattere, "char");
191 disp(["Carattere ",carattere," inviato."]);
192 end
193
194 function inviaCarattereV(~, ~, oggConnessione)
195 % Invia il carattere    V     a l l Esp32
196 write(oggConnessione, 'V', "char");
197 disp('Carattere "V" inviato.');
198
199
200 end
201
202
203 % Funzione per aprire la finestra Kp, Ki, Kd
204 function apriFinestraKpKiKd(~, ~, connessione)
205 figKpKiKd = uifigure('Name', 'Invia Kp, Ki, Kd', 'Position', [600,
        400, 350, 250]);
206
207 % Campi di input per Kp, Ki e Kd
208 uilabel(figKpKiKd, 'Position', [20, 200, 50, 22], 'Text', 'Kp:');
209 kpField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 200, 100,
        22]);
210
211 uilabel(figKpKiKd, 'Position', [20, 150, 50, 22], 'Text', 'Ki:');
212 kiField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 150, 100,
        22]);
213
214 uilabel(figKpKiKd, 'Position', [20, 100, 50, 22], 'Text', 'Kd:');
215 kdField = uieditfield(figKpKiKd, 'numeric', 'Position', [70, 100, 100,
        22]);
216
217 % Pulsante per inviare i dati
218 uibutton(figKpKiKd, 'Position', [70, 50, 100, 30], 'Text', 'Invia',
      ...
219     'ButtonPushedFcn', @(src, event)inviaKpKiKd(src, event,
```

```
        connessione, kpField, kiField, kdField));
220 end
221
222 % Funzione per inviare Kp, Ki, Kd alla connessione
223 function inviaKpKiKd(~, ~, connessione, kpField, kiField, kdField)
224 kp = kpField.Value;
225 ki = kiField.Value;
226 kd = kdField.Value;
227
228 % Stringa formattata per inviare Kp, Ki, Kd
229 stringaDaInviare = sprintf('P');
230 %stringaDaInviare = sprintf("ciao")
231 % Invia la stringa formattata all'Esp32 tramite la connessione
232 %write(connessione,'PID', 'char');
233 write(connessione, stringaDaInviare, "char");
234 write(connessione, kp, "single");
235 write(connessione, ki, "single");
236 write(connessione, kd, "single");
237
238 %disp(['Valori inviati: ', stringaDaInviare]);
239 fprintf("%s%f%f%f\n",stringaDaInviare,kp,ki,kd);
240 end
241
242 function inviaCarattereS(~, ~, oggConnessione)
243 % Invia il carattere    S     a l l Esp32
244
245 write(oggConnessione, 'S', "char");
246 disp('Carattere "S" inviato.');
247 end
248
249 function splitstringa (stringaInput)
250 global fileID;
251 global orario;
252 orario=false;
253 valoriCell = strsplit(stringaInput, ',');
254 % Converti i valori da stringhe a numeri
255 valoriNumerici = str2double(valoriCell);
256 % Visualizza i valori numerici
257 disp(valoriNumerici);
258 fprintf(fileID, '%f ', valoriNumerici);
259 if(orario==true)
260     currentTime = datetime('now', 'Format', 'HH:mm:ss');
261     fprintf(fileID, '%s', char(currentTime));
262 end
263 fprintf(fileID, '\n');
264 end
265
266 function closeFig(src, ~, connessione)
267 global fileID;
268 global descrizioni;
269 global fig2;
```

```
delete(src);
delete(fig2);
disp("Chiusura del programma e interruzione della connessione");
if(descrizioni==true)
    fprintf(fileID, 'Connessione chiusa. \n');
    fclose(fileID);
end
% Chiudi la connessione
if ~isempty(connessione) && isvalid(connessione)
    delete(connessione);
end

end
```

Eseguito il codice abbiamo la possibilità di:

1. Creare un file dove salvare i dati mandati dall'Esp32.

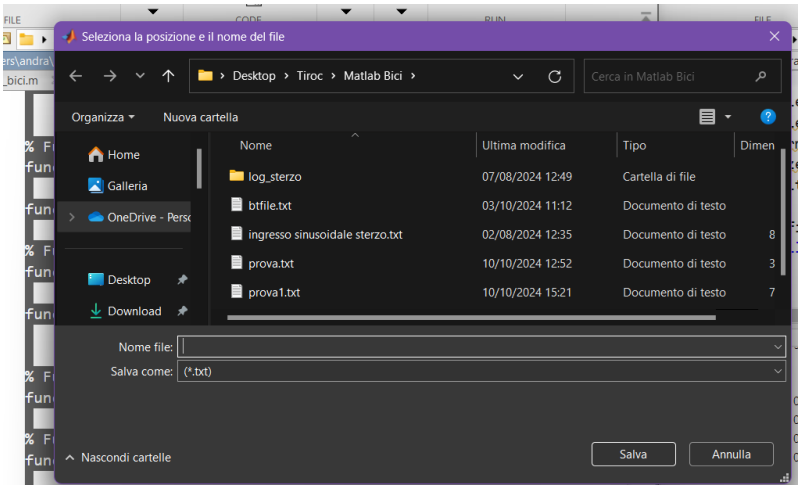

Figure 6.3: *Finestra per la creazione file*

2. Connettere Matlab con l'Esp32 via bluetooth.

Figure 6.4: *Finestra per avviare la connessione via bluetooth*

3. Avviare la comunicazione con il tasto Start(V) o fermare la comunicazione con il tasto Stop(S).

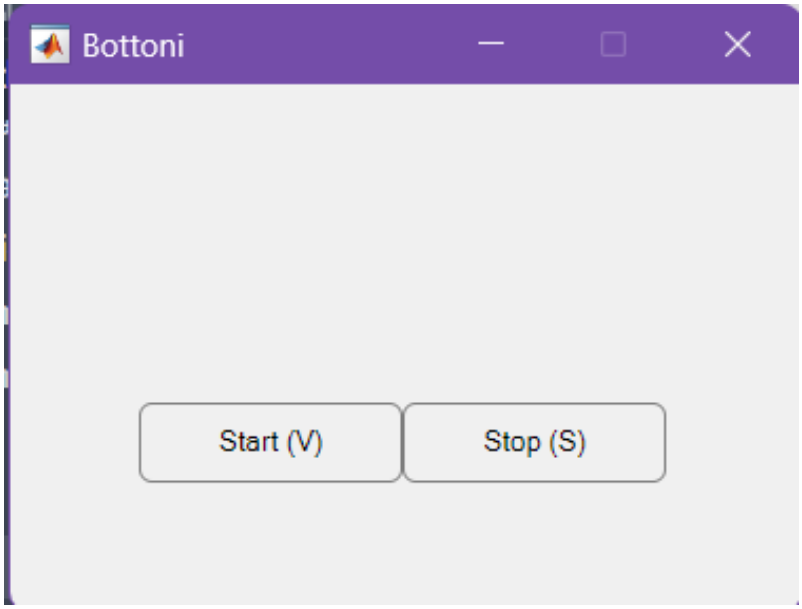

Figure 6.5: *Finestra per avviare la tramissione dei dati o interromperla*

4. Settare i parametri di un PID, in particolare è stato usato per settare i parametri del PID dell'IMU, senza dover ogni volta ricollegare la bicicletta al cavo e flashare il codice.

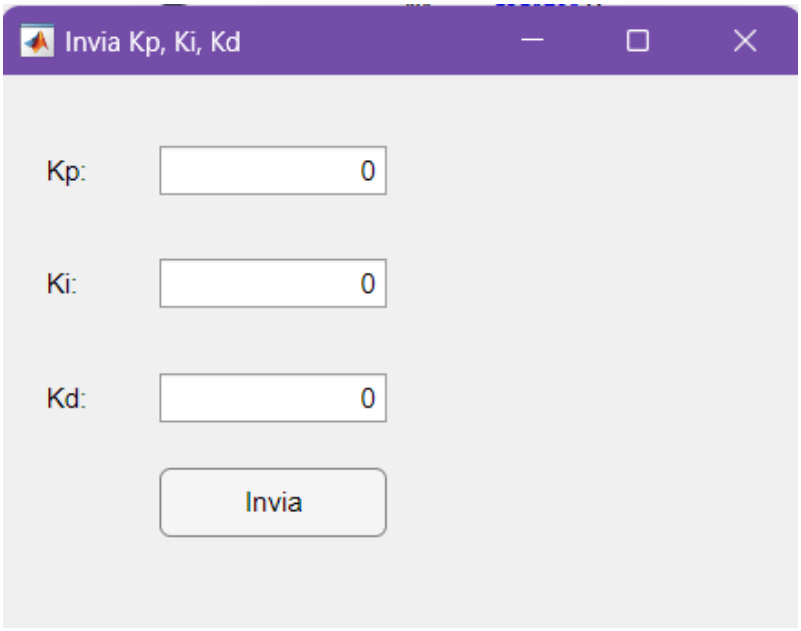

Figure 6.6: *Finestra per settare i parametri di un PID*

6.2.4 Codice STM32 per la Trasmissione dei Dati via Bluetooth

Una volta stabilita la comunicazione Bluetooth, quando la periferica USART (in modalità di ascolto) riceve un byte, viene attivata una funzione di callback che gestisce le seguenti situazioni:

- Lettura del carattere 'V': La comunicazione viene avviata e la variabile `trasmissione_attiva` viene impostata a 1;
- Lettura del carattere 'S': La comunicazione viene interrotta e `trasmissione_attiva` viene impostata a 0;
- Lettura del carattere 'P': In seguito alla ricezione di questo carattere, vengono letti tre ulteriori valori di tipo `float`, che rappresentano i tre parametri del PID. Dopo aver ricevuto i parametri, il PID viene configurato.

```
1
2 // Funzione per trasmettere dati tramite UART2
3 void Trasmissione_dati(void *data, size_t size) {
4     HAL_UART_Transmit(&huart2, (uint8_t*) data, size, HAL_MAX_DELAY);
5 }
6
7 // Callback chiamata quando un byte viene ricevuto sulla UART2
8 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
9     if (huart == &huart2) {
10         if (rx_buffer[0] == 'S') {
11             // Interrompi la trasmissione
12             printf("Arrivato: %c\r\n", rx_buffer[0]);
```

```
13              trasmissione_attiva = 0;
14
15          } else if (rx_buffer[0] == 'V') {
16              // Avvia la trasmissione
17              printf("Arrivato: %c\r\n", rx_buffer[0]);
18              trasmissione_attiva = 1;
19          } else if (rx_buffer[0] == 'P') {
20              //HAL_UART_Receive_IT(&huart2, (uint8_t*) rx_buffer, 1);
                    //
21              i = 1;
22              // Ricevi 4 byte tramite UART (blocca fino a ricezione)
23              HAL_UART_Receive_IT(&huart2, &bytesricevuti, 12);
24              rx_buffer[0] = 0;
25              // Ricostruisci il float dai 4 byte ricevuti
26              bytesricevuti1[0] = bytesricevuti[0];
27              bytesricevuti1[1] = bytesricevuti[1];
28              bytesricevuti1[2] = bytesricevuti[2];
29              bytesricevuti1[3] = bytesricevuti[3];
30              bytesricevuti2[0] = bytesricevuti[4];
31              bytesricevuti2[1] = bytesricevuti[5];
32              bytesricevuti2[2] = bytesricevuti[6];
33              bytesricevuti2[3] = bytesricevuti[7];
34              bytesricevuti3[0] = bytesricevuti[8];
35              bytesricevuti3[1] = bytesricevuti[9];
36              bytesricevuti3[2] = bytesricevuti[10];
37              bytesricevuti3[3] = bytesricevuti[11];
38
39              //printf("float: %f",floatricevuto);
40
41          }
42
43          HAL_UART_Receive_IT(&huart2, (uint8_t*) rx_buffer, 1); //
44          memcpy(&floatricevuto1, &bytesricevuti1, sizeof(float));
45          memcpy(&floatricevuto2, &bytesricevuti2, sizeof(float));
46          memcpy(&floatricevuto3, &bytesricevuti3, sizeof(float));
47
48          if (i == 1) {
49              tune_PID(&pid_roll, floatricevuto1, floatricevuto2,
                    floatricevuto3); //prova ad alzare
50              i = 0;
51
52          }
53
54      }
55  }
```

Quando la trasmissione è attiva (`trasmissione_attiva` = 1), un timer invia tutti i dati raccolti ogni 3 secondi e successivamente resetta il buffer che li contiene.

```
1       if (htim == &htim6) {
2
```

```
3         // Gestione  d e l l invio  periodico dei dati
4
5         if (trasmissione_attiva == 1) {
6             Trasmissione_dati(bufferDati, indiceBuffer);
7             indiceBuffer = 0;
8             //in genere indice buffer arriva a sui 15000 ogni 3
                  secondi
9             if (indiceBuffer >= 30000) //se per qualche motivo non
                  avviene l'azzeramento dell'indice
10                indiceBuffer = 0;
11            memset(bufferDati, 0, sizeof(bufferDati));  // Pulizia
                  buffer
12
13        }
```

Durante ogni iterazione (che avviene ogni 0.01s) all'interno del ciclo `while`, i dati vengono aggiornati e salvati in una struttura `Datibici`.

```
1     if (trasmissione_attiva == 1) {
2                 //dati bicicletta
3
4                 int bytesWritten = sprintf(&bufferDati[indiceBuffer],
5                         "%f,%f,%f,%f,%f,%f,%f,%f,%f%d\n",
6                         datibici.angle_steer, datibici.
                              desired_filtered_torque,
7                         datibici.desired_speed_metsec, datibici.roll,
8                         datibici.speed_metsec, datibici.torque,
9                         datibici.u_back_wheel, datibici.u_front_wheel,
10                        datibici.tempo, datibici.corrente_non_filtrata
                              ,
11                        datibici.corrente_filtrata);
12                indiceBuffer += bytesWritten;
13    }
```

6.2.5 Codice MATLAB per la Visualizzazione dei Dati

Il seguente codice MATLAB consente di graficare i valori ricevuti via Bluetooth, a condizione che i dati siano stati preventivamente salvati in un file. La visualizzazione non avviene in tempo reale, poiché i dati vengono aggiornati ogni 3 secondi. Tuttavia, questo approccio è utile per analizzare il comportamento delle variabili durante il movimento della bicicletta, fornendo una panoramica dell'andamento complessivo.

```
1     filename = ['prova1.txt']
2 fileId = fopen(filename,"r")
3 formatSpec = '%f \n';
4 sizeA = [9 inf];
5 deltaTs = 0.01; % Intervallo di campionamento (10 ms)
6
```

```
f=figure;
while true
    %A=fscanf
    A = dlmread(filename, ' ', 2, 0)
    A1 = A(:,1);
    A2 = A(:,2);
    A3 = A(:,3);
    A4 = A(:,4);
    A5 = A(:,5);
    A6 = A(:,6);
    A7 = A(:,7);
    A8 = A(:,8);
    A9 = A(:,9);


    % Calcola il numero di campioni letti
    N = length(A1);

    % Crea un vettore per il tempo basato sul numero di campioni e
        deltaTs
    tempo = (0:N-1) * deltaTs; % Crea un vettore del tempo in secondi

    % Traccia i grafici usando il tempo sull'asse x
    % Qui per esempio rappresento solo 2 valori
    plot(tempo, A2, 'DisplayName', 'desired_filtered_torque'); hold on
        ;
    plot(tempo, A6, 'DisplayName', 'torque');


    plot(A1, 'DisplayName', 'angle_steer'); hold on;
    plot(A2, 'DisplayName', 'desired_filtered_torque');
    plot(A3, 'DisplayName', 'desired_speed_metsec');
    plot(A4, 'DisplayName', 'roll');
    plot(A5, 'DisplayName', 'speed_metsec');
    plot(A6, 'DisplayName', 'torque');
    plot(A7, 'DisplayName', 'u_back_wheel');
    plot(A8, 'DisplayName', 'u_front_wheel');
    plot(A9, 'DisplayName', 'tempo');
    legend();
    xlabel('Tempo [s]');
    ylabel('Valori');
    drawnow()
    pause(3)
    hold off;
end
```

Capitolo 7

Test e risultati

7.0.1 Test PID per la trazione

I parametri del PID per la trazione sono i seguenti:

- $K_P = 14$
- $K_I = 40$
- $K_D = 0$

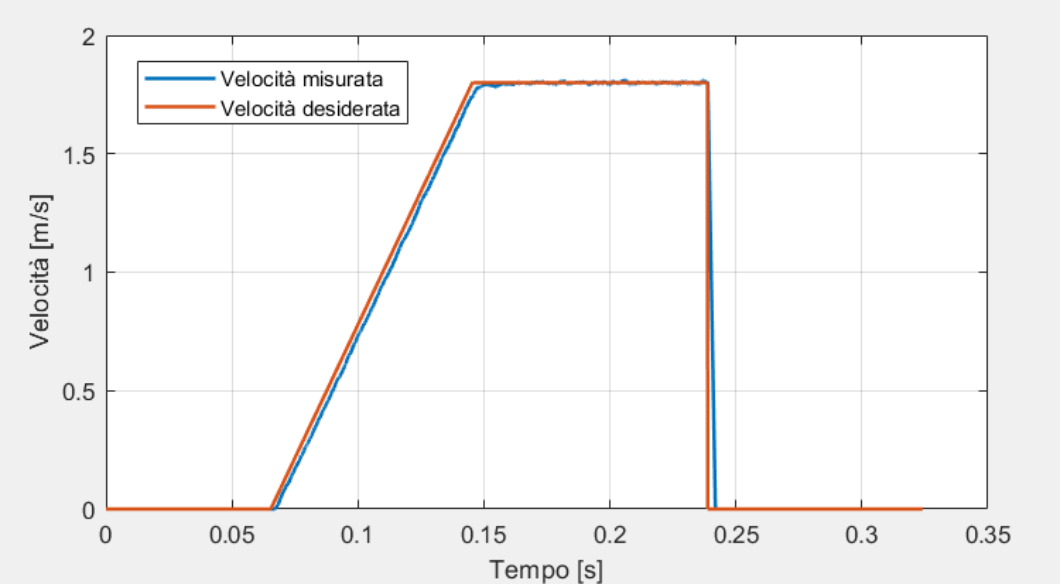

Figure 7.1: *Grafico dell'andamento della velocità misurata e della velocità desiderata*

7.0.2 Test PID Roll e PID Sterzo

Il PID responsabile della generazione della coppia desiderata a partire dall'angolo di rollio è stato il più complesso da configurare. L'unica velocità su cui si è concentrato il mantenimento dell'equilibrio è stata di 6.7 km/h, corrispondente alla velocità massima supportata dal motore dedicato alla trazione.

I PID per il roll e per lo sterzo risultavano strettamente interconnessi: ad esempio, aumentando il guadagno proporzionale del primo, un errore di rollio generava una coppia maggiore, richiedendo al sistema di sterzare più rapidamente. In caso contrario, la bicicletta oscillava e perdeva l'equilibrio. Tuttavia, una maggiore reattività del PID dello sterzo comportava movimenti eccessivamente rapidi e piccoli della ruota anteriore, rischiando di destabilizzare l'e-Bike e causarne la caduta.

Sono stati trovati 2 "PID roll" che mantengono l'equilibrio, a cui corrispondono 2 "PID sterzo".

Il primo "PID roll" è:

- $K_P = 0.000081$
- $K_I = 0$
- $K_D = 0.000300$

a cui corrisponde il "PID sterzo":

- $K_P = 120000$
- $K_I = 40000$
- $K_D = 0$

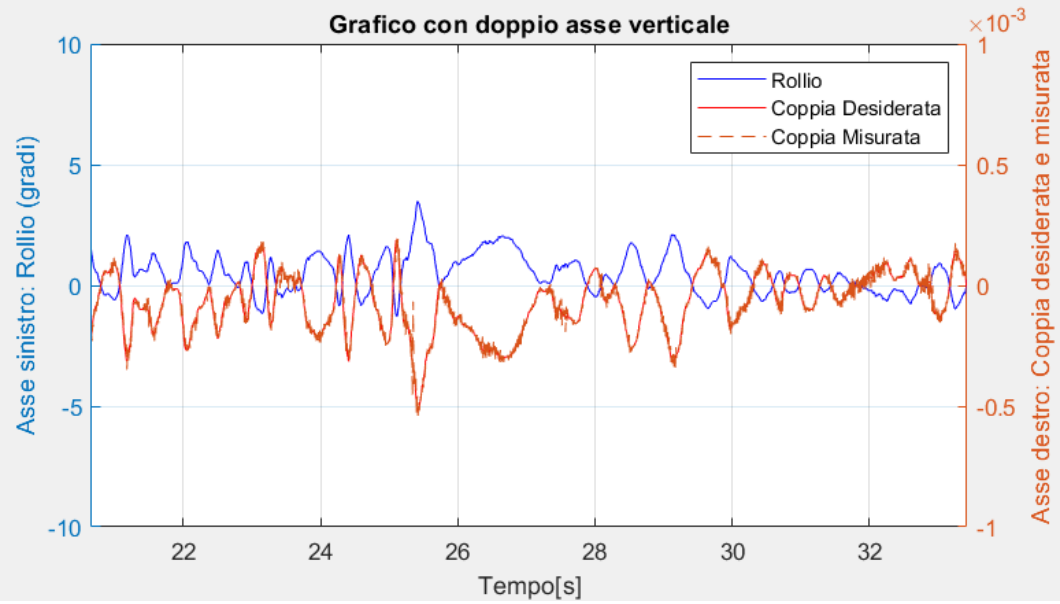

Figure 7.2: *Prima versione di PID Roll e PID Sterzo*

Il secondo "PID roll" è:

- $K_P = 0.000150$
- $K_I = 0$

- $K_D = 0.000600$

a cui corrisponde il "PID sterzo":

- $K_P = 120000$
- $K_I = 30000$
- $K_D = 0$

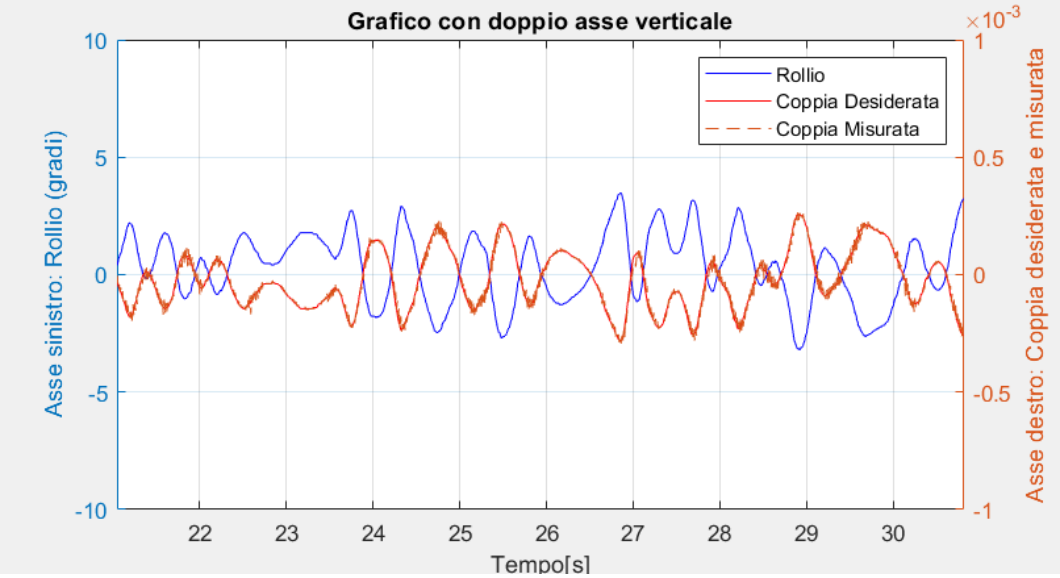

Figure 7.3: *Seconda versione di PID Roll e PID Sterzo*

Figure 7.4: *Foto della bicicletta in equilibrio sul tapis roulant*

7.1 Conclusioni e sviluppi futuri

In conclusione, le soluzioni adottate mostrano essere adeguate per mantenere in equilibrio l'e-Bike anche se necessitano di ulteriori messe a punto visto che attualmente è necessaria qualche correzione di supporto esterna. I PID che hanno dato i migliori risultati sono mostrati nella figura 7.3. Tuttavia, la bicicletta non riesce ancora a bilanciarsi completamente da sola, a causa di un offset non corretto nell'angolo di rollio rilevato, che occorre sistemare. Durante i test, si sono verificati episodi in cui l'angolo di equilibrio variava da un lato all'altro della bicicletta, senza alcuna modifica al codice. Questo problema potrebbe essere attribuito ad un fissaggio meccanico non ottimale dell'IMU o ad una sensibilità insufficiente del sensore stesso.

Un altro aspetto rilevato riguarda l'offset dell'ADC, che è ricomparso in alcuni casi, sebbene risulti meno significativo rispetto alla situazione iniziale.

Grazie all'oversampling, la precisione della misura dell'ADC è migliorata notevolmente. Tuttavia, c'è ancora margine di miglioramento. Ad esempio, si potrebbe leggere il valore dell'ADC durante il periodo di inattività della scheda STM, ovvero quando non sta eseguendo il codice previsto ogni 0.01 secondi. In questo intervallo, prima che la `flag_Tc` assuma il valore 1 (indicante il trascorrere dei 0.01 secondi), il sistema potrebbe effettuare letture multiple e calcolare una media, ottenendo così una misura ancora più precisa.

Per sviluppi futuri, sarebbe utile esplorare ulteriori strategie per migliorare la precisione dei sensori e ottimizzare il controllo del PID, soprattutto per ridurre la necessità di un intervento esterno e migliorare l'autonomia della bicicletta nel mantenimento dell'equilibrio. Un'altra area da approfondire potrebbe essere l'integrazione di algoritmi di filtraggio più avanzati per stabilizzare le letture dell'IMU.

Una volta perfezionato il controllo del rollio, si potrebbe introdurre un controllo sullo yaw rate (velocità angolare attorno all'asse verticale), permettendo così alla bicicletta di seguire una traiettoria desiderata. Questo approccio potrebbe essere sviluppato sulla base delle indicazioni presenti nell'articolo del professore, migliorando ulteriormente la manovrabilità e l'autonomia della e-Bike nella guida autonoma.

Bibliografia

[1] URL: https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads.

[2] Batteria fullpower silver edition v2. URL: https://www.eolomodellismo.it/oscommerce/product_info.php?products_id=1703.

[3] Batteria turnigy nano-tech a-spec g2. URL: https://hobbyking.com/it_it/turnigy-nano-tech-a-spec-g2-2600mah-4s-65-130c-lipo-pack.html.

[4] Lm2596 regolatore di tensione step down buck converter. URL: https://t.ly/XQzl9.

[5] Md10c dcdriver. URL: https://www.cytron.io/p-10amp-5v-30v-dc-motor-driver.

[6] Motor - dcx35l gb kl 18v. URL: https://www.maxongroup.co.uk/maxon/view/product/motor/dcmotor/DCX/DCX35/DCX35L01GBKL511.

[7] A.Scala Giuseppe Bonci Andrea, Longhi Sauro. Revised double loop vehicle controller for trajectory tracking of motorcycles.

[8] Perugini Andrea e Praolini Mattia. Filtro di kalman applicato ad un sensore di corrente, 2024. URL: https://www.overleaf.com/read/jbqzxhpnvjgz#a38fce.

[9] Wikipedia. Microcontrollore. URL: https://it.wikipedia.org/wiki/Microcontrollore.