POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Aerospaziale

Tesi Magistrale

Processor in the loop simulations and Code generation for Unmanned Aerial systems

Processor o Software?

Relatore:
Prof.ssa Elisa Capello

Co-Relatore:
Dott. Davide Carminati

Autore:
Luigi Sante

Dicembre 2020

Sommario

Ringraziamenti

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

La tesi prevede lo sviluppo del software necessario per il governo di un velivolo a pilotaggio remoto, implementando diversi modelli di controllo e validando il controllore con diverse simulazioni.

Data la complessità della generazione del software necessario alla guida e controllo dei velivoli di questo tipo, l'approccio più intuitivo, che permette la collaborazione tra diverse figure professionali quali programmatori, ingegneri aerospaziali e meccatronici, risulta essere sicuramente basata sull'utilizzo di una logica progettuale a modelli. Questo tipo di approccio permette di semplificare lo sviluppo del software, concentrando gli sforzi di chi si occupa della creazione dei modelli di controllo sugli aspetti effettivamente importanti della funzionalità di questi, lasciando la complessità della generazione di algoritmi e le relative basi di dati a chi ha maggiore competenza informatica, attraverso lo sviluppo degli strumenti che automatizzano la generazione del codice e la sua compilazione. Lo sviluppo del software effettuato in questo modo è accompagnato per ogni passaggio dalla corrispettiva fase di verifica: MIL, SIL, PIL, HIL; aspetto derivato proprio dal classica metodologia di sviluppo del software utilizzante il modello denominato a V, e pienamente solidificato in molti settori industriali.

É stato scelto di utilizzare per l'implementazione dell'autopilota il dispositivo PixHawk con il relativo firmware PX4. Lo sviluppo del software avviene quindi specificatamente per questo ecosistema di prodotti e relativi firmware. Il codice è open-source, permettendo la possibilità di essere studiato e modificato. Il codice si trova ad uno stato abbastanza evoluto nello sviluppo, possedendo tutta una serie di funzionalità già sufficiente per gestire un velivolo a pilotaggio remoto. Nel caso però si voglia inserire un personale modello di guida e controllo o di un nuovo stimatore, non sono previsti nativamente strumenti di modellazione, limitando lo sviluppo del software solo alle persone con una competenza informatica alta, incrementando la complessità di progettazione stessa. In definitiva si ha difficoltà di creazione del codice sorgente partendo da un modello ,esistono comunque gli strumenti per verificare il codice tramite SIL e PIL.

In modo complementare all'utilizzo di PX4, si affianca l'utilizzo di un tool specifico per Simulink, chiamato "Embedded Coder Support Package for PX4 Autopilots". Grazie a questo tool si permette la generazione del codice in modo automatico partendo dai modelli creati su Simulink e l'inserimento conforme del modulo prodotto all'interno del firmware e delle restanti funzionalità. Integrando quindi gli strumenti di compilazione presenti in PX4 si possono effettuare le simulazioni SIL e PIL, avvicinando lo sviluppo ad una approccio più industriale.

Per testare correttamente il funzionamento di un velivolo a pilotaggio remoto, occorre ricreare in un ambiente simulato il velivolo stesso e le interazioni principali con l'ambiente in cui opera. La scelta dell'ambiente simulato ricade nell'utilizzo di Gazebo. Come anticipato precedentemente la procedura di SIL è nativamente presente nel progetto di PX4. I simulatori selezionabili per questa fase di test sono svariati, il tool di Simulink però limita la scelta ad un simulatore esterno solamente, ovvero JMavSim. Attraverso una procedura particolare però si è reso possibile l'utilizzo di Gazebo, ambiente più versatile e completamente personalizzabile.

Sono stati creati due modelli di controllo delle diverse dinamiche del velivolo e successivamente alla corretta configurazione dei parametri dei controllori, sono stati messi a confronto attraverso simulazioni SIL. Per quando riguarda il controllo di posizione è stato utilizzato un semplice controllore PID, mentre per il controllo di quota e di assetto, sono stati utilizzati rispettivamente un controllore PID e un controllore SMC.

Che tipo di algoritmo di guida è stato utilizzato?

Non è invece stato possibile effettuare correttamente la simulazione PIL a causa di alcuni problemi di comunicazione tra simulatore PixHawk. La trasmissione tra simulatore e autopilota avviene correttamente per quanto riguarda la gestione dei dati dei sensori, ciò non viene altrettanto per quanto riguarda i messaggi da fornire agli attuatori. Questo problema é probabilmente causato dalla differenza sostanziale che c'è tra l'applicazione generata dal tool di Simulink e la logica di base di PX4 nella gestione dei messaggi agli attuatori. Questo aspetto richiede ulteriori indagini in lavori futuri.

Capitolo 2

Revisione della letteratura

Capitolo 3

Sistema Quadrirotore

3.1 Descrizione del drone

3.2 PX4 Autopilot

Il firmware utilizzato nelle simulazioni di software in the loop e processor in the loop è il PX4 Autopilot.

Questo software mette a disposizione diverse funzionalità per avere un sistema di gestione e controllo robusto e affidabile, implementato in diversi tipi di sistemi. L'implementazione non è quindi specifica solo a mezzi aerei di qualsiasi configurazione, ma anche a velicoli di terra, marini e razzi. Il software è open-source e vanta del contributo di parecchi sviluppatori, dagli esperti del settore a contributi di livello accademico. Lo sviluppo open-source permette quindi di aggiungere o modificare le funzionalità messe a disposizione in modo da soddisfare le proprie esigenze e arricchire il progetto generico di nuove funzionalità utili ad altri. Il sistema operativo sulla quale viene eseguito materialmente il codice può essere Nuttx o Linux/MacOS la cui distinzione principale in questa applicazione è solo nella gestione di task e thread.

Il sistema operativo Nuttx è un sistema RTOS (Real-Time Operating System) è svilupato appositamente per implementazioni embedeed. Essendo sviluppato per un contesto specifico ha tutte le caratteristiche necessarie per essere eseguito in sistemi che devono avere prestazioni migliori con poche risorse disponibili. Vengono utilizzati gli standard POSIX e ANSI. Inoltre, sono implementate funzionalità di programmazione concorrenziale per l'esecuzione di processi in parallelo. Le funzionalità del firmware vengono eseguite in questo sistema come task separati e ogni task può eseguire diversi thread. Nell'implementazione su sistemi Linux/MacOS invece i moduli sono eseguiti come thread del processo principale, non c'è quindi una distinzione tra threads e tasks, oltre a non essere ottimizzato per sistemi embedeed.

3.2.1 Architettura del software

Il firmware è principalmente suddiviso in due categorie di moduli:

- **Flight stack** : composta dalla parte che stima lo stato del sistema e il relativo controllo

- **Middleware** : composta dalle interfacce che collegano i vari moduli interni di PX4 tra di loro e verso l'esterno, con la possibilità di integrare gli hardware utilizzati.

Il sistema quindi separa le varie funzionalità in moduli separati, eseguiti in modo indipendente che scambiano i dati e comandi tra di loro e con l'esterno attraverso messaggi asincroni. Nella figura 3.1 è riportato lo schema di alto livello del software di PX4 e la sua modularità.

Flight stack

Il flight stack, mostrato in figura 3.2 è l'insieme di moduli che si occupano della stima dello stato del sistema e di tutti le funzionalità per il controllo,la guida e la navigazione. Esiste anche un modulo per interfacciarsi con il volo manuale attraverso radiocomando.

Estimator L'estimator è il modulo che prendendo i dati da uno o più sensori determina lo stato attuale del velivolo. E' possibile selezionare diversi tipi di estimato, quello selezionato in questo caso è uno stimatore con filtro di Kalman.

Controller Si occupa di prendere in input i vari punti della pianificazione e confrontarli con lo stato attuale determinato dell'estimator. In questo modo vengono determinati i segnali di comando di output che saranno poi elaborati dal mixer. Questa funzionalità è implementata da più moduli, suddividendo la dinamica lenta e la dinamica veloce del drone. Verrà disabilitata la sua funzionalità per sostituirla con l'applicazione sviluppata su Simulink.

Mixer Il mixer si occupa di tradurre i segnali del comandi normalizzati di rollio, imbardata, beccheggio e manetta, da consegnare all'hardware che genera gli impulsi pwm utilizzati per il controllo del motore.

Middleware

Questo insieme di moduli si occupa invece di tutte le comunicazioni interne tra processi e tra PX4 e il mondo esterno. É composta principalmente dai driver per i sensori, i canali di comunicazione verso l'esterno e il bus di trasmissione di messaggi attraverso µORB e MAVLink. In questo contesto ricade anche la connessione con un simulatore per testare il codice generato nelle varie fasi di validazione.

3.2.2 Strumenti per lo sviluppo del codice

L'intero codice del firmware PX4 viene messo a disposizione attraverso la piattaforma github. Il progetto contiene all'interno le toolchain necessarie per compilare il sistema nei vari sistemi operativi. Agendo sulle varie possibili configurazioni di compilazione è possibile modificare e aggiunge delle funzionalità. Modificando le impostazioni di compilazione si può generare il programma finale da caricare ed eseguire sull'autopilota. Sono presenti anche delle configurazioni per effettuare l'analisi e la verifica del codice generato attraverso l'utilizzo di un ambiente simulato. I simulatori che presentano una configurazione di base sono : Gazebo, jMAVSim , AirSim, Xplane. Per quanto riguarda questa tesi, verrà utilizzato il software Gazebo sfruttando parzialmente il codice già presente e adottando alcune modifiche.

Nulla vieta comunque di poter collegare un simulatore diverso attraverso la creazione di un interfaccia dati con il firmware. Infatti, la connessione viene effettuata attraverso UDP o seriale, utilizzando su essa il protocollo MAVLink.

Prendere spunto dalla guida al tool di simulink

Parlare del codice generato

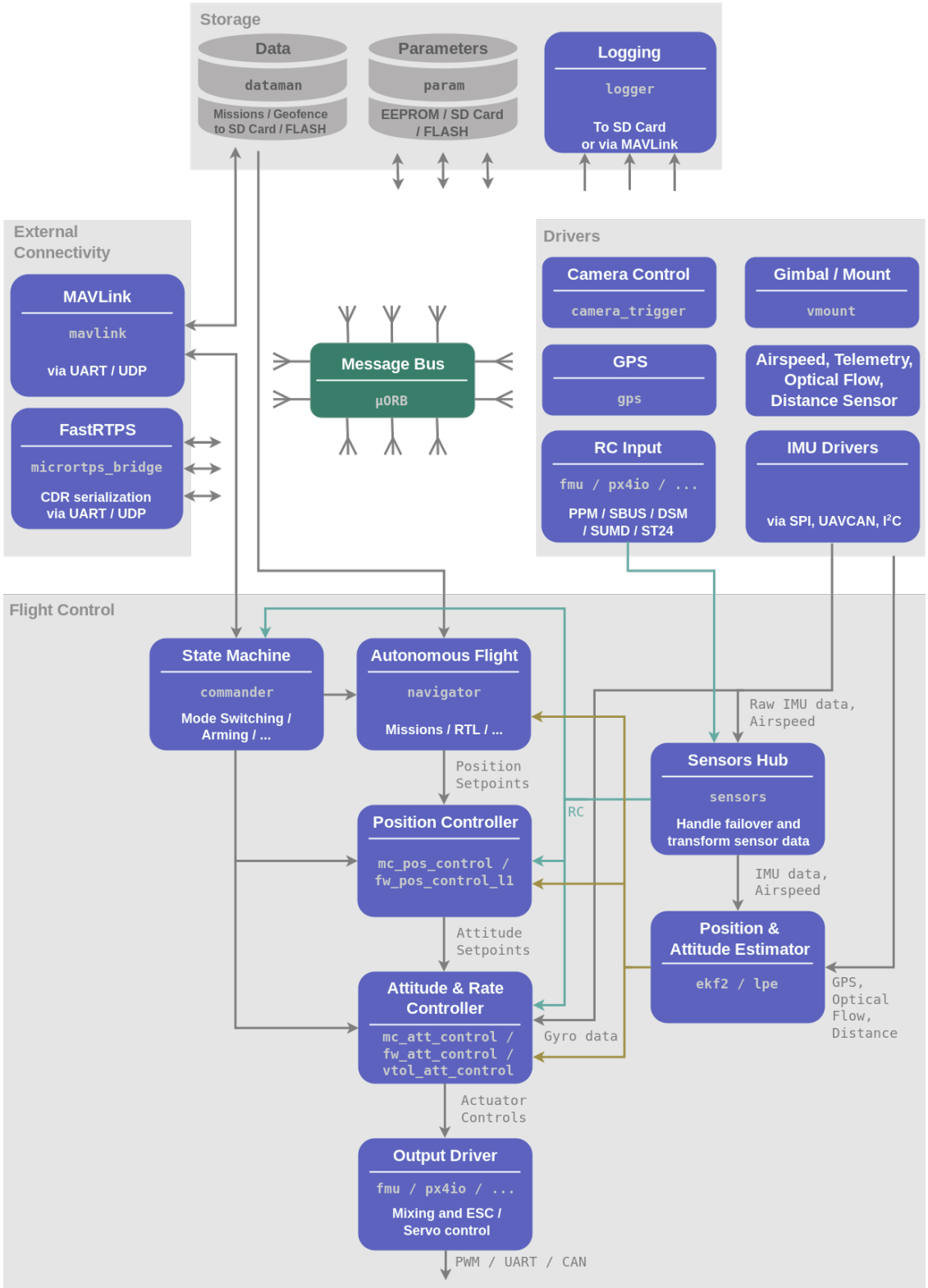


Figura 3.1: Architettura del codice di PX4 Autopilot

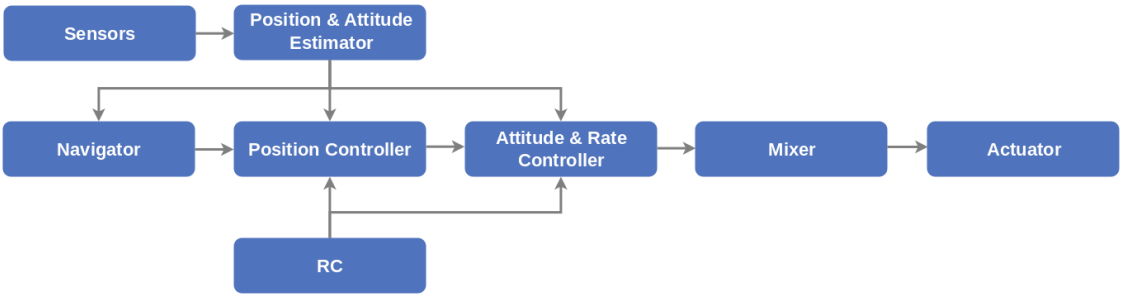


Figura 3.2: Architettura del flight stack di PX4

3.3 Modello matematico

3.4 Leggi di controllo

3.4.1 Controllore PID

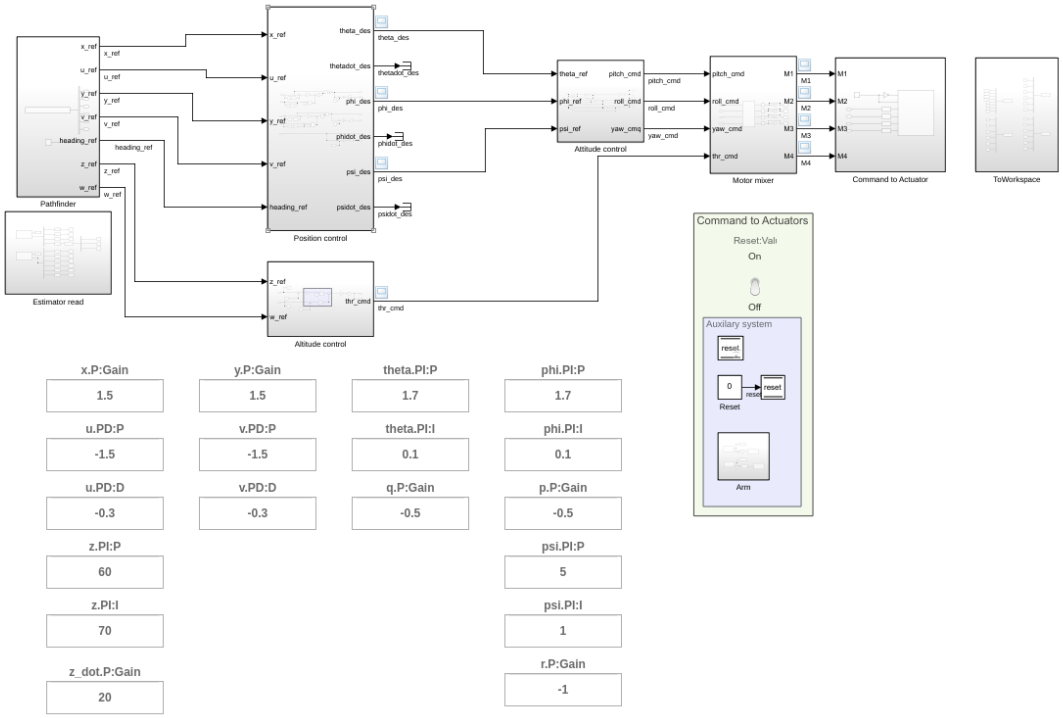


Figura 3.3: Modello di controllo completo PID

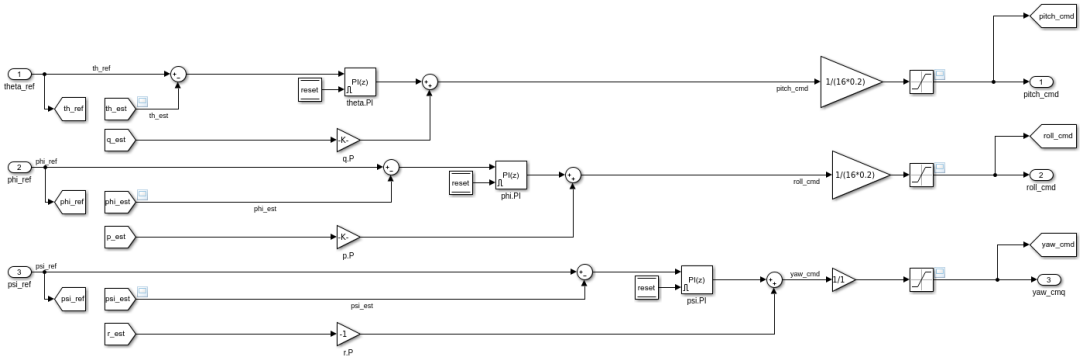


Figura 3.4: Modello di controllo d' assetto PID

Inserire le tabelle con i parametri

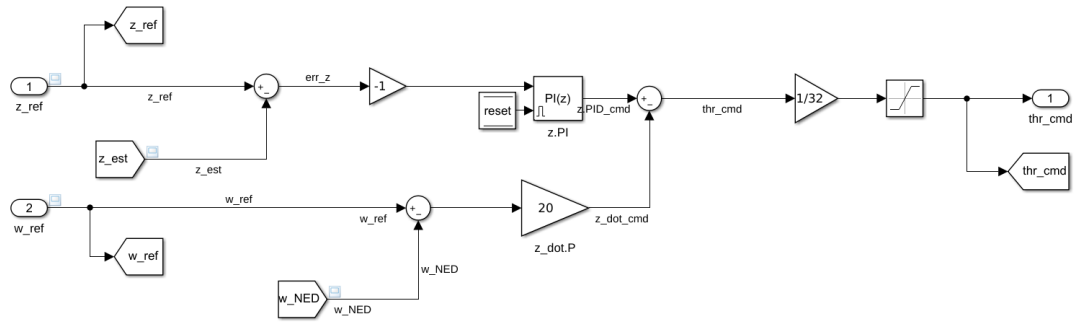


Figura 3.5: Modello di controllo di altitudine PID

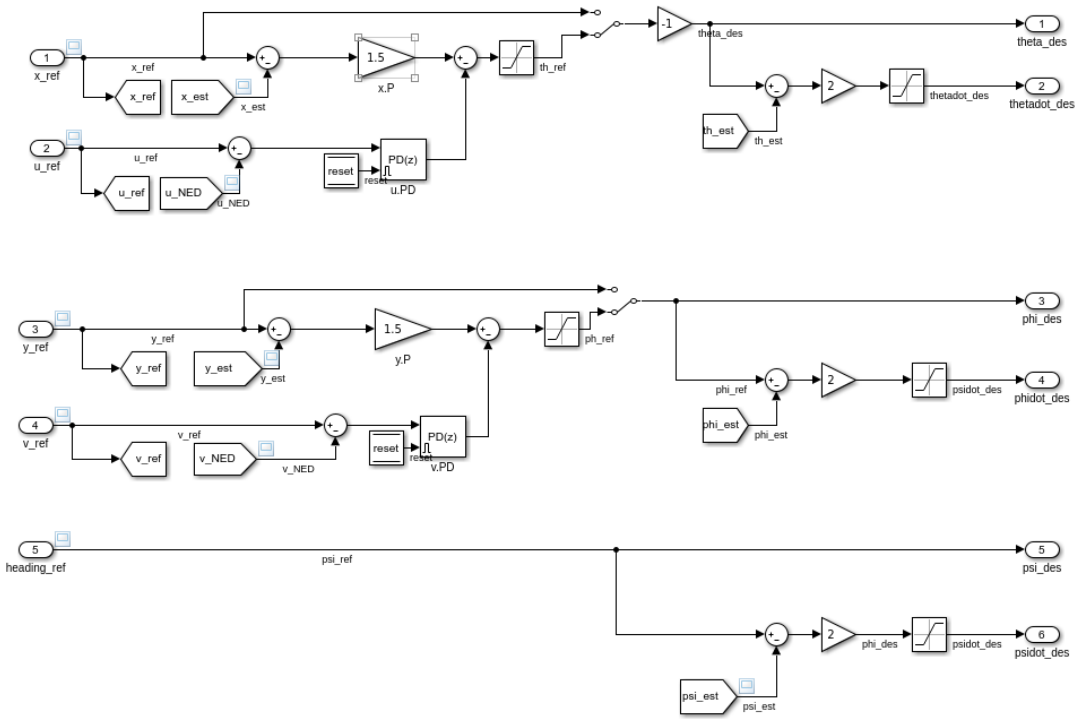


Figura 3.6: Modello di controllo di posizione PID

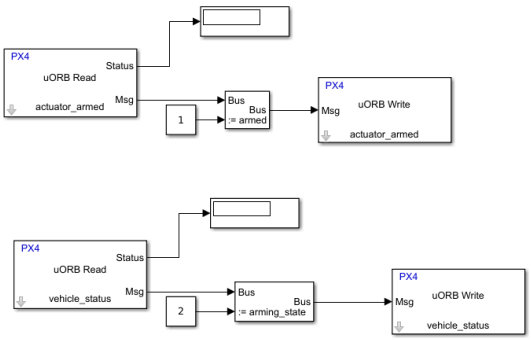


Figura 3.7: Modello di gestione dei segnali di armamento

3.5 Algoritmo di guida

Qual é l'algoritmo di guida?

Capitolo 4

Simulazioni

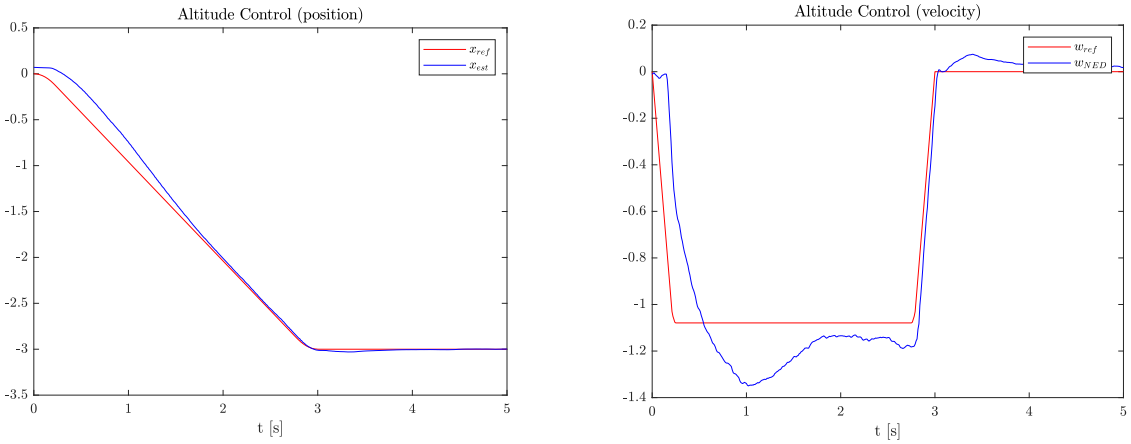
4.1 Simulazione SITL

```
#File : sil.h
#!/bin/bash

export GAZEBO_MODEL_DATABASE_URI=""

sim=$1

Firmware=$(realpath .)

cp posix-configs/SITL/init/ekf2/iris posix-configs/SITL/init/ekf2/sim

if [ "$#" -lt 1 ]
then
   echo Specificare il simulatore : jmavsim , gazebo
   exit 1
fi

if [ "$sim" == "gazebo" ]
then
   cd build/posix_sitl_default/build_gazebo
if [ "$(ls -A .)" ]
then
   echo "Makefile gia' presente"
else
   echo "Creazione makefile"
   cmake $Firmware/Tools/sitl_gazebo
fi
   echo "Esecuzione makefile"
   make
   cd $Firmware/build/posix_sitl_default
else
   cd $Firmware/build/posix_sitl_default
fi

$Firmware/Tools/sitl_run.sh $Firmware/build/posix_sitl_default/px4
    posix-configs/SITL/init/ekf2 none $sim sim $Firmware
    $Firmware/build/posix_sitl_default
```

4.1.1 PID

STEP

Tabella dei waypoints



(a) Controllo posizione

(b) Controllo velocità

Figura 4.1: Risposta del controllore PID di quota al segnale STEP

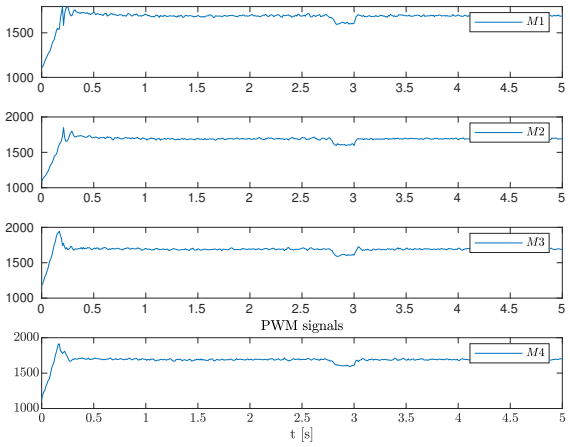
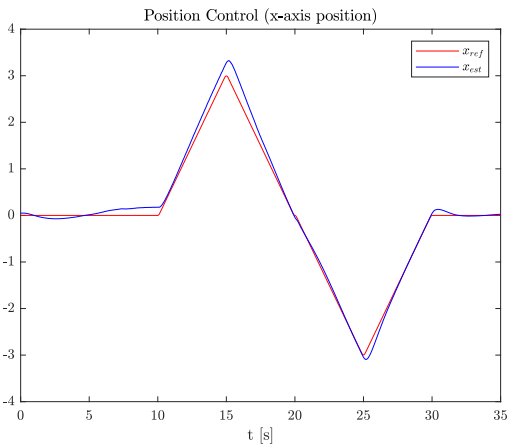
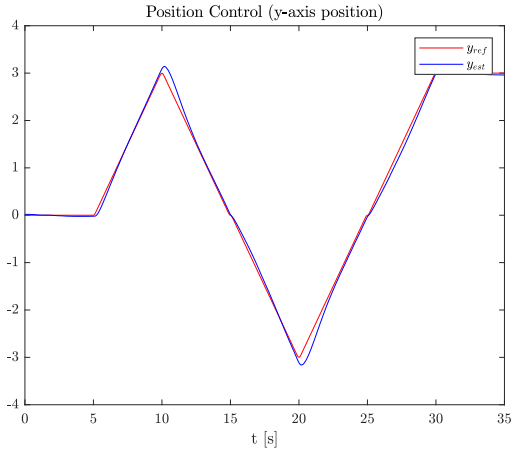


Figura 4.2: Segnali PWM del controllore PID al segnale STEP

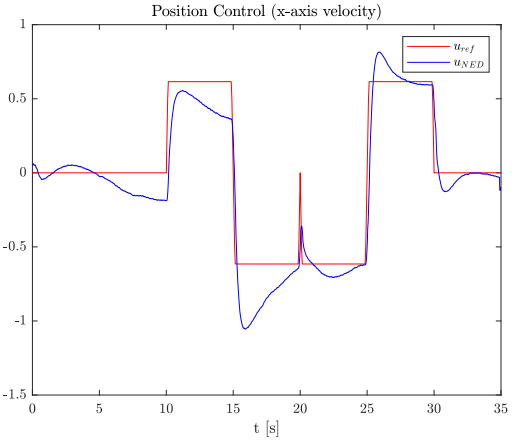
SQUARE



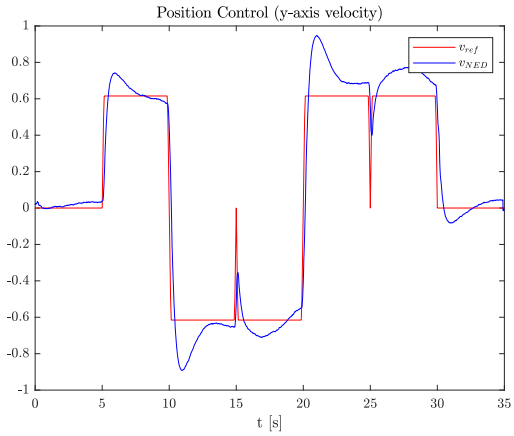
(a) Controllo posizione lungo x



(b) Controllo posizione lungo y

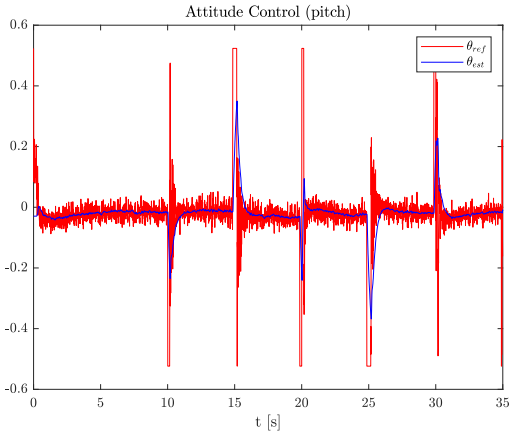


(c) Controllo velocità lungo x

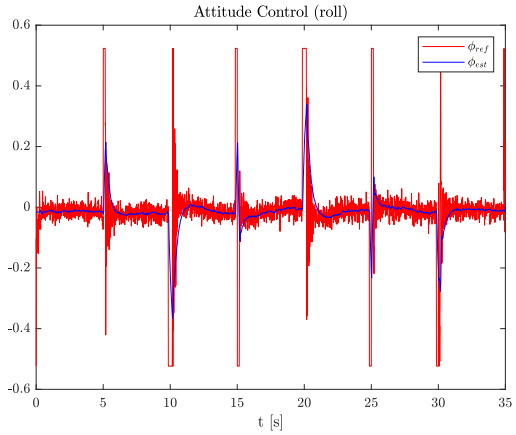


(d) Controllo velocità lungo y

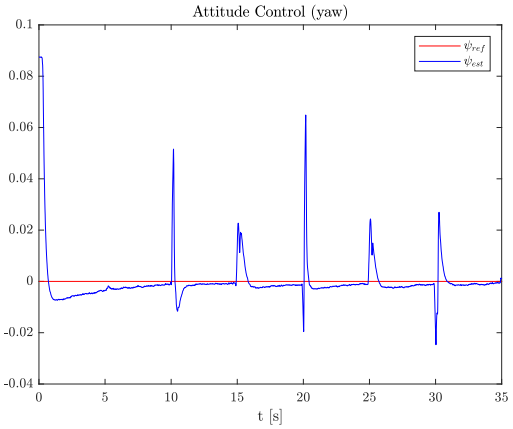
Figura 4.3: Risposta in posizione con controllore interno PID al comando SQUARE



(a) Controllo beccheggio



(b) Controllo rollio



(c) Controllo imbardata

Figura 4.4: Risposta dell' assetto con controllore interno PID al comando SQUARE

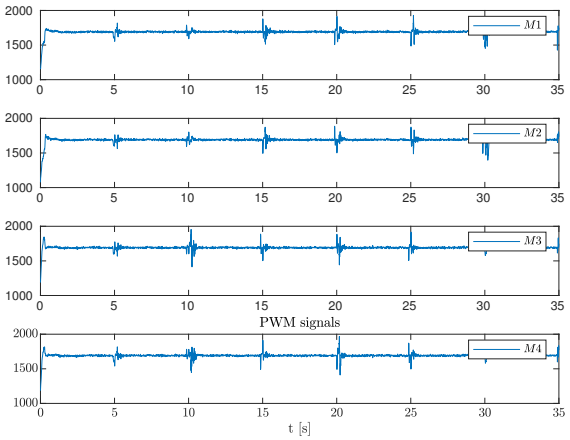


Figura 4.5: Segnali PWM del controllore PID al segnale SQUARE

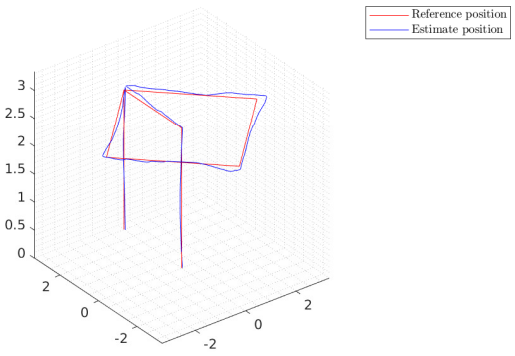
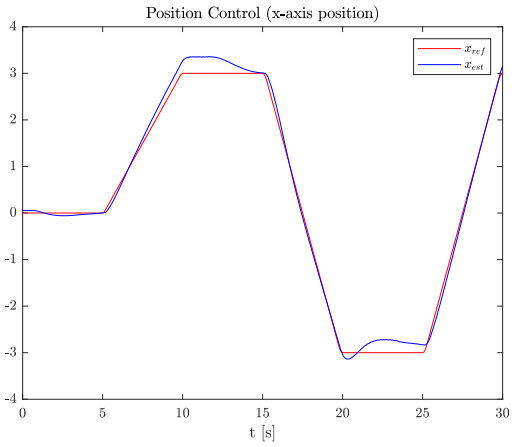
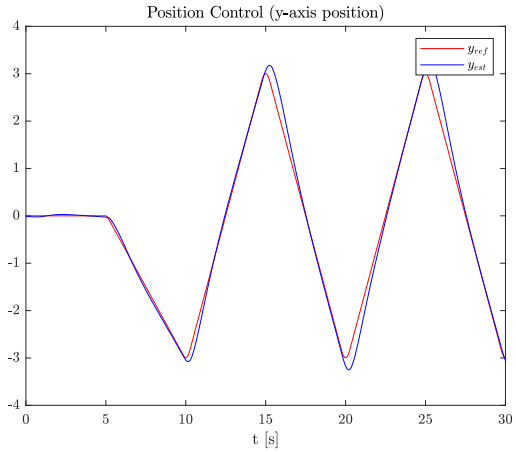


Figura 4.6: Traiettoria percorsa con controllore PID al segnale SQUARE

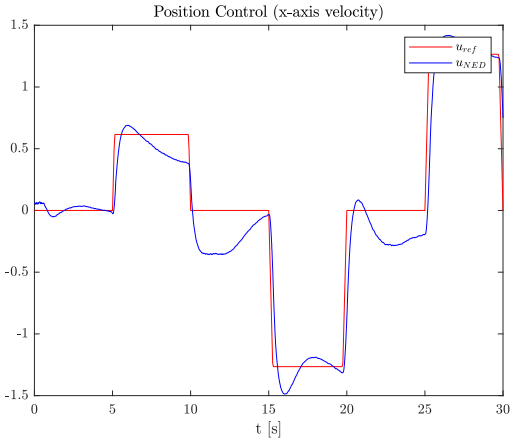
BUTTERFLY



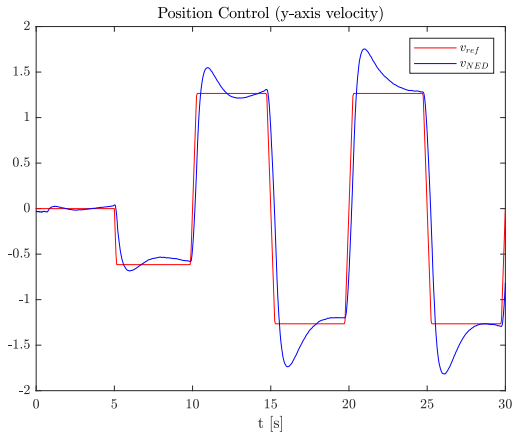
(a) Controllo posizione lungo x



(b) Controllo posizione lungo y

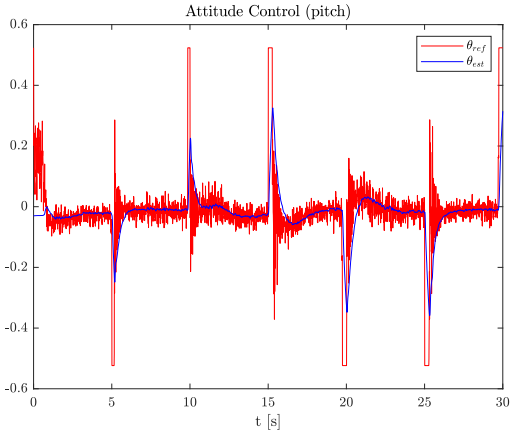


(c) Controllo velocità lungo x

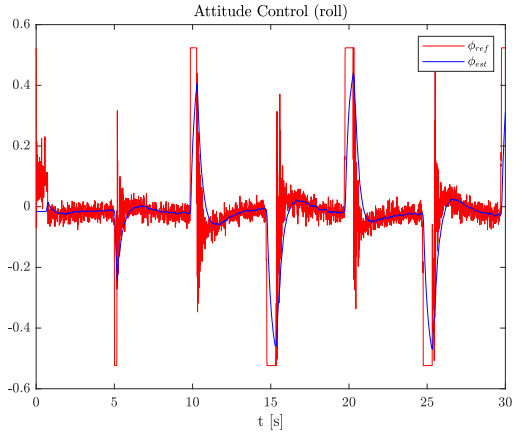


(d) Controllo velocità lungo y

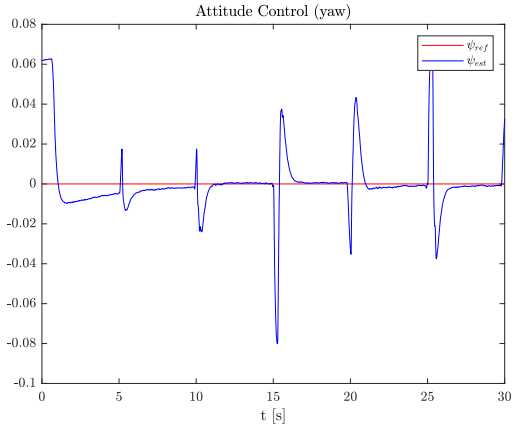
Figura 4.7: Risposta in posizione con controllore interno PID al comando BUTTERFLY



(a) Controllo beccheggio



(b) Controllo rollio



(c) Controllo imbardata

Figura 4.8: Risposta dell' assetto con controllore interno PID al comando BUTTERFLY

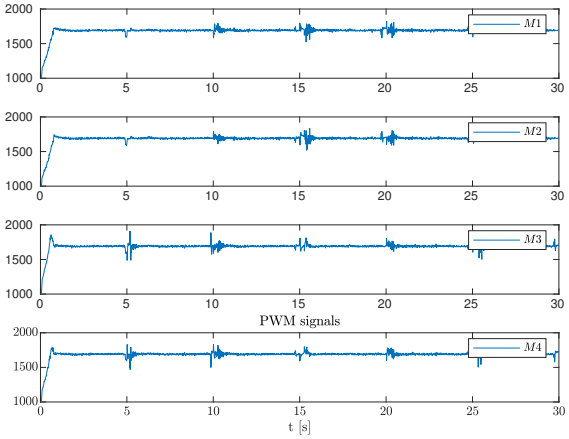


Figura 4.9: Segnali PWM del controllore PID al segnale BUTTERFLY

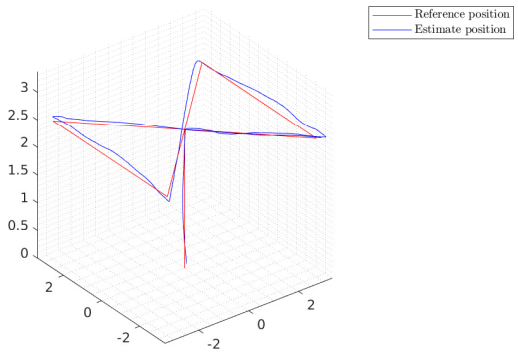
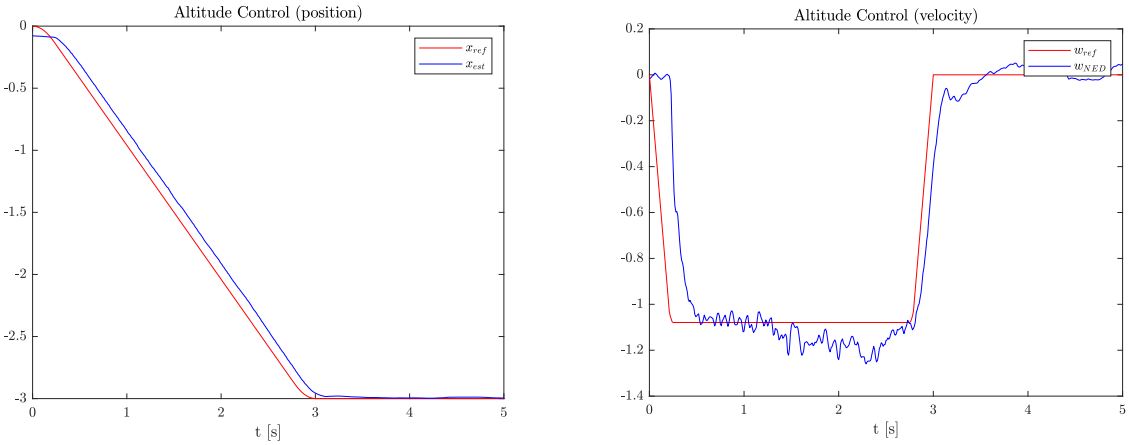


Figura 4.10: Traiettoria percorsa con controllore PID al segnale SQUARE

4.1.2 SMC

STEP

Tabella dei waypoints



(a) Controllo posizione (b) Controllo velocità

Figura 4.11: Risposta del controllore SMC di quota al segnale STEP

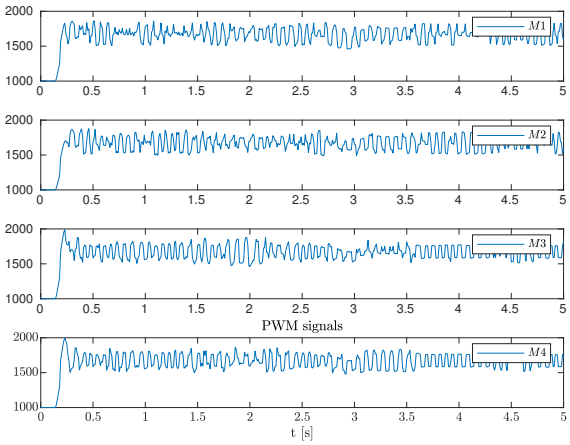
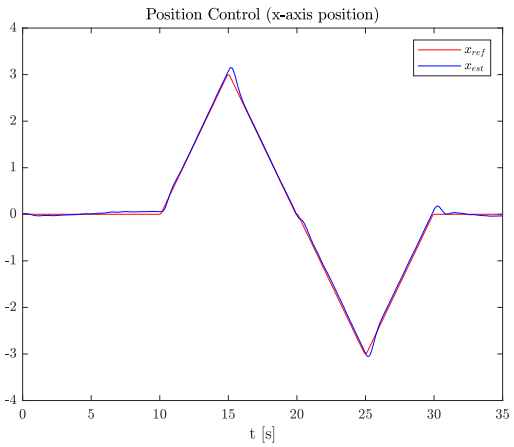
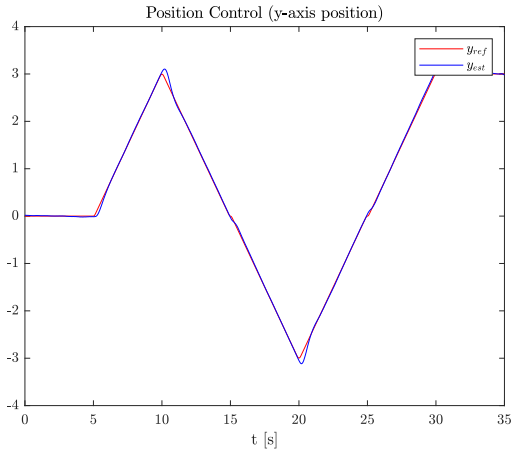


Figura 4.12: Segnali PWM generati del controllore SMC al segnale STEP

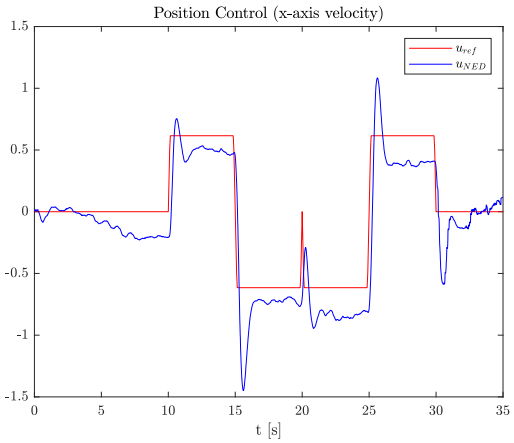
SQUARE



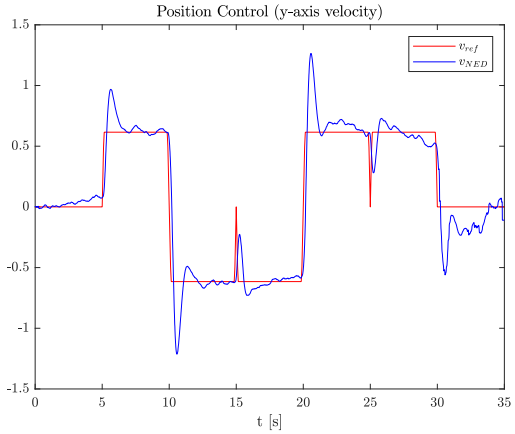
(a) Controllo posizione lungo x



(b) Controllo posizione lungo y

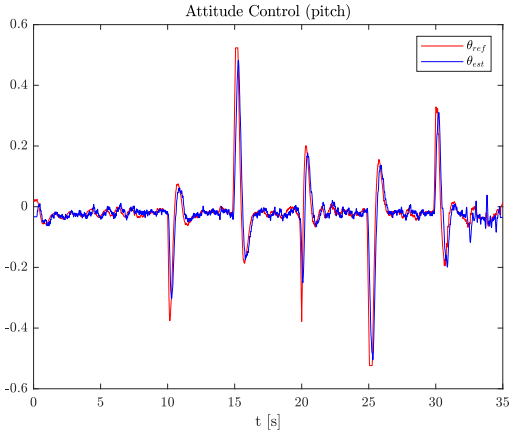


(c) Controllo velocità lungo x

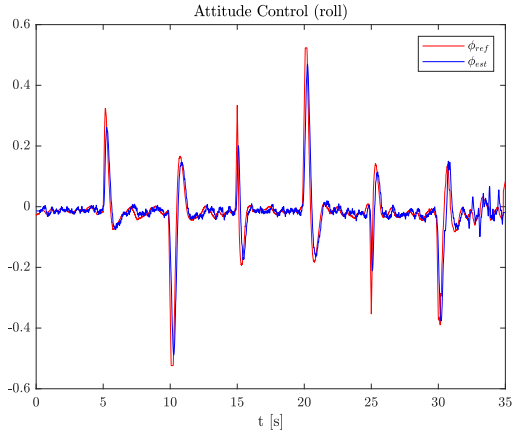


(d) Controllo velocità lungo y

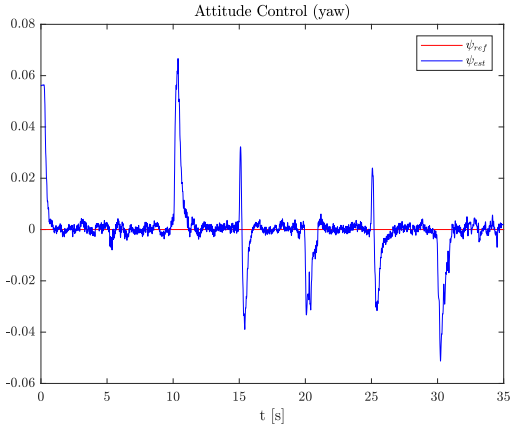
Figura 4.13: Risposta del controllo posizione con controllore SMC al comando SQUARE



(a) Controllo beccheggio



(b) Controllo rollio



(c) Controllo imbardata

Figura 4.14: Risposta dell' assetto con controllore interno SMC al comando SQUARE

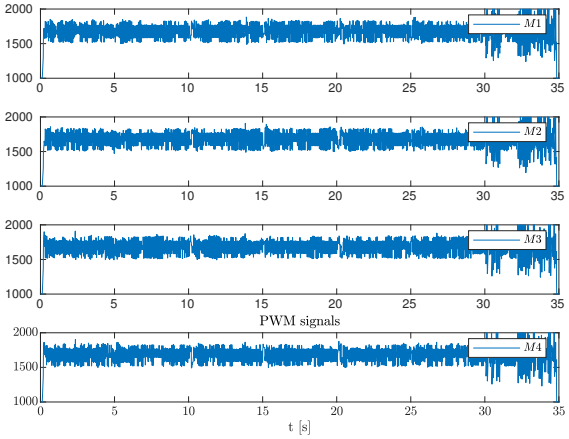


Figura 4.15: Segnali PWM del controllore SMC al segnale SQUARE

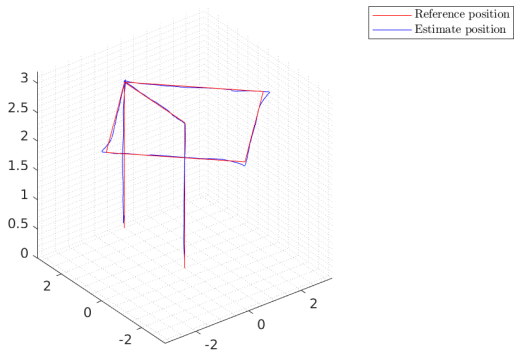
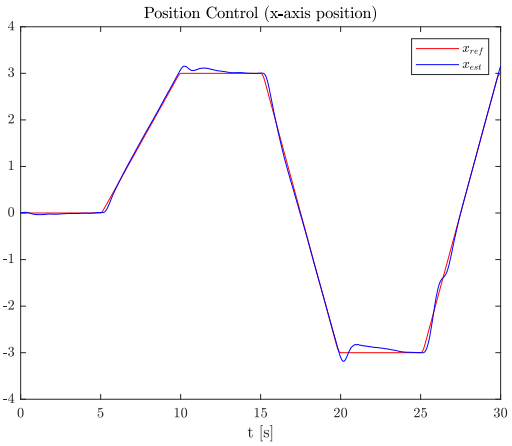
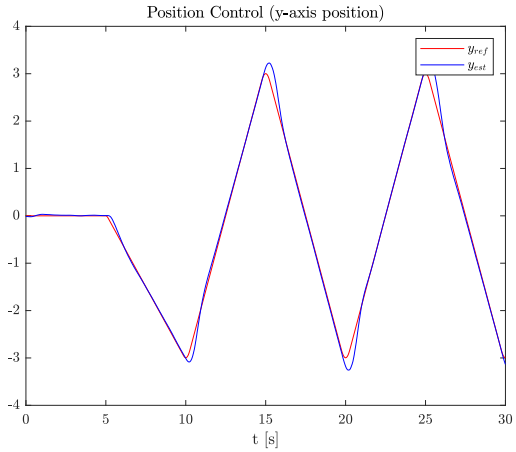


Figura 4.16: Traiettoria percorsa con controllore SMC al segnale SQUARE

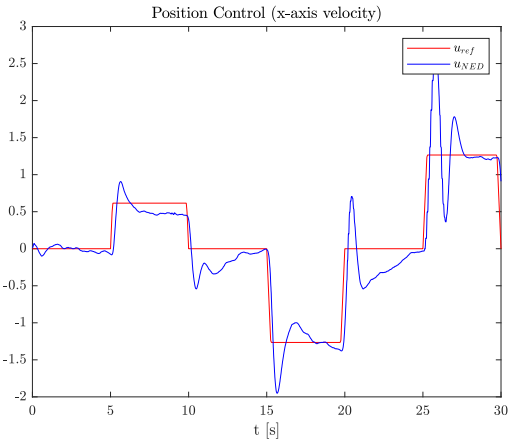
BUTTERFLY



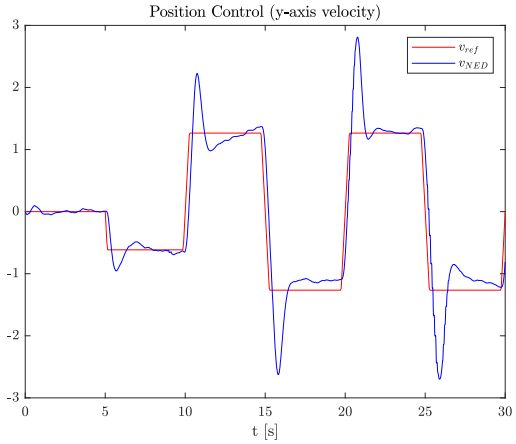
(a) Controllo posizione lungo x



(b) Controllo posizione lungo y

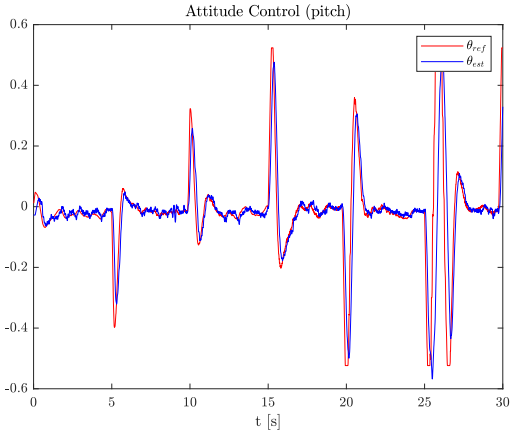


(c) Controllo velocità lungo x

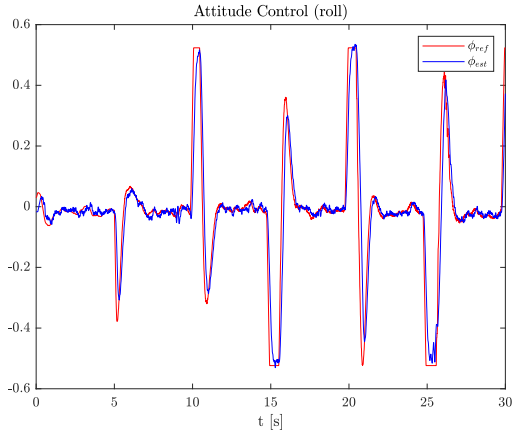


(d) Controllo velocità lungo y

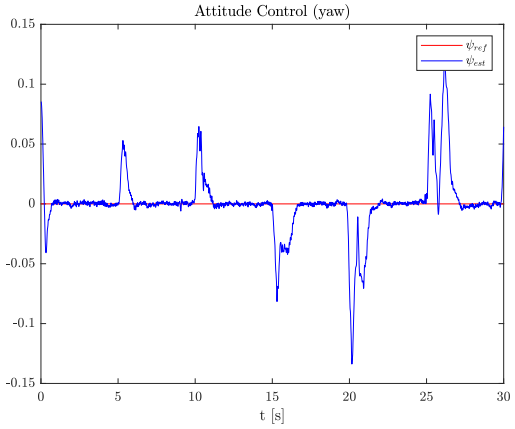
Figura 4.17: Risposta in posizione con controllore interno SMC al comando BUTTERFLY



(a) Controllo beccheggio



(b) Controllo rollio



(c) Controllo imbardata

Figura 4.18: Risposta dell' assetto con controllore interno SMC al comando BUTTERFLY

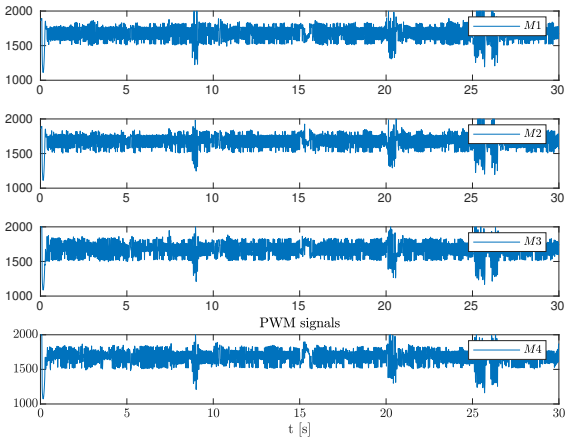


Figura 4.19: Segnali PWM del controllore SMC al segnale BUTTERFLY

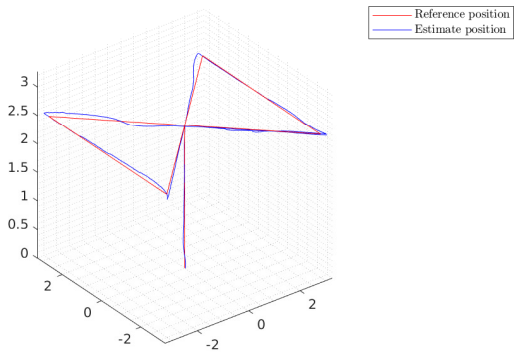


Figura 4.20: Traiettoria percorsa con controllore SMC al segnale SQUARE

4.1.3 Confronto

4.2 Conclusioni

Bibliografia

[1] Davide Carminati. «Design and Testing of Indoor UAS Control Techniques». Politecnico di Torino, 2019.

[2] *MAVLink Developer Guide*. Dronecode Project. 2020. URL: https://mavlink.io/en/ (visitato il 30/03/2020).

[3] *NuttX*. Wikipedia. 2020. URL: https://it.wikipedia.org/wiki/NuttX (visitato il 30/03/2020).

[4] *PX4 Autopilot User Guide (1.8.0)*. PX4 Dev Team. 2020. URL: https://docs.px4.io/v1.8.0/en/#px4-autopilot-user-guide--180 (visitato il 30/03/2020).

[5] *PX4 Development Guide (v1.8.0)*. PX4 Dev Team. 2020. URL: https://dev.px4.io/v1.8.0/en/index.html#px4-development-guide-v180 (visitato il 30/03/2020).

[6] Inc. The MathWorks. *PX4 Autopilots Support from Embedded Coder*. 2020. URL: https://it.mathworks.com/hardware-support/px4-autopilots.html (visitato il 30/03/2020).