POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di laurea magistrale

Sviluppo di un modello per aerogeneratore ad alta quota

POLITECNICO
DI TORINO

Relatore:
Paolo Maggiore
Matteo Davide Lorenzo Dalla Vedova

Tutor Aziendale:
Ing. Riccardo Cortese

Candidato:
Gianluca Gregnol

Ottobre 2019

Indice

Elenco delle tabelle

Elenco delle figure

Sommario

L'elaborato in oggetto propone la descrizione degli step adottati per la preparazione di un test di volo di un aeromodello, collegato per mezzo di un filo ad un macchinario, volto a produrre energia elettrica sfruttando il vento. Nella prima parte dell'elaborato viene descritto brevemente quali siano le attuali tecnologie per la produzione di energia elettrica da fonti rinnovabili, ed in particolare quella eolica. Quindi si mostreranno le motivazioni che hanno portato alla nascita di progetti come quello in oggetto a questa tesi. Si descrivera' dunque il primo step di lavoro, relativo al controllo da terra dell'aeromodello attraverso scheda Arduino interfacciandosi con il radiocomando. Si procedera' alla descrizione del codice usato per la ricezione di telemetria ed invio di comandi all'aeromodello, per mezzo di una scheda Arduino e del protocollo MAVLink, senza l'ausilio del radiocomando. Infine verra' presentato un primo modello di volo semplificato 6 DOF per l'aeromodello in oggetto, il cui sviluppo dovra' essere proseguito per portarlo alla sua forma finale.

Il test di volo dovra' andare a testare la manovra di riferimento per la generazione di energia, nonche' permettere di esaminare i comandi eseguiti dal pilota al fine di confrontare la risposta del velivolo reale con il modello del simulatore. Dal confronto di questi dati sara' possibile anche valutare come migliorare la manovra per ottimizzarla al meglio, portando quindi all'aumento dell'efficienza del ciclo di lavoro, e dunque alla produzione netta di energia.

Capitolo 1

Energia rinnovabile - il vento

Si definisce energia rinnovabile quella raccolta da risorse rinnovabili, ovvero risorse che in un certo periodo temporale vengono naturalmente reintegrate. Un esempio di energia rinnovabile e' la luce solare, le maree, il calore geotermico e il vento.

Il vento e' una fonte di energia in media stabile di anno in anno, ma molto variabile in scale temporali piu' ridotte. Ad oggi la maggior parte, se non tutta la produzione di energia elettrica proveniente dalla tecnologia legata al campo eolico proviene da venti molto vicini alla superficie terreste.

Tale energia viene in gran parte prodotta da impianti eolici costituiti da aerogeneratori chiamati turbine eoliche.

Queste sono caratterizzate dalla presenza di un rotore che, ruotando grazie all'energia del vento, permette appunto la produzione di energia elettrica.

Figura 1.1: Tipica struttura di una turbina eolica

1.1 I parchi eolici

Tralasciando gli aspetti costruttivi di questi aerogeneratori, queste turbine vengono installate generalmente in cosidetti parchi eolici ("wind farm" in inglese), ovvero degli agglomerati di turbine eoliche situati in una regione circoscritta. Essendo queste delle strutture molto alte, rumorose ed ingombranti, non posso essere installate ovunque. Si contraddistinguono essenzialmente tre tipi di installazioni di parchi eolici:

- Parchi eolici On-shore: questo e' il metodo usato per la maggior parte per ragioni storiche e tecnologiche. Infatti la caratteristica di tali parchi e' di avere installazioni almeno distanti 3 km dalla costa piu' vicina, tipicamente su colline, alture o comunque in zone aperte e ventose. Tale tipologia di parco puo' generare da 20 kW a 20 MW di potenza e puo' essere collegato alla rete pubblica o per portare energia in aree isolate.

Figura 1.2: tipico esempio di parco eolico on-shore

- Parchi eolici Near-shore: sono parchi situati a meno di 3 km dalla costa, tipicamente nell'entroterra o nel mare, ma con una distanza che non ecceda i 10 km dalla costa. In termini di produzione, i parchi nell'entroterra generano potenze simili al caso precedente, mentre per le installazioni marittime si possono generare potenze nell'odrine del MW.

- Parchi eolici Off-shore: sono impianti installati ad alcuni chilometri di distanza dalla costa marittima o di laghi, per un utilizzo piu' efficiente delle forti correnti tipiche di queste aree. Alcune di queste turbine posso essere fluttuanti, perche' operanti su fondali molto profondi. Ad Havsui, in Norvegia, sorgerà il più grande impianto eolico off-shore al mondo, che potrà fornire 1,5GW di potenza elettrica.

Figura 1.3: tipico esempio di parco eolico off-shore

1.2 Limiti della turbina eolica

I limiti di questa tecnologia tuttavia sono notevoli ed abbastanza ingombranti.

Partendo dal primo limite, l'efficienza massima degli impianti eolici, questa può essere calcolata mediante la Legge di Betz che afferma e dimostra che l'energia massima che può essere prodotta da un generatore corrisponde al 59,3% dell'energia prodotta dal vento che lo attraversa.

Il secondo limite e' puramente tecnologico e dettato dalla lunghezza delle pale, dall'altezza della torre e dai meccanismi di accensione. Ogni sistema infatti e' caratterizzato dalla velocita' di "cut-in", cioe' la minima velocita' del vento per cui il sistema non si accende finche' non viene raggiunta.

Un altro limite e' dettato dall'impatto ambientale, visivo ed ecologico, che questi parchi hanno. Essendo costruzioni molto invadenti e rumorose, come gia' scritto sopra, la costruzione e' limitata a zone non abitate. A livello ecologico, l'energia consumata per produrre e trasportare i materiali usati per la costruzione di un impianto e' uguale a quella prodotta dallo stesso nei primi mesi di attivita', mentre quella prodotta e' equivalente a $70-75$ volte quella necessaria alla sua costruzione.

Vi sono poi altri limiti di queste turbine legati alla propagazione delle telecomunicazioni e al disturbo della fauna, in quanto vengono percepite come minacce da parte di volatili.

Considerata questa breve analisi sugli impianti eolici tradizionali che sfruttano il vento presente a quote molto basse, vediamo quale sia una delle alternative, ovvero lo sfruttamento del vento ad alta quota.

1.3 Il progetto Kitenergy

Con aerogeneratore per venti ad alta quota si intende una struttura sprovvista di torre di sostegno che possa beneficiare della maggiore velocita' dei venti presenti a quote piu' alte, svincolandosi quindi dai costi di costruzione ed ingombro delle torri e dei cinematismi necessari per le parti rotanti.

Il vincolo ora diventa un cavo connesso a terra che puo' essere usato da una macchina per produrre elettricita' o per portare l'elettricita' prodotta a bordo velivolo a terra.

La potenza prodotta utilizzando questa tecnologia puo' essere espressa mediante la formula di Lloyd. Il progetto di Kitenergy ha sviluppato la sua idea basandosi su questa prima opzione. Si fa uso infatti di un kite o di un'ala costituita di materiale tessile, fissata ad una coppia di cavi ancorati ad una macchina a terra, per srotolare un cavo e produrre energia.

Il cavo ha una doppia funzionalita' in tal caso: controllare il kite e trasmettere la trazione. In tal modo si riduce notevolmente la dimensione della macchina globale e risulta piu' semplice intercettare il vento.

La KSU (Kite Steering Unit) e' l'unita di controllo di terra, composta da due attuatori elettrici capaci di lavorare sia come motori che come generatori elettrici, a cui e' collegato il kite. In base ai dati che il sistema ricevere dai diversi sensori (direzione del vento, velocita', coordinate del kite, etc) tale sistema controlla la traiettoria del kite per massimizzare la generazione di energia.

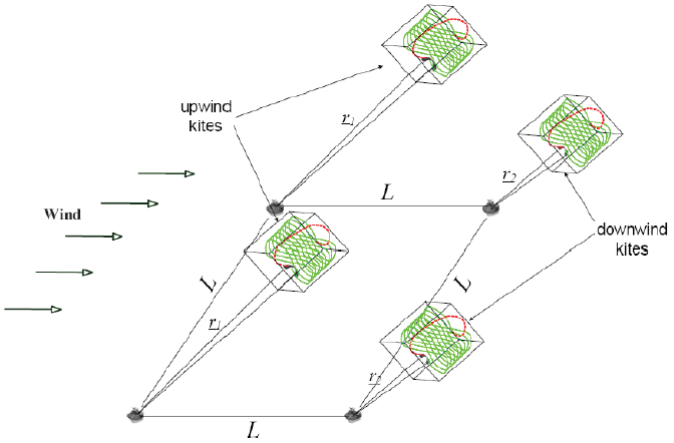

Figura 1.4: Esempio di manovra con l'uso di un kite

Il kite inizialmente viene rilasciato da terra e raggiunge poi la quota operativa. La manovra che questo sviluppa nella fase di generazione di energia e' tipicamente a forma di "8" e permette di srotolare i cavi. Terminato lo srotolamento, il kite entra nella fase di recupero, che permette di arrotolare i cavi, e per fare cio' si porta in una posizione di minima resistenza. Tale fase e' uno dei maggiori problemi per la produzione di energia nel ciclo poiche' non richiede soltanto di spendere parte dell'energia prodotta, ma la manovra deve essere svolta in maniera molto lenta al fine di ridurre le richieste di potenza complessive al sistema. Cio' implica che tale fase occupi dal 20% al 50% del ciclo totale, a seconda che vi sia la presenza di venti piu' leggeri o pesanti rispettivamente.

Visto tale problema nella fase di recupero, l'idea successiva e' stata quella di capire se un aerogeneratore ad ala fissa potesse permettere di ricavare gli stessi livelli di energia o addirittura maggiori. Questo perche' nella fase di recupero un velivolo di questo tipo sarebbe in grado di orientarsi molto rapidamente verso la condizione di minima resistenza e sarebbe piu' controllabile di un kite, portando ad una riduzione dei tempi del recupero e quindi ad un minor consumo di energia, aumentando l'efficienza globale della macchina.

L'utilizzo di un velivolo ad ala fissa consente un minor uso di cavi in quanto le superfici di controllo sono attuate mediante attuatori dedicati.

Capitolo 2

Il velivolo

Figura 2.1: skywalker X8

Lo Skywalker X8 e' un velivolo tutt'ala dotato di una unica coppia di elevoni come superficie di controllo. Gli elevoni hanno una doppia funzione: se attivati nella stessa direzione operano come comando di controllo del beccheggio, se vengono attivati in direzione opposta l'uno all'altro operano come comando di controllo del rollio.

Il velivolo e' dotato di una piccola baia nella quale e' possibile riporre tutta la strumentazione elettronica utile al volo, quale batteria, controllore, GPS, etc.

Figura 2.2: baia avionica del velivolo

Si riportano di seguito le principali caratteristiche geometriche del velivolo:

- Superficie alare [S]: 0.81 m^2
- Corda alare MAC [c]: 0.34 m
- Apertura alare [b]:2.16 m
- Massa (con strumentazione) [m]: 5.85 kg

Essendo questo un velivolo reperibile commercialmente, diversi studi sono stati fatti in termini aerodinamici.

Uno di questi ha permesso di caratterizzare il velivolo in termini di coefficienti aerodinamici, anche se l'accuratezza di tali risultati e' sicuramente un fattore da non trascurare. Per ulteriori approfondimenti si veda il capitolo relativo al simulatore di volo.

Capitolo 3

La strumentazione

3.1 Taranis X9D Plus

3.1.1 Caratteristiche generali

FrSky Taranis X9D Plus è il radiocomando scelto per il controllo del velivolo e su cui si sono basati i codici creati, le cui funzioni verranno esposte nelle sezioni successive. Tale radiocomando e' dotato di firmware open-source, il che lo rende programmabile da esperti del settore che vogliano personalizzare le funzionalita' dello stesso.

Figura 3.1: il radiocomando Taranis X9D Plus

Le caratteristiche principali di questo modello sono qui di seguito elencate:

- Telemetria completa di allarmi RSSI
- Controllo del ROS (rapporto onde stazionarie) dell'antenna del trasmettitore

- 16 canali (fino a 32 in combinazione con un modulo esterno)
- 60 Memorie (infiniti modelli memorizzabili su SD)
- 64 mixer, 9 fasi di volo
- 32 interruttori logici
- 32 funzioni personalizzabili
- Porta USB
- Bassa latenza (9 *ms*)
- Log dei dati in real time su scheda SD
- Modalità di volo selezionabili
- Porta maestro allievo
- Completa integrazione in Companion9X (un programma opensource per la programmazione, il backup, il controllo e la simulazione della radio)
- Software opensource

Di particolare interesse, come si avra' modo di leggere nei seguenti capitoli, e' la possibilita' di avere una porta maestro allievo. Questa funzione generalmente consente la connessione fisica di due radiocomandi. In tal modo, uno di questi sara' "l'allievo" e potra' controllare il velivolo, mentre l'altro sara' in modalita' "maestro" e avra' la possibilita' in ogni istante di escludere i comandi dati dall'allievo, prendendo direttamente il controllo del mezzo. Questa modalita' consentira' la connessione fisica del radiocomando con la scheda Arduino, la quale agira' in modalita' "allievo".

3.1.2 Pagina SERVI

Tra le svariate pagine di controllo del radiocomando, di particolare utilita' e' la pagina dedicata alla gestione dei servocomandi. Dalle impostazioni infatti e' possibile adattarli alle caratteristiche meccaniche del nostro modello.

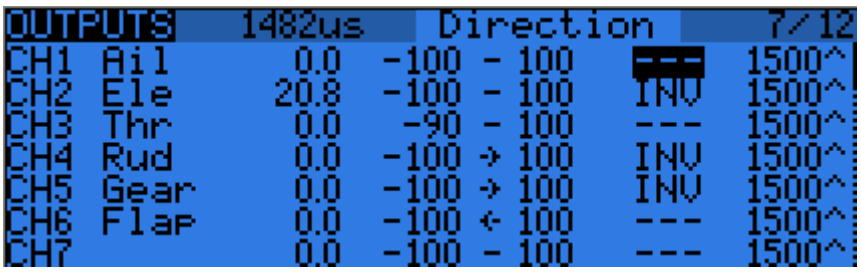

Figura 3.2: Illustrazione reale della pagina SERVI del radiocomando

Per ogni canale e' possibile infatti definire:

- offset o subtrim
- limiti di corsa in ambedue le direzioni. Questi sono limiti fisici che andranno a far si che il servocomando non "sforzi" oltre una certa posizione.
- inversione della corsa del servocomando
- regolazione del centro PPM, ovvero il valore PPM per cui si avra' escursione nulla della superficie.

Il PPM (pulse position modulation) è la tipologia di impulsi usati per comandare i servocomandi.

Nelle vecchie radio a modulazione FM o AM una serie di impulsi di larghezza variabile PWM (pulse width modulation) veniva trasmessa per comandare tutti i servocomandi collegati alle riceventi. La ricevente estraeva i singoli impulsi PWM dal "treno" e li inviava ai singoli servocomandi.

Gli impulsi trasmessi hanno normalmente il centro a 1500 μs e un'escursione di $+/-500$ μs.

3.2 Arduino UNO

3.2.1 Caratteristiche generali

Arduino Uno e' una scheda con microcontrollore basata su ATmega328P.

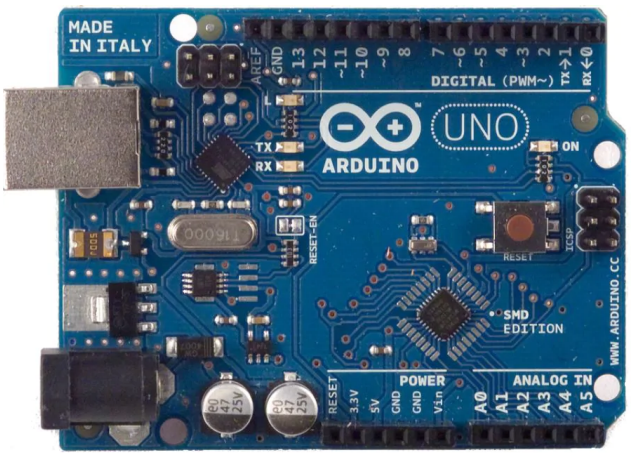

Figura 3.3: Scheda Arduino Uno

Alcune delle caratteristiche principali di tale prodotto sono:

- 14 pin digitali di input/output (di cui 6 possono essere usati come output PWM)
- 6 input analogici

- Oscillatore al quarzo da 16 MHz
- connettore USB
- bottone di reset
- alimentazione 5 V (con regolatore lineare di tensione)

3.2.2 Software

L'ambiente di sviluppo integrato di Arduino, denominato IDE, e' un'applicazione scritta in Java. L'IDE permette la scrittura del codice, la sua compilazione e il caricamento sulla scheda a cui e' collegato. Inoltre include un editor di testo dotato di syntax hightlighting, controllo delle parentesi ed indentazione automatica.

L'IDE e' dotato di due importanti strumenti: un monitor seriale ed un plotter seriale.

Il monitor seriale permette al computer la lettura dei dati in uscita da Arduino attraverso la porta seriale COM. Un parametro di impostazione molto importante e' il boudrate, espresso in *bps* o boudrate per secondo. Questo strumento sara' molto importante per poter leggere ad esempio dati di telemetria, per permettere all'utente di inserire degli input di comando o semplicemente per leggere i risultati dei calcoli provenienti dal codice arduino che e' in esecuzione.

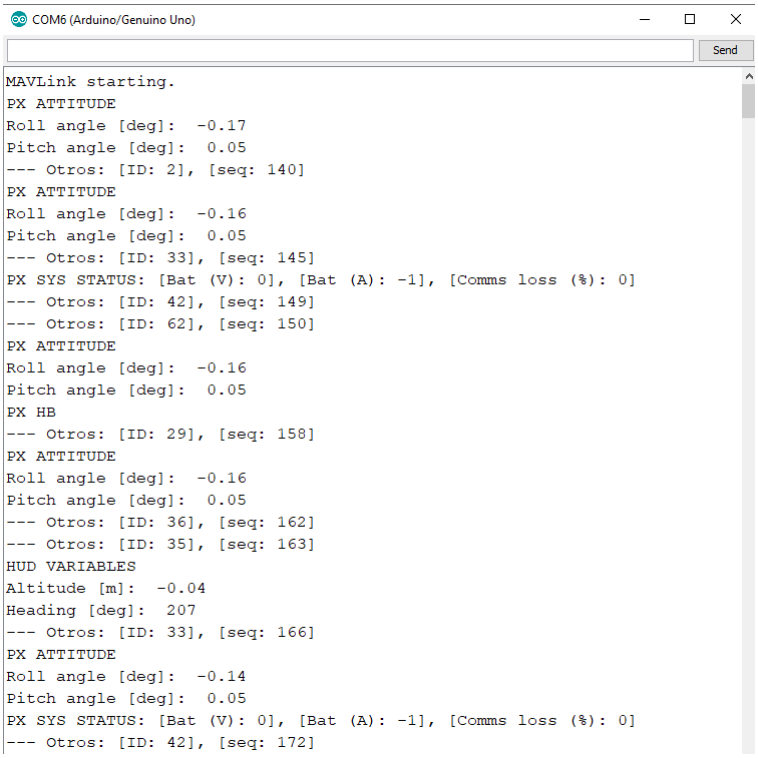

Figura 3.4: Esempio di informazioni stampate a video su monitor seriale

Il plotter seriale, esistente dalla versione 1.6.7 dell'IDE, permette la visualizzazione grafica di una o piu' variabili (o di qualsiasi funzione matematica) nel tempo. Per utilizzarlo e' necessario inviare le informazioni numeriche alla porta COM, che verranno quindi elaborate. Questo strumento sara' molto importante per la visualizzazione di segnali PPM o per visualizzare l'andamento di diversi parametri di volo.

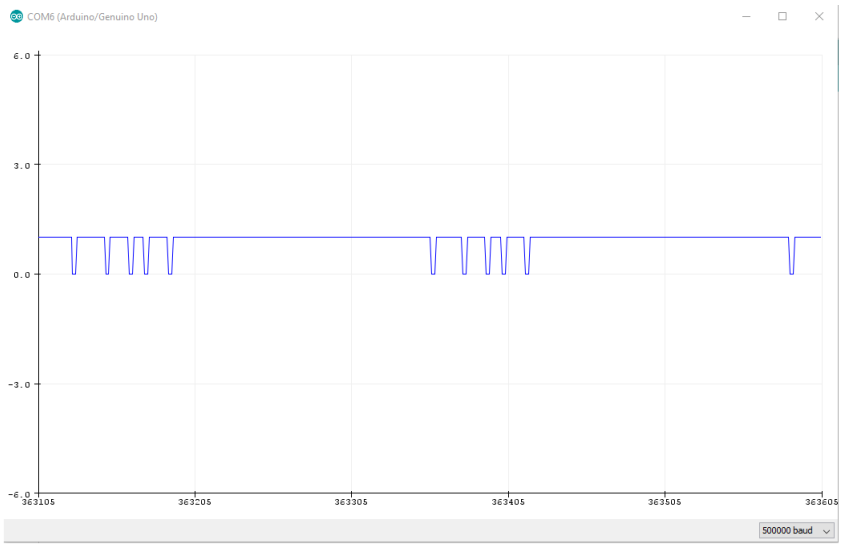

Figura 3.5: Esempio di segnale PPM riprodotto su plotter seriale

Da notare come non vi sia la possibilita', usando Arduino UNO, di poter visualizzare contemporaneamente i due strumenti di monitor seriale e plotter seriale, essendo questo dotato di un'unica porta seriale. Cio' puo' essere un fattore limitante in fase di test del codice, ma e' comunque un limite gestibile con opportuni accorgimenti.

3.3 Pixhawk 2.1 - The cube

Il controllore di volo Pixhawk 2.1, e' un sistema di autopilota avanzato, adattabile a mezzi di aria, terra ed acqua.

Infatti puo' essere usato al fine di controllare (attraverso l'opportuno firmware):

- velivoli ad ala fissa (tradizionali o VTOL)
- elicotteri
- multicotteri
- rover
- barche
- sottomarini

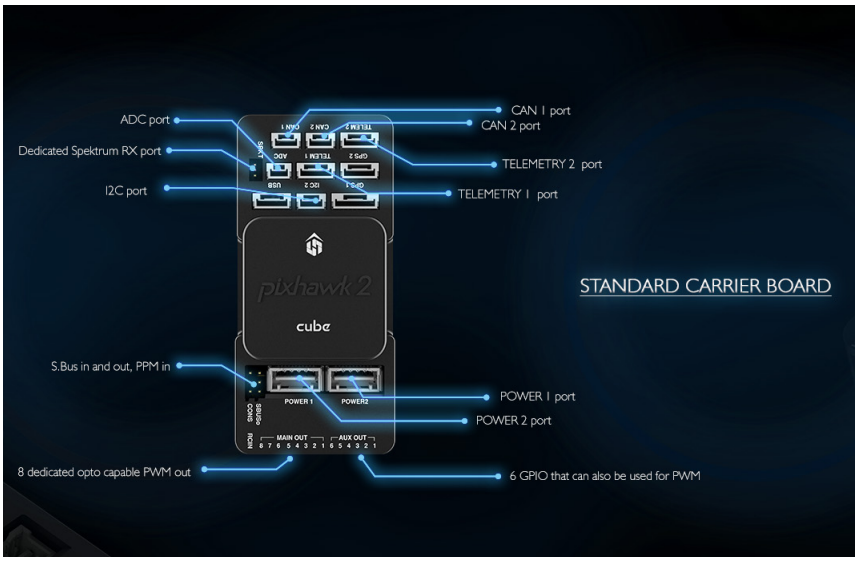

Figura 3.6: PX2 - the cube. Si trovano indicati i diversi I/O disponibili per questa versione

- altri componenti robotici

Tale controllore, detto anche "the cube" o "PX2" e' pensato e realizzato per funzionare con una scheda di base specifica per diminuire drasticamente il numero di cavi, migliorare l'affidabilita' e la facilita' d'uso. Tale componente include isolamento da vibrazioni su due delle tre IMU presenti nel dispositivo, mentre la terza IMU viene usata come backup.

Alcune carattestiche tecniche essenziali sono le seguenti:

→ Scheda principale

- STM32F427; flash 2 MiB, RAM 256 KiB.
- On-board 16 KiB SPI FRAM
- Accelerometro / giroscopio MPU9250 o ICM 20xxx
- Barometro MS5611
- Sensori connessi attraverso SPI
- Interfaccia Micro SD attraverso SDIO

→ Schede con smorzamento delle vibrazioni

- Accelerometro / giroscopio integrato LSM303D
- Giroscopio L3GD20
- Accelerometro / giroscopio MPU9250 o ICM 20xxx
- Barometro MS5611
- Sensori connessi attraverso SPI

→ Porte I/O

- 14 output PWM per servomotori (8 da IO, 6 da FMU)
- input R/C da CPPM, Spektrum / DSM e S.Bus
- input analogico / PWM RSSI
- 5 porte seriali per scopi generici, 2 con full flow control
- 3 input analogici
- switch / LED di sicurezza

Su tale dispositivo e' presente il firmware denominato Arduplane, per velivoli convenzionali. La configurazione dello stesso al fine di rendere il velivolo pronto all'uso e' stata cura dell'azienda ProS3 per mezzo del programma Mission Planner. La principale funzione che si utilizzera' di questo programma sara' la possibilita' di operare il download dei log di volo del velivolo una volta giunto a terra.

Attraverso il protocollo MAVLink sara' possibile far comunicare la scheda Arduino con questo componente, ricevendo dati di telemetria ed inviando comandi specifici (si veda l'apposito capitolo).

Capitolo 4

Generazione di comandi di volo con tecnica PPM

Per poter comunicare con il sistema di bordo del velivolo, la trasmittente Taranis invia un segnale di tipo PPM (Pulse Position Modulation).

In questo capitolo verra' illustrato come, con l'ausilio di Arduino, della trasmittente Taranis e della tecnica PPM, si siano comandate le superifici di volo del velivolo.

4.1 Il segnale PPM

Nella modulazione di posizione o pulse position modulation (PPM) viene variata, in funzione del segnale modulante, la posizione degli impulsi della portante, spostandoli in anticipo o in ritardo rispetto alla posizione periodica che hanno in assenza di modulazione. Larghezza ed ampiezza degli impulsi rimangono inalterate.

Questo tipo di modulazione permette o la divisione del canale in τ/T slot temporali (canali) oppure di mandare un messaggio codificato nel tempo e di inviare nel tempo T un numero τ di bit (sono possibili 2τ combinazioni temporali diverse).

Ogni canale ha un valore associato al servoattuatore (da ora in poi chiamati semplicemente servo) dettato dal periodo dell'onda quadra, espresso convenzionamente in $\mu\ s$.

Generalmente si trasmette un impulso di riferimento a cui valutare la posizione dell'impulso modulato, in modo da sincronizzare il ricevitore con la trasmittente.

Generalmente per i servoattuatori si hanno periodi d'onda (o impulsi) che variano da un minimo di 1000 $\mu\ s$ ad un massimo di 2000 $\mu\ s$. Al periodo minimo d'onda corrispondera' ad esempio l'escursione massima negativa della superficie di volo associata al servo, mentre al periodo massimo d'onda corrispondera' l'escursione positiva massima della superficie di volo associata al servo.

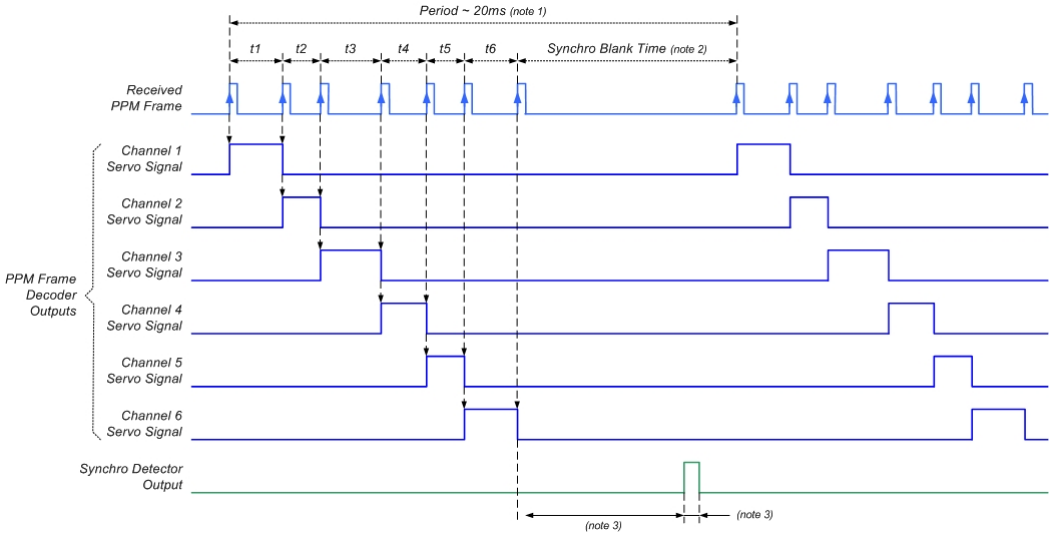

Figura 4.1: Esempio di segnale PPM

4.2 Generazione del segnale PPM con Arduino

Per generare un segnale PPM e' necessario che vi sia una certa frequenza di campionamento del segnale, quindi vi e' la necessita' di riuscire a creare un determinato numero di pacchetti di segnale (con pacchetto indichiamo una serie di periodi d'onda pari al numero di canali da gestire, piu' il segnale di riferimento). Tale paramentro viene definito come boudrate e si riferisce al numero di pacchetti inviati al secondo.

Dal momento che l'utilizzo di timer esterni ad Arduino avrebbe implicato una diminuzione della frequenza di campionamento del segnale PPM, si e' ricorsi all'utilizzo di timer interni allo stesso. Cosi facendo infatti sono stati utilizzati contatori gia' attivi nel microcontrollore, ottimizzando la velocita' di esecuzione complessiva dello script.

Per fare cio' ci si e' affidati ad un codice creato da David Hasko, che e' stato poi opportunamente manipolato per renderlo adeguato alle caratteristiche della trasmittente e del velivolo.

In particolare sono state identificate le seguenti caratteristiche del segnale PPM:

- Boudrate: 500000 (minimo valore di stabilita')
- Numero di canali gestibili: 4
- Valore dei periodi d'onda dei canali all'avvio: 1500 μs (posizione neutra delle superfici di volo)
- Lunghezza totale del pacchetto: 22500 μs
- Lunghezza fronte d'onda di separazione: 300 μs
- Polarita' dell'impulso: 0

Si e' dunque proceduto a creare la parte di codice che permettesse la gestione della lunghezza dei periodi d'onda dei singoli canali. Questo problema e' stato affrontato in due maniere simili, che si sono susseguite per rendere il codice ed il suo utilizzo sempre piu' intuitivo, e che verranno di seguito illustrate.

4.3 Soluzione 1: variazione della posizione dei servo mediante bottoni tattili

Per permettere la variazione del valore di ogni canale si e' quindi ricorsi al semplice utilizzo di 3 pulsanti con le seguenti funzionalita':

- aumento del valore del periodo associato al canale (e quindi alla posizione del servo)
- diminuzione del valore del periodo associato al canale (e quindi alla posizione del servo)
- selezione del canale su cui operare

Onde evitare che la pressione del pulsante permettesse repentini cambiamenti del valore associato allo stesso a seguito di una singola pressione, si e' imposto un tempo minimo entro cui questa non fosse efficace (definito TBNC o "time between next change" ed espresso in *ms*). Questa precauzione si e' resa necessaria a causa della velocita' di esecuzione del loop del codice nell'Arduino che altrimenti avrebbe reso non gestibile l'utilizzo di tale metodologia. Come timer per verificare tale condizione si e' usata la funzione "delay()".

Segue, per praticita' comprensiva, uno schema della configurazione utilizzata. Il circuito e' composto da:

- Scheda Arduino Uno;
- Trasmittente Taranis X9D Plus;
- Cavo jack 3.5 *mm* con terminali "ground" e "segnale";
- 3 resistori [10000 Ω];
- 3 pulsanti tattili.

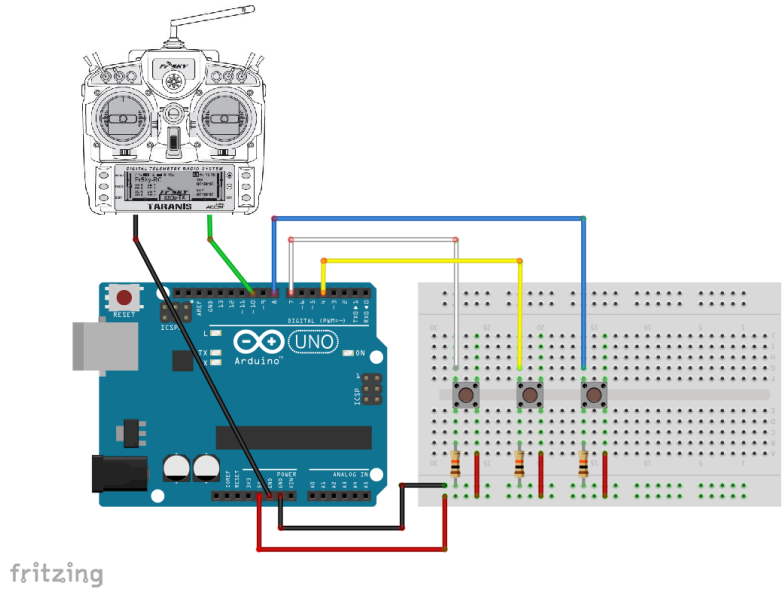

Figura 4.2: Schema del circuito utilizzato per la comunicazione

La trasmittente verra' collegata con Arduino attraverso il cavo con terminale jack. L'Arduino invece sara' connessa attraverso un pin di output PWM (che trasmettera' il segnale PPM).

L'output del segnale PPM generato da Arduino, inviato dunque alla trasmittente, viene visualizzato attraverso il monitor seriale, mentre output grafico viene visualizzato attraverso il plotter seriale.

Viene di seguito illustrato un esempio pratico del codice utilizzato.

Premendo il bottone di variazione del canale si puo' scegliere tra i valori 0, 1, 2, 3, che corrispondono ai canali 1, 2, 3, 4 rispettivamente.

La situazione iniziale e' quella rappresentata nella figura 4.3 dove, per comodita' di utilizzo, si e' usata polarita' dell'impulso pari ad 1. A sinistra il monitor seriale mostra nella prima colonna il canale selezionato, nelle successive colonne invece sono rappresentati i valori dei singoli canali, aggiornati con boudrate pari a 500000. A destra il plotter seriale mostra invece l'andamento del segnale, dove sono facilmente identificabili tutti i canali.

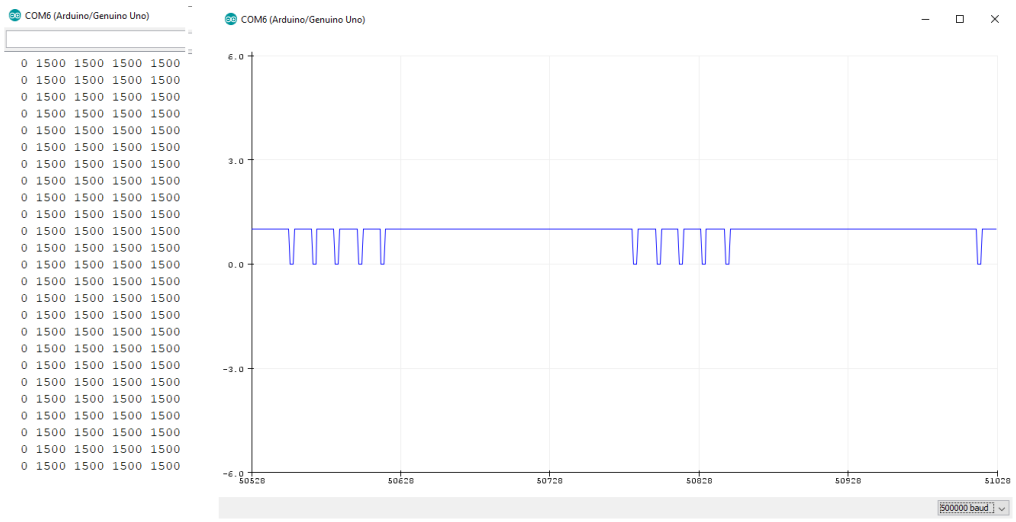

Figura 4.3: rappresentazione del serial monitor e serial plotter per valori del canali pari a 1500

Variando il canale e quindi il valore a questo associato, si ottiene quanto riportato nella figura 4.4. In tal caso si e' portato a 2000 μs il valore del canale 1 e a 1000 μs il valore del canale 3. I canali 2 e 4 non vengono variati e quindi permangono al valore 1500 μs.

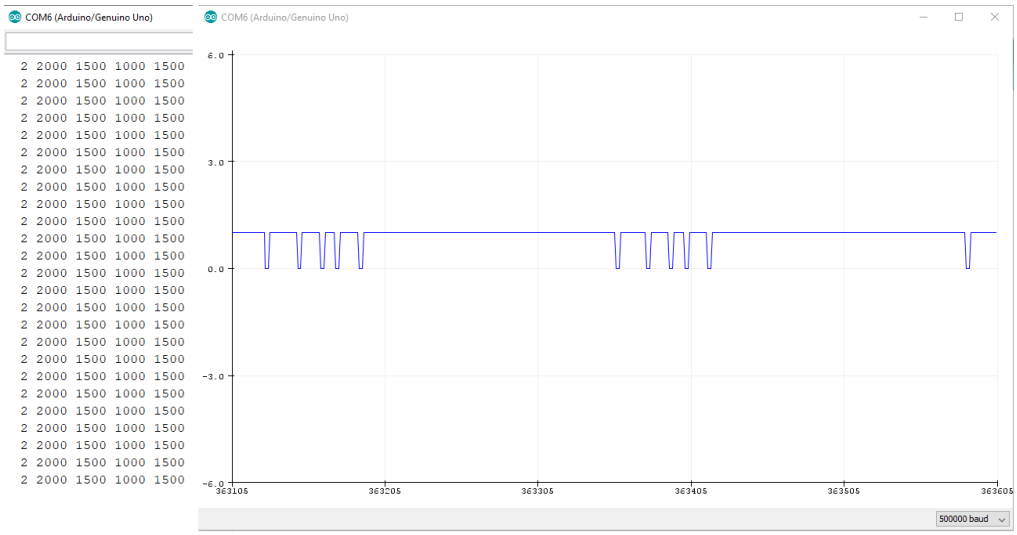

Figura 4.4: rappresentazione del serial monitor e serial plotter per valore del canale 1 = 2000 μs, canale 2 = 1500 μs , canale 3 = 1000 μs, canale 4 = 1500 μs

4.4 Soluzione 2: variazione della posizione dei servo mediante monitor seriale

La seconda soluzione adottata si e' resa necessaria per automatizzare l'invio dei comandi da parte dell'Arduino. L'uso di pulsanti infatti, benche' ottimo in fase di test del codice, risulta scomodo e poco efficace qualora si debbano inviare con frequenze molto elevate dei comandi al velivolo. Per eliminare questa limitazione si e' dunque operato nel modo di seguito illustrato.

Si e' creata una stringa che registrasse l'input dell'utente sotto forma di due parametri:

- Canale selezionato [1 cifra]
- Valore del periodo d'onda del canale [4 cifre]

Dal momento che per i nostri test si sono utilizzati 4 canali con un valore del periodo d'onda $1000 \leq T \leq 2000\ ms$, questi valori rappresentano i limiti entro cui l'utente dovra' operare.

Sono stati dunque inseriti 3 LED di colore diverso per monitorare direttamente lo stato di invio del comando:

- LED verde che si illumina quando il programma viene eseguito correttamente e senza errori
- LED rosso che lampeggia quando il comando inserito dall'utente non e' valido (i valori di uno o entrambe i parametri non rientrano nei limiti sopra citati)
- LED blu che lampeggia quando il comando inserito viene eseguito correttamente

Infine si e' creato un vettore di comandi ed uno temporale per inviare in maniera autonoma una serie di comandi in istanti ben definiti nel tempo. L'utente dovra' definire prima dell'avvio del programma il numero di comandi e il tempo di esecuzione degli stessi. Una volta avviato, questo confrontera' istante per istante l'elemento del vettore temporale i-esimo con il timer di Arduino. Quando tale valore verra' riscontrato, verra' inviato il comando i-esimo contenuto nel vettore dei comandi. Verra' quindi incrementato il contatore e si procedera' fintanto che non verra' ad essere eseguito l'ultimo comando del vettore dei comandi.

4.5 Applicazione pratica e limitazione

Quello appena riportato e' lo sviluppo che ha seguito la necessita' di portare gradualmente il velivolo ad essere sempre piu' autonomo, cioe' indipendente dal comando del pilota. Attraverso l'utilizzo della tecnica PPM, il pilota lancera' il velivolo dall'apposita piattaforma, portandolo ad una quota e ad un assetto opportuni. A questo punto sara' compito del programma gestire i comandi precompilati al fine di far compiere le opportune manovre al velivolo per la generazione di energia elettrica. Come e' facile notare pero', questo metodo presenta una limitazione. Vi e' infatti la dipendenza del velivolo da un elemento quale la trasmittente che rende forzato il campo di operativita' dello stesso alla portata dell'attrezzatura. Questa limitazione per i primi test sara'

fondamentale. Infatti si utilizzera' tale tecnica abbinata alla modalita' maestro - allievo, al fine di riprodurre fedelmente dei comandi, avendo comunque la possibilita' di riprendere il controllo del velivolo in ogni istante da parte del pilota. Essendo infatti fasi di test preliminari, potrebbe essere necessario escludere il programma agente, lasciando al pilota la priorita' di controllo del velivolo, qualora si verificassero delle condizioni di pericolo dovute ad una qualsiasi anomalia che potrebbe presentarsi.

Capitolo 5

Generazione di comandi di volo attraverso MAVLink

La naturale evoluzione del progetto, a seguito dell'identificazione della limitazione della tecnica esposta al capitolo precedente, e' quindi stata quella di rendere il velivolo indipendente dalla trasmittente, montando a bordo dello stesso gli elementi necessari al fine di generare e gestire le opportune manovre.

Questa nuova fase di progetto e' stata possibile attraverso l'uso del protocollo di comunicazione MAVLink per garantire la comunicazione tra la scheda Arduino ed il modulo $PX2$. Prima di descrivere come si e' proceduto pero', e' importante capire cosa MAVLink sia e quali siano le sue principali caratteristiche applicative.

5.1 Il protocollo MAVLink

MAVLink, acronimo di Micro Air Vehicle Link, e' un protocollo di comunicazione molto leggero la cui applicazione e' orientata verso velivoli (ma non solo) di piccole dimensioni ed e' stato rilasciato per la prima volta all'inizio del 2009 da Lorenz Meier.

Tale protocollo e' usato per le comunicazioni tra la stazione di controllo di terra ed il velivolo, ma anche per la comunicazione di sistemi a bordo dello stesso (orientamento nello spazio, posizione GPS, movimenti di strutture gimbal) o con i cosidetti "Companion Computer", quali ad esempio Arduino, Raspberry, etc.

Nel nostro caso proprio questa sua ultima capacita' ci consentira' di gestire efficacemente il sistema velivolo durante il volo.

Il protocollo sfrutta il collegamento seriale, distribuito come librerie di funzioni e messaggi precompilati in formato header-only (header-only message marshalling library).

MAVLink e' stato rilasciato in una prima versione 1.0 e successivamente e' stato migliorato con una versione 2.0. Quest'ultima rispetto alla prima presenta le seguenti migliorie:

- MAVLink2 migliora MAVLink1 consentendo l'aggiunta di nuovi campi ai messaggi MAVLink1 esistenti, supportndo nuovi messaggi con Message ID maggiore di "255" e aggiunge il supporto per la firma dei messaggi
- La struttura dei pacchetti e' variata dalla versione 1.0 alla successiva 2.0 e per completezza verranno riportate entrambe qui di seguito
- MAVLink2 è retrocompatibile con MAVLink1, il che significa che se un dispositivo comprende i messaggi MAVlink2, certamente comprende i messaggi MAVLink1
- Se un dispositivo in grado di comprendere solo MAVLink1 riceve un messaggio che include campi aggiuntivi (aggiunti in MAVLink2), il dispositivo vedrà solo i campi originali. Cioè il dispositivo sarà in grado di leggere il messaggio ma non "vedrà" i campi aggiuntivi
- La porta seriale di un controllore di volo (presumibilmente connessa a una radio di telemetria) può essere impostata per utilizzare MAVLink2 impostando il parametro SERIALx_PROTOCOL su "2" (dove "x" è il numero di porta seriale sul controllore di volo)

5.2 Struttura dei pacchetti

A seguito di quanto sopra riportato, e' facile dunque capire come vi sia una struttura dei pacchetti differente tra le due versioni di MAVLink.

Nei codici utilizzati per questa tesi si e' fatto uso della versione 2.0, ma per completezza verranno di seguito riportate le strutture dei pacchetti per entrambe le versioni.

Campo	Indice (Bytes)	Scopo
Inizio del frame	0	Denota l'inizio del frame di trasmissione (v1.0: 0xFE)
Lunghezza del payload	1	Lunghezza del payload (n)
Sequenza pacchetto	2	Ogni componente conta la sequenza di invio. Consente il rilevamento della perdita di pacchetti.
System ID	3	Identificazione del sistema di INVIO. Permette di differenziare diversi sistemi sulla stessa rete.
Component ID	4	Identificazione del componente di INVIO. Permette di differenziare diversi componenti dello stesso sistema, ad es. IMU e pilota automatico.
Message ID	5	Identificazione del messaggio: l'ID definisce cosa sia il "payload" e come debba essere decodificato correttamente.
Payload	6 ad (n+6)	Il dato nel messaggio,che dipende dal Message ID
CRC	(n+7) ad (n+8)	Check-sum dell'intero pacchetto, escluso il segnale di inizio pacchetto (da LSB a MSB)

Tabella 5.1: Struttura dei pacchetti per la versione di MAVLink 1.0

Campo	Indice (Bytes)	Scopo
Inizio del frame	0	Denota l'inizio del frame di trasmissione (v1.0: 0xFE)
Lunghezza del payload	1	Lunghezza del payload (n)
Flag di incompatibilita'	2	Flag che deve essere compreso per la compatibilita' di MAVLink
Flag di compatibilita'	3	Flag che puo' essere ignorato se non compreso
Sequenza pacchetto	4	Ogni componente conta la sequenza di invio. Consente il rilevamento della perdita di pacchetti.
System ID	5	Identificazione del sistema di INVIO. Permette di differenziare diversi sistemi sulla stessa rete.
Component ID	6	Identificazione del componente di INVIO. Permette di differenziare diversi componenti dello stesso sistema, ad es. IMU e pilota automatico.
Message ID	7 to 9	Identificazione del messaggio: l'ID definisce cosa sia il "payload" e come debba essere decodificato correttamente.
Payload	10 ad (n+10)	Il dato nel messaggio,che dipende dal Message ID
CRC	(n+11) ad (n+12)	Check-sum dell'intero pacchetto, escluso il segnale di inizio pacchetto (da LSB a MSB)
Firma	(n+12) ad (n+26)	Firma per verificare che il messaggio provenga da una fonte affidabile

Tabella 5.2: Struttura dei pacchetti per la versione di MAVLink 2.0

5.3 Integrazione con Arduino

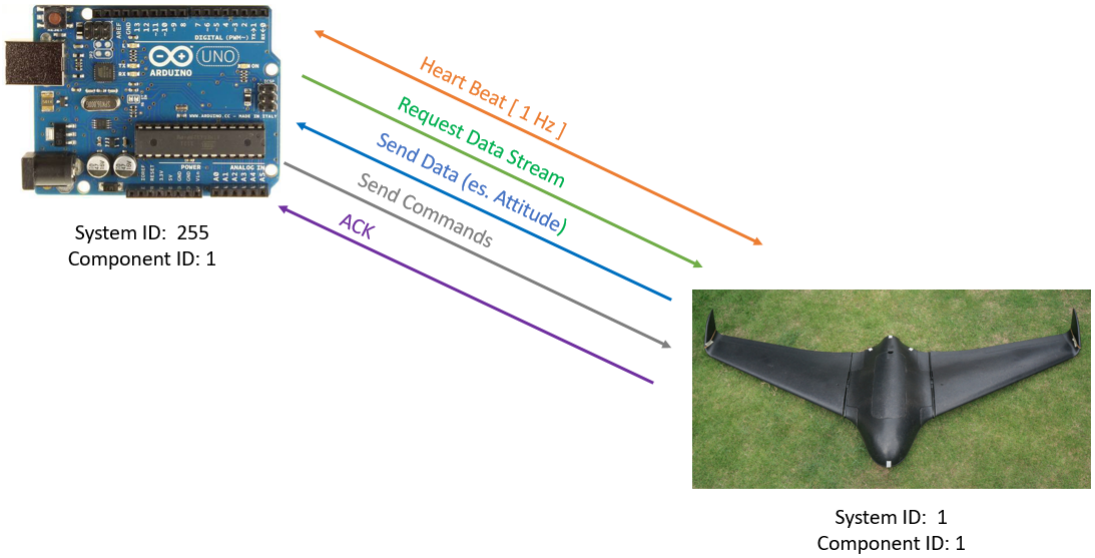

Figura 5.1: Flusso di messaggi I/O Arduino - velivolo

Come si e' potuto notare dai paragrafi precedenti, e' possibile utilizzare il protocollo MAVLink al fine di permettere la comunicazione tra il controllore Pixhawk ed Arduino. Per capire come questa integrazione sia possibile si e' seguita una guida scritta da Juan Pedro López dal forum Ardupilot, a cui sono state apportate le necessarie modifiche per rendere il tutto consono alle necessita' di progetto.

La trattazione che si sta per fare vede l'Arduino collegata con presa USB al computer per il monitoraggio dei dati in ingresso/uscita. Quando si montera' a bordo velivolo, il sistema verra' alimentato dalla batteria dello stesso.

Anzitutto si e' selezionato il modo e metodo di trasmissione dati tra i due apparecchi in termini di Boudrate, oltre che di versione del protocollo, senza il quale si avrebbero problematiche di comunicazione.

Dal programma Mission Planner si e' dunque impostato:

- modo di trasmissione: porta seriale di telemetria "SERIAL2";
- metodo di trasmissione: SERIAL2_PROTOCOL = 2 (protocollo MAVLink 2 coincidente alla versione della libreria in utilizzo su Arduino)
- velocita' di trasmissione: SERIAL2_BAUD = 57 (boudrate pari a 57600, modificabile secondo le esigenze)

Quindi si e' proceduto alla connessione fisica tra i due elementi attraverso la porta di telemetria 2. Questa e' dotata di 6 pin, ma solo 3 sono necessari per Arduino.

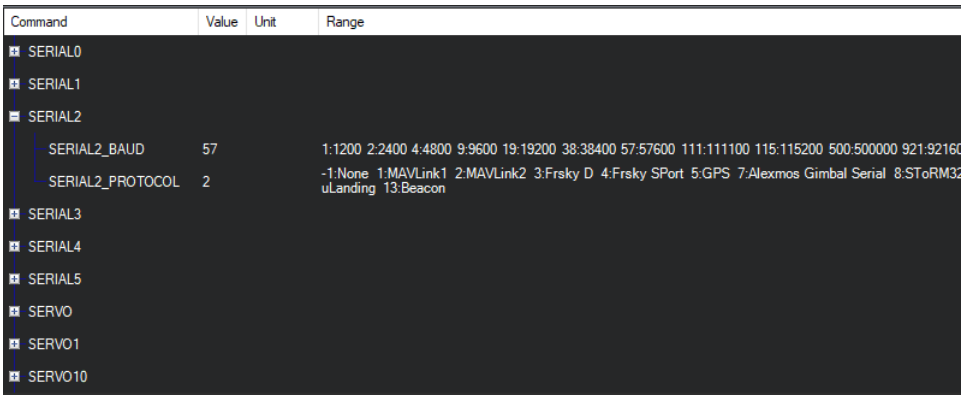

Figura 5.2: impostazione dei parametri di comunicazione tra Pixhawk ed Arduino attraverso Mission Planner

Lo schema utilizzato e la configurazione sono riportati qui di seguito. L'indicizzazione dei Pin della porta di telemetria parte da sinistra con il Pin 1 e a seguire verso destra i Pin successivi.

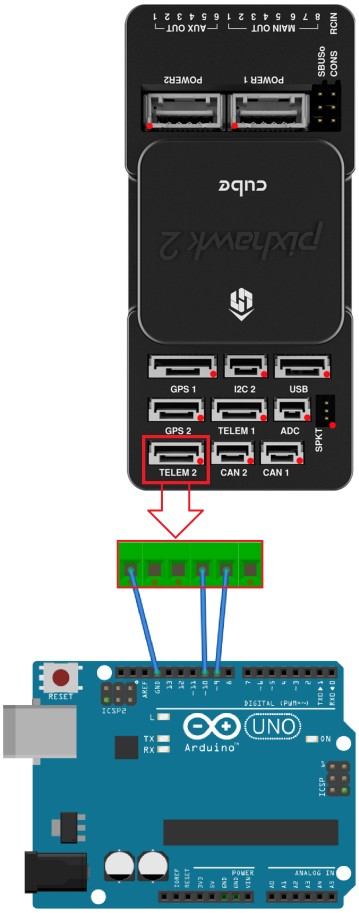

Figura 5.3: connessione PX2 - arduino

Formato: Pin TELEM2 PX2 → Pin Arduino UNO

- Pin 1 → Pin GND
- Pin 2 → nessuna connessione
- Pin 3 → nessuna connessione
- Pin 4 → Pin 10 [PX2(TX) - UNO (RX)]
- Pin 5 → Pin 9 [PX2(RX) - UNO (TX)]
- Pin 6 → nessuna connessione

Si noti che con il termine "TX" si indica la parola trasmissione, con il termine "RX" si indica la parola ricezione. In tal modo il PX2 sara' in grado di mandare i dati di telemetria necessari alla scheda Arduino e quest'ultima sara' in grado di riceverli e di inviare successivamente gli opportuni comandi.

Consolidata quindi la connessione "fisica" si e' potuti passare alla stesura e revisione del codice vero e proprio.

5.4 Il codice

E' necessario fare una doverosa premessa: non esiste alcun controllo del flusso di informazioni tra il controllore ed Arduino, ma ogni volta e' necessario richiedere al controllore le informazioni che si vogliono e bisogna controllare che queste arrivino e che siano non danneggiate.

Si andra' ora a spiegare in dettaglio il codice, con la cura di aver scaricato l'opportuna libreria, come precedentemente discusso.

5.4.1 SoftwareSerial ed Heartbeat

Iniziamo con la dichiarazione della libreria SoftwareSerial.h:

```
#include <SoftwareSerial.h>
```

Questa e' stata inclusa per poter creare una porta seriale aggiuntiva. Si e' sentita la necessita' di adottare una scelta simile poiche', al contrario della scheda Arduino Mega, che dispone di molteplici coppie di pin seriali, l'Arduino Uno dispone di solamente una coppia di queste. In tal maniera dunque, con l'ausilio di una porta seriale aggiuntiva, questa potra' essere usata per le comunicazioni con PX2 in modo da poter utilizzare la porta UART e il monitor seriale nell'IDE Arduino per interagire con la scheda.

Nel nostro caso si sono usati i pin 9 e pin 10 per la creazione di questa porta seriale aggiuntiva, rispettivamente come input ed output di sistema.

```
#define RxPin0 9
#define TxPin0 10

SoftwareSerial pxSerial = SoftwareSerial(RxPin0, TxPin0);
```

Si prosegue dunque all'inizializzazione delle variabili globali, che serviranno come mezzo di comunicazione tra le funzioni stesse per rendere piu' leggero il codice ed evitare l'introduzione di variabili aggiuntive.

```
float roll;           // Roll angle (deg)
float pitch;          // Pitch angle (deg)
float yaw;            // Yaw angle (deg)
```

```
float rollspeed;       // Roll angular speed (deg/s)
float pitchspeed;      // Pitch angular speed (deg/s)
float yawspeed;        // Yaw angular speed (deg/s)

float altitude;        // Current altitude [MSL] (m)
float heading;         // Current heading (deg)
float airspeed;        // Current airspeed (m/s)
float groundspeed;     // Current ground speed (m/s)

int channel_RC[7];     // PPM channels value
```

A questo punto e' necessario richiedere il cosidetto "heartbeat", ovvero un segnale che ci faccia capire molto velocemente che vi sia una effettiva comunicazione con il controllore (se non riceviamo questo segnale, sicuramente non riusciremo a ricevere nulla dal PX2). Per fare cio' bisogna usare l'apposita funzione.

Come gia' anticipato nella sezione 5.1 relativa alla discussione generale sul protocollo MAVLink, questo presenta una struttura header-only. In genere queste funzioni presentano in output un elemento di tipo "struct", al cui interno si trovarno i diversi parametri, a cui e' facilmente garantito accesso una volta ricevuti dall'Arduino.

La funzione mavlink_msg_heartbeat.h, che permette di richiede l'heartbeat, necessita' di diversi parametri di input, qui di seguito riportati:

- &msg → dove andra' ad essere memorizzato l'output
- type → il tipo di velivolo che si sta identificando
- autopilot_type → il tipo di autopilota attivato (se presente)
- system_mode → il modo del sistema (ogni system_type ha i propri)
- custom_mode → se presenti eventuali modi definiti dall'utente
- system_state → lo stato del sistema

Si impacchettano dunque tutti questi input nel messagio da inviare attraverso la funzione annessa alla libreria qui sopra citata:

```
mavlink_msg_heartbeat_pack(sysid,compid, &msg, type,
   autopilot_type, system_mode, custom_mode, system_state)
```

ed infine si invia il messaggio attraverso la porta seriale:

```
uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);

pxSerial.write(buf, len);
```

e si procede alla stampa a video del messaggio che confermi che effettivamente c'e' stata una richiesta di informazioni da parte dell'Arduino.

A questo punto del programma siamo quindi in grado di capire se c'e' comunicazione tra i due sistemi.

Si procede dunque con l'introduzione delle 3 funzioni principali di tutto il codice, ovvero:

- Mav_Request_Data() → Richiesta della telemetria
- comm_receive() → ricezione della telemetria e stampa a video
- elevon_control() → gestione della telemetria ricevuta ed invio del comando di controllo degli elevoni

5.4.2 Mav_Request_Data()

MAVLink consente la richiesta di numerose tipologie di dato: IMU, GPS, valori dei canali collegati ai servoattuatori, assetto, etc.

Tutti queste richieste possono essere trovate all'interno della libreria "common.h":

```
enum MAV_DATA_STREAM
{
        MAV_DATA_STREAM_ALL=0,
        MAV_DATA_STREAM_RAW_SENSORS=1,
        MAV_DATA_STREAM_EXTENDED_STATUS=2,
        MAV_DATA_STREAM_RC_CHANNELS=3,
        MAV_DATA_STREAM_RAW_CONTROLLER=4,
        MAV_DATA_STREAM_POSITION=6,
        MAV_DATA_STREAM_EXTRA1=10,
        MAV_DATA_STREAM_EXTRA2=11,
        MAV_DATA_STREAM_EXTRA3=12,
        MAV_DATA_STREAM_ENUM_END=13,
};
```

Di tutti questi si andranno ad utilizzare i seguenti:

- MAV_DATA_STREAM_EXTENDED_STATUS → stato del GPS. stato della batteria
- MAV_DATA_STREAM_EXTRA1 → dati di assetto del velivolo e rispettive velocita' (rollio, becheggio, imbardata)
- MAV_DATA_STREAM_EXTRA2 → airspeed, groundspeed, altitudine, velocita' di salita', heading, % manetta
- MAV_DATA_STREAM_RC_CHANNELS → valore dei canali associati agli attuatori (in millisecondi)

Si procede a definire i vettori contenenti le richieste degli stream, con una frequenza di 1 Hz (modificabile in base alle esigenze):

```
const int  maxStreams = 4;
const uint8_t MAVStreams[maxStreams] = {
   MAV_DATA_STREAM_EXTENDED_STATUS, MAV_DATA_STREAM_EXTRA1,
   MAV_DATA_STREAM_EXTRA2, MAV_DATA_STREAM_RC_CHANNELS};
const uint16_t MAVRates[maxStreams] = {0x01, 0x01, 0x01, 0x01};
```

quindi, come visto per il caso dell' "heartbeat", si impacchetta ogni stream e si scrive sul buffer (questa volta con ciclo iterativo essendoci 4 stream di dati):

```
for (int i = 0; i < maxStreams; i++) {

mavlink_msg_request_data_stream_pack(2, 200, &msg, 1, 0,
   MAVStreams[i], MAVRates[i], 1);
    uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
#ifdef SOFT_SERIAL_DEBUGGING
    pxSerial.write(buf, len);
#else
    Serial.write(buf, len);
#endif

}
```

In particolare essendo la struttura del messaggio del tipo:

```
mavlink_msg_request_data_stream_pack_chan(system_id,
   component_id,chan, &msg, target_system, target_component,
   req_stream_id, req_message_rate, start_stop)
```

sono stati usati i seguenti valori per i parametri qui sotto riportati per identificare il componente che invia la richiesta dal componente a cui questa e' indirizzata (target). In genere il velivolo e' identificato con il system_id = 1, mentre l'Arduino (o la ground station) con un $1 \leq$ system_id ≤ 256.

- system_id = 2
- component_id = 200
- target_system = 1
- target_component = 0

Non rimane quindi che utilizzare la funzione comm_receive() per leggere i dati derivanti dalla richiesta appena inviata e scriverli a video e/o manipolarli secondo le nostre esigenze.

5.4.3 comm_receive()

Il primo step per ricevere i messaggi dal PX2, dal momento che si stanno utilizzando delle porte seriali, e' quello di controllare se effettivamanente queste ricevano qualcosa:

```
mavlink_message_t msg;
mavlink_status_t status;

#ifdef SOFT_SERIAL_DEBUGGING
  while (pxSerial.available() > 0) {
    uint8_t c = pxSerial.read();
#else
  while (Serial.available() > 0) {
    uint8_t c = Serial.read();
#endif

if (mavlink_parse_char(MAVLINK_COMM_0, c, &msg, &status)) {

  switch (msg.msgid) {
```

Ogni messaggio in ingresso e' caratterizzato da un numero che lo contraddistingue (acessibile nel nostro caso come "msg.msgid"). Dal momento che il numero del messaggio e' univoco, si e' usata una struttura di controllo di tipo "switch".

Si procede all'identificazione della tipologia di messaggio ricevuto e quindi alla decodifica opportuna, ricordando che questo viene recepito come struttura "struct".

Il messaggio da decodificare viene inviato dalla PX2 e posto in "msg" (come si puo' vedere da quanto riportato sopra). Quindi questo viene salvato in una variabile di tipo "struct" opportuna, come di seguito presentato per il caso MAVLINK_MSG_ID_ATTITUDE (numero identificativo del messaggio: 30):

- variabile da cui estrarre il messaggio codificato: mavlink_message_t msg
- variabile in cui salvare il messaggio decodificato: mavlink_attitude_t_attitude
- comando per decodifica: mavlink_msg_attitude_decode(&msg, &attitude)

Per completezza di comprensione si riportano ora in ordine tutti i casi di messaggio analizzati.

- Heartbeat

```
case MAVLINK_MSG_ID_HEARTBEAT:
     {
#ifdef SOFT_SERIAL_DEBUGGING
      Serial.println("PX␣HB");
#endif
```

```
        }
        break;
```

- Dati IMU

```
case MAVLINK_MSG_ID_RAW_IMU:
        {
         mavlink_raw_imu_t raw_imu;
         mavlink_msg_raw_imu_decode(&msg, &raw_imu);
        }
        break;
```

- Assetto e relative velocita' (con conversione da radianti a gradi)

```
case MAVLINK_MSG_ID_ATTITUDE:
        {

         mavlink_attitude_t attitude;
         mavlink_msg_attitude_decode(&msg, &attitude);

#ifdef SOFT_SERIAL_DEBUGGING

         roll = attitude.roll * 180 / 3.1416;
         pitch = attitude.pitch * 180 / 3.1416;
         yaw = attitude.yaw * 180 / 3.1416;

         rollspeed = attitude.rollspeed * 180 / 3.1416;
         pitchspeed = attitude.pitchspeed * 180 / 3.1416;
         yawspeed = attitude.yawspeed * 180 / 3.1416;

#endif
        }
        break;
```

- Variabili HUD

```
case MAVLINK_MSG_ID_VFR_HUD:
        {
           mavlink_vfr_hud_t air_var;
           mavlink_msg_vfr_hud_decode(&msg, &air_var);

#ifdef SOFT_SERIAL_DEBUGGING
```

```
          altitude = air_var.alt;
          heading = air_var.heading;
          airspeed = air_var.airspeed;
          groundspeed = air_var.groundspeed;

#endif
        }
        break;
```

- Valore canali servoattuatori (caso 4 canali)

```
case MAVLINK_MSG_ID_RC_CHANNELS_RAW:
      {
       mavlink_rc_channels_raw_t channel_value;
       mavlink_msg_rc_channels_raw_decode(&msg, &
          channel_value);

#ifdef SOFT_SERIAL_DEBUGGING

       channel_RC[0] = channel_value.chan1_raw;
       channel_RC[1] = channel_value.chan2_raw;
       channel_RC[2] = channel_value.chan3_raw;
       channel_RC[3] = channel_value.chan4_raw;

#endif
      }
      break;
```

- Controllo dell'esecuzione del comando

```
          case  MAVLINK_MSG_ID_COMMAND_ACK: // #76
            {
              mavlink_command_ack_t commandack;
              mavlink_msg_command_ack_decode(&msg, &commandack
                 );
#ifdef SOFT_SERIAL_DEBUGGING

              if (commandack.result == 0) {
                Serial.print("\n Command EXECUTED: ");
                Serial.println(commandack.result);
              }

              if (commandack.result == 1) {
                Serial.print("\n Command NOT EXECUTED: ");
```

```
                Serial.println(commandack.result);
              }

              if (commandack.result == 4) {
                Serial.print("\n Command UNKNOWN/UNSUPPORTED: "
                    );
                Serial.println(commandack.result);
              }

              if (commandack.result == 5) {
                Serial.print("\n Command being executed: ");
                Serial.println(commandack.result);
              }
            }
#endif
            break;
```

5.4.4 elevon_control()

Prima di inviare comandi al modello, e' necessario armarlo, come e' possibile vedere dall'immagine successiva relativa al protocollo di comando.

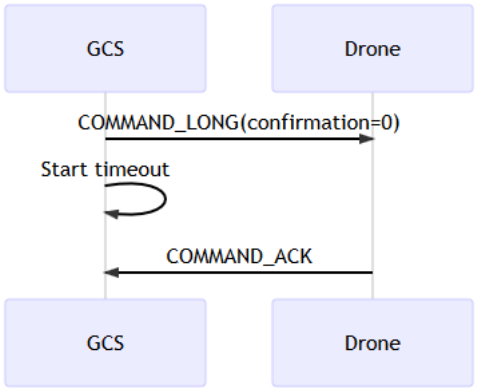

Figura 5.4: Protocollo di comando

Armare il velivolo e' necessario per prevenire che i comandi agiscano quando non e' necessario o quando l'autopilota non e' completamente configurato e pronto.

Per armare il velivolo si fa uso della funzione:

```
mavlink_msg_command_long (#76)
```

.

Come per le funzioni precedenti, il primo step e' quello di impacchettare il comando da scrivere sul buffer:

```
mavlink_msg_command_long_pack(system_id, component_id,
   mavlink_message_t* msg, target_system, target_component,
   command, confirmation, param1, param2, param3, param4, param5
   , param6, param7)
```

.

Come si nota, e' possibile inviare fino a 7 parametri di comando alla volta, ma per i nostri scopi sara' sufficiente inviare il solo comando per armare il velivolo.

Il sesto parametro del messaggio ("command") richiede che sia specificato il numero associato al comando da eseguire, e dunque il parametro associato al comando ("param1").

Nel nostro caso il comando da eseguire e' " MAV_CMD_COMPONENT_ARM_DISARM ", cui e' associato il numero 400, mentre il parametro 1 relativo a tale comando (reperibile dalla documentazione) vede un valore pari a 0 nel caso in cui si voglia disarmare, 1 nel caso in cui si voglia armare il componente.

Essendo i valori di system_id e component_id sempre i medesimi, ecco che il messaggio finale da inviare risultera' il seguente:

```
 mavlink_msg_command_long_pack(2, 200, &msg, 1, 0, 400, 0, 1, 0,
     0, 0, 0, 0, 0);
```

.

Si procede quindi alla scrittura del messaggio sul buffer:

```
  uint16_t len = mavlink_msg_to_send_buffer(buf, &msg);
#ifdef SOFT_SERIAL_DEBUGGING
  pxSerial.write(buf, len);
#else
  Serial.write(buf, len);
#endif
```

A questo punto la richiesta di armare il velivolo e' stata inviata, ma e' necessario capire se effettivamente questo comando sia andato a buon fine o meno.

Nella funzione "comm_receive" al paragrafo precedente, si e' potuta notare la presenza del messaggio MAVLINK_MSG_ID_COMMAND_ACK (#76), corrispondente al controllo dell'esecuzione del comando.

Infatti grazie a questo e' possibile verificare lo stato del comando inviato attraverso la funzione prima esposta. In particolare dalla documentazione e' possibile notare come la struttura della risposta sia cosi composta:

```
typedef struct __mavlink_command_ack_t
{
 uint16_t command; ///< Command ID, as defined by MAV_CMD enum.
 uint8_t result; ///< See MAV_RESULT enum
} mavlink_command_ack_t;
```

Quindi alla variabile "result" sara' associato un valore che deriva dal messaggio MAV_MISSION_RESULT.

Questo propone i seguenti output:

Value	Field Name	Description
0	MAV_RESULT_ACCEPTED	Command ACCEPTED and EXECUTED
1	MAV_RESULT_TEMPORARILY_REJECTED	Command TEMPORARY REJECTED/DENIED
2	MAV_RESULT_DENIED	Command PERMANENTLY DENIED
3	MAV_RESULT_UNSUPPORTED	Command UNKNOWN/UNSUPPORTED
4	MAV_RESULT_FAILED	Command executed, but failed
5	MAV_RESULT_IN_PROGRESS	WIP: Command being executed

Tabella 5.3: Possibili risposte del messaggio MAV_MISSION_RESULT

E' quindi ora possibile operare una richiesta di variazione del valore dei canali associati agli elevoni attraverso il comando:

```
mavlink_msg_rc_channels_override
```

Come anche per le altre funzioni, e' necessario impacchettare il comando con gli opportuni valori dei parametri richiesti per poi scriverlo sul buffer:

```
mavlink_msg_rc_channels_override_pack( system_id, component_id,
   mavlink_message_t* msg, target_system, target_component,
   chan1_raw, chan2_raw, chan3_raw, chan4_raw, chan5_raw,
   chan6_raw, chan7_raw, chan8_raw)
```

Si nota come sia possibile gestire contemporaneamente fino ad un massimo di 8 canali.

Per i segnali dei canali e' possibile assegnare un valore da 1000 – 2000 *ms*, che indicano l'escursione minima e massima rispettivamente associata al comando, mentre qualora non si voglia variare il canale (perche' ad esempio, come nel nostro caso, non si hanno 8 canali) bastera' porre tale valore a 0.

Nel nostro caso, essendo in fase di test il comando, si e' posto arbitrariamente il valore x = 1000 a tutti i canali. Durante i test si potranno variare questi valori in maniera opportuna a seconda dei canali a cui saranno associate le superfici:

```
 int x = 1000;
```

```
  mavlink_msg_rc_channels_override_pack(2, 200, &msg, 1, 0, x, x
     , x, x, x, x, x, x);
  len = mavlink_msg_to_send_buffer(buf, &msg);
#ifdef SOFT_SERIAL_DEBUGGING
  pxSerial.write(buf, len);
#else
  Serial.write(buf, len);
#endif
```

Capitolo 6

Simulatore di volo

Come anticipato nei capitoli precedenti, la tesi in oggetto mira a preparare tutti o quasi gli strumenti necessari al fine di operare un test di volo con il velivolo precedentemente menzionato per stimare quale sia l'energia che e' possibile produrre con un modello in scala.

Se nella prima e seconda parte del lavoro e' stato mostrato come poter efficacemente mandare comandi e ricevere la telemetria del velivolo, ora vi e' la necessita' di creare un modello di volo che prenda come input lo stato del velivolo in termini di assetto, velocita' ed altitudine e restituisca in uscita dal modello i comandi necessari al fine di compiere la manovra voluta.

6.1 Le equazioni del moto

Il sistema di equazioni che si e' usato e' un tipico sistema a 6 gradi di liberta' (6 DOF) in assi corpo, composto da equazioni delle forze, della cinematica e di momento, qui di seguito riportate.

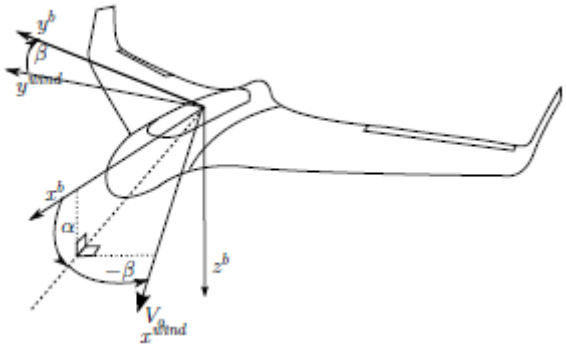

Figura 6.1: Sistema di riferimento in assi corpo velivolo

Equazioni delle forze

$$\begin{cases} \dot{u} = rv - qw - g\sin\theta + \frac{F_z}{m} \\ \dot{v} = -ru + pw + g\sin\phi\cos\theta + \frac{F_y}{m} \\ \dot{w} = qu - pv + g\cos\phi\cos\theta + \frac{F_z}{m} \end{cases}$$

Equazioni della cinematica

$$\begin{cases} \dot{\phi} = p + \tan\theta(q\sin\phi + r\cos\phi) \\ \dot{\theta} = q\cos\phi - r\sin\phi \\ \dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta} \end{cases}$$

Equazioni dei momenti

$$\begin{cases} \dot{p} = (c_1 r + c_2 p)q + c_3 L + c_4 N \\ \dot{q} = c_5 pr - c_6(p^2 - r^2) + c_7 M \\ \dot{r} = (c_8 p - c_2 r)q + c_4 L + c_9 N \end{cases}$$

dove

- $c_1 = \frac{(J_y - J_z)J_z - J_{xz}^2}{\Gamma}$
- $c_2 = (J_x - J_y + J_z)Jx\Gamma$
- $c_3 = \frac{J_z}{\Gamma}$
- $c_4 = \frac{J_{xz}}{\Gamma}$
- $c_5 = \frac{J_z - J_x}{J_y}$
- $c_6 = \frac{J_{xz}}{J_y}$
- $c_7 = \frac{1}{J_y}$
- $c_8 = \frac{J_x(Jx - J_y) + J_x^2}{\Gamma}$
- $c_9 = J_x\Gamma$
- $\Gamma = J_x Jz - J_{xz}^2$

Le componenti delle forze F_x, F_y, F_z contengono al loro interno i contributi di natura aerodinamica del velivolo, del cavo attaccato al velivolo e del vento e potranno essere misurate attraverso gli accelerometri di bordo del controllore $PX2$.

6.2 I coefficienti aerodinamici

I contributi di natura aerodinamica relativi al velivolo possono essere calcolati mediante l'uso dei coefficienti aerodinamici.

Si e' trovato uno studio che ha caratterizzato il velivolo in oggetto sotto questi termini. Essendo l'unico studio noto, non e' stato possibile confrontare i risultati ottenuti con studi analoghi, pertanto l'incertezza sui valori riportati non puo' essere determinata.

Si riportano di seguito le equazioni in funzione dell'angolo di incidenza α:

- $C_l = 0.1039\alpha + 0.2095$
- $C_d = 0.0003\alpha^2 + 0.0016\alpha + 0.0213$
- $C_m = -0.0177\alpha - 0.0858$
- $C_{y_b} = 5 \cdot 10^{-5}\alpha^2 - 0.001\alpha - 0.3204$
- $C_{l_b} = -0.0173\alpha - 0.1586$
- $C_{n_b} = 0.0003\alpha^2 - 0.0019\alpha + 0.0452$
- $C_{y_r} = -0.0006\alpha^2 + 0.0108\alpha + 0.1628$
- $C_{n_r} = -7 \cdot 10^{-5}\alpha^2 - 0.0016\alpha - 0.0257$
- $C_{l_v} = -0.0187\alpha - 0.1505$
- $C_{n_v} = -0.0046\alpha + 0.0438$
- $C_{m_q} = -0.0102\alpha - 2.3565$
- $C_{z_q} = 2 \cdot 10^{-7}\alpha + 2 \cdot 10^{-7}$
- $C_{m_u} = -0.0184\alpha - 0.1572$
- $C_{n_{\delta a}} = -4 \cdot 10^{-5}\alpha + 0.0003$
- $C_{l_{\delta a}} = 2 \cdot 10^{-6}\alpha^2 - 3 \cdot 10^{6}\alpha - 0.0059$
- $C_{m_{\delta e}} = 3 \cdot 10^{-6}\alpha^2 - 8 \cdot 10^{6}\alpha - 0.0106$
- $C_{z_{\delta e}} = 6 \cdot 10^{-6}\alpha^2 - 5 \cdot 10^{-6}\alpha - 0.0208$

6.3 Il cavo

La forza aerodinamica di resistenza generata dal cavo e' stata calcolata come segue.
La forza espressa come $F_c = \frac{1}{2}\rho V^2 S C_{D_c}$ ha necessita' di vedere determinati tutti i parametri.

Il coefficiente di resistenza del cavo $C_D c = 1.2$ e' noto da dati aziendali;

La densita' ρ e' nota, poiche' dai dati di telemetria e' nota l'altitudine "z" di volo rispetto al suolo e quindi e' esprimibile come:

$$\rho = 1.225 \cdot \left(\frac{T_0 - 0.0065 \cdot z}{T_0}\right)^{(m-1)}$$

con:

- $T_0 = 288.15\ K$
- $m = 5.2561$

La superficie del cavo "S" puo' essere approssimata ad un cilindro, di cui solo la meta' viene investito dalla corrente. Pertanto:

$$S = \frac{2\pi R}{2} \cdot L = \pi \cdot R \cdot L$$

con

- $R = 0.005\ m$ [raggio del cavo]
- L lunghezza del cavo [variabile con lo srotolamento]

La velocita' del flusso e' nota come $V = \sqrt{u^2 + v^2 + w^2}$, le cui componenti sono note dalle equazioni del moto;

Considerando pero' una velocita' triangolare lungo la corda, l'equazione della forza diventa:
$F_c = \frac{1}{6}\rho V^2 S C_{D_c}$

6.4 Potenza generata

Per il calcolo teorico della potenza generata, bisogna determinare una velocita' di srotolamento del cavo che ottimizzi le prestazioni.
Il paper "Crosswind kite power" Miles L. Loyd e' un documento di riferimento in questo campo. Da tale studio si e' stimato che la velocita' di srotolamento del cavo dovesse essere pari ad un

terzo della velocita' del vento: $V_l = \frac{1}{3}V_w$
La potenza generata sara' dunque funzione della velocita' di srotolamento e della forza di trazione generata dal cavo: $P = V_l \cdot F_s$ La densita' di energia a disposizione del velivolo invece sara' pari a: $P_w = \frac{1}{2}\rho V_w^3$

Per il calcolo della potenza netta generata nel ciclo dovra' essere anche tenuto in conto il costo energetico della fase di recupero. In questo studio non verra' tenuto in conto il calcolo della potenza generata.

6.5 Flusso logico

Segue uno schema nel quale viene rappresentato il metodo di calcolo utilizzato per il simulatore.

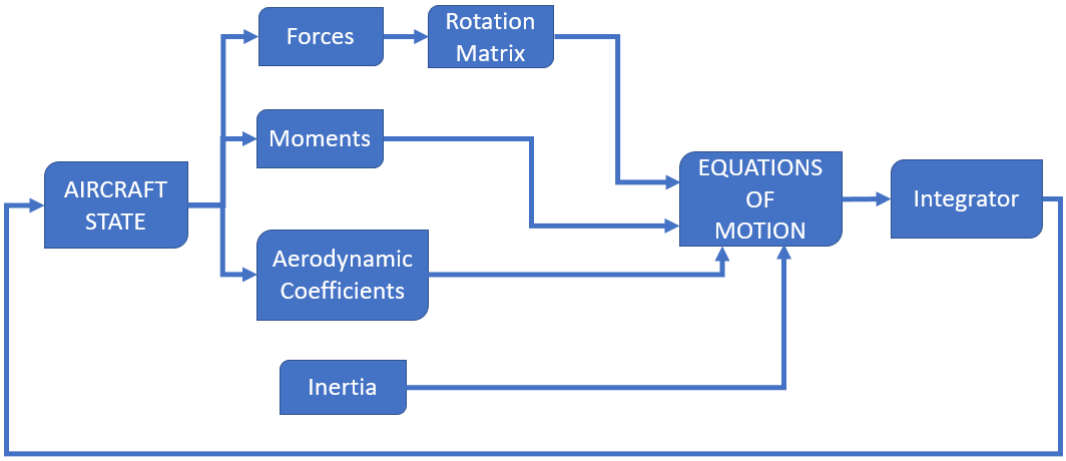

Figura 6.2: Flusso logico del simulatore

A partire dalle condizioni iniziali, e' possibile calcolare i coefficienti aerodinamici in funzione di α, i quali, insieme ai comandi, consentiranno di calcolare forze e momenti agenti sul velivolo. Dunque questi elementi, insieme ai coefficienti di inerzia, permetteranno di calcolare le variabili di stato derivate attraverso le opportune equazioni. Dunque, attraverso una integrazione, sara' possibile ricavare le variabili di stato effettive, che determineranno lo stato del velivolo.

Bibliografia

[1] Patrizia Rosafio, *"Analisi della resistenza aerodinamica introdotta dai cavi del KE60 "*, Kitenergy, 2018.

[2] M. L. Loyd, *"Crosswind Kite Power"*, Journal of Energy, vol. 4, no. 3, 1980

[3] Miguel López Alonso, *"Aeromechanical Model of Electric Wind Turbine with Fixed Wing"*, 2019

[4] Stevens, Lewis, Johnson *"Aircraft control and simulation"*, 2016

[5] Carlo Casarosa *"Meccanica del Volo"*, 2014

[6] Gryte, Hann, Alam, Rohac, Johansen, Fossen *"Aerodynamic modeling of the Skywalker X8 Fixed-Wing Unmanned Aerial Vehicle"*

[7] Shyam Balasubramanian *"MavLink Tutorial for Absolute Dummies (Part –I)"*

[8] *"Pixhawk2 assembly guide"*, HEX

[9] Andre Bernet (Kilrah) *"Opentx per FrSky TARANIS - manuale d'uso versione 1.0.7-J"*

[10] `https://discuss.ardupilot.org/t/mavlink-and-arduino-step-by-step/25566`

[11] `http://ardupilot.org/`

[12] `http://johnjswilliams.weebly.com`

[13] `https://forum.arduino.cc`