Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

System Support for Internet-connected Drones

Relatore: Prof. Luca Mottola

Tesi di laurea di:
Daniel Cantoni Matr. 798984

Anno Accademico 2014–2015

Ai miei genitori

Ringraziamenti

Innanzitutto desidero ringraziare il Professor Luca Mottola per avermi permesso di realizzare questo lavoro e per avermi seguito con pazienza durante tutta la durata della tesi.

Un grandissimo ringraziamento va alla mia morosa Giulia per avermi sostenuto durante i momenti di difficoltà e per avermi dato la forza per andare avanti.

Un grazie va sicuramente a mio fratello che è sempre pronto ad ascoltarmi e disponibile a dedicare il suo tempo per aiutarmi.

Inoltre desidero ringraziare tutti gli Amici del collegio con cui ho condiviso questi anni meravigliosi e tutti i compagni di Università per avermi accompagnato durante il mio percorso di studi.

Infine, il ringraziamento più importante va ai miei genitori per tutti i sacrifici fatti e per aver sempre sostenuto ogni mia scelta. Senza di loro non avrei mai potuto raggiungere questo traguardo.

Abstract

Gli APR (*Aeromobili a Pilotaggio Remoto*), comunemente noti come droni, sono una tecnologia che negli ultimi anni ha suscitato molto interesse ed il cui utilizzo è andato crescendo, soprattutto in ambito civile. Essi sono utilizzati per svolgere applicazioni in vari contesti; ad esempio, sono utilizzati per il telerilevamento, in missioni di soccorso e per missioni di sorveglianza.

Generalmente la programmazione degli APR avviene specificando una lista di *waypoints*, ovvero delle coordinate che il velivolo deve raggiungere in sequenza per completare una missione. Questa tecnica di programmazione fa sì che il drone sia poco intelligente in quanto non sfrutta la possibilità di percepire, tramite i sensori installati a bordo, informazioni riguardanti l'ambiente circostante per modificare dinamicamente il piano di volo al fine di raggiungere in modo autonomo gli obiettivi della missione. Inoltre, al giorno d'oggi, per coordinare l'andamento di una missione è necessario utilizzare una stazione di controllo. Ciò rappresenta un altro limite perché essa, essendo pensata per essere utilizzata da un operatore umano, non permette l'interoperabilità con altri sistemi informatici.

Lo scopo della tesi è quello di superare questi due limiti: in primo luogo, rendendo gli APR capaci di utilizzare tecniche di *Active Sensing* per gestire la navigazione così da poter modificare dinamicamente il piano di volo secondo gli scopi della missione; in secondo luogo, offrendo la possibilità di comunicare con il velivolo mediante l'uso di Internet in modo da abilitare l'impiego dei droni nel contesto dell'*Internet of Things*.

Per raggiungere questi obiettivi abbiamo sviluppato IDrOS - *Internet Drone Operating System*, ovvero un sistema operativo ad alto livello pensato per il controllo degli APR che offre: 1) un modello di programmazione che permette agli APR di utilizzare tecniche di *Active Sensing*; 2) vari binding che permettono di utilizzare alcuni protocolli Internet per comunicare con il velivolo.

IDrOS è stato testato su un prototipo di APR da noi realizzato; i risultati ottenuti sono stati positivi e migliori rispetto a quelli ottenuti dall'applicazione con cui ci siamo confrontati.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Elenco dei Listati

Capitolo 1

Introduzione

Gli APR (*Aeromobili a Pilotaggio Remoto*), comunemente noti come droni, sono una tecnologia che negli ultimi anni ha suscitato molto interesse ed il cui utilizzo è andato crescendo, non solo in campo militare ma anche in quello civile.

Questi aeromobili, infatti, risultano molto più versatili rispetto ai velivoli tradizionali in quanto hanno costi operativi e di gestione minori, possono essere equipaggiati con svariate tipologie di sensori, possono volare a quote basse e, non necessitando della presenza del pilota umano a bordo, possono essere utilizzati nelle operazioni che richiedono di raggiungere aree inaccessibili o impervie oppure in tutte quelle missioni caratterizzate da un elevato pericolo per la vita umana.

La loro versatilità fa sì che vengano impiegati per svolgere numerose applicazioni in contesti molto diversi tra di loro: ad esempio, i droni vengono sempre più diffusamente utilizzati per il telerilevamento e per missioni a supporto delle attività di ricerca e soccorso in aree colpite da calamità naturali.

Ad oggi, gli APR sono pilotati tramite un radiocomando oppure tramite una stazione di controllo: la prima soluzione è utilizzata quasi esclusivamente in contesti amatoriali, mentre la seconda viene utilizzata per svolgere applicazioni più complesse come quelle citate precedentemente.

Tramite la stazione di controllo è possibile programmare il piano di volo da far seguire al velivolo specificando una lista di *waypoints*, ovvero una serie di coordinate geografiche che il drone deve raggiungere in sequenza; questo approccio di programmazione limita l'APR in quanto esso deve seguire solamente piani pre-programmati e raggiungere uno dopo l'altro i punti specificati dall'utente.

Ne segue che la programmazione tramite *waypoints* non rende il drone intelligente in quanto non permette di sfruttare tecniche di *Active Sensing [13]* per gestire la navigazione, ovvero non consente al velivolo di percepire, tramite i sensori o la camera digitale installata a bordo, informazioni riguardanti l'ambiente circostante per decidere autonomamente come gestire la navigazione e come acquisire campioni dai sensori al fine di soddisfare gli obiettivi specificati dall'utente per il completa-

mento della missione. Ad esempio, in un'applicazione dove l'APR è utilizzato in una missione di soccorso per ricercare un'eventuale persona dispersa, nel caso di programmazione con *waypoints* se la vittima non si trova in uno dei punti specificati nel piano di volo non verrà trovata; mentre applicando tecniche di *Active Sensing* il drone è svincolato dall'area da perlustrare, quindi volerà continuamente fino al raggiungimento del suo obiettivo ovvero finché non troverà la vittima. Come si può notare da questo esempio, il limite della navigazione tramite *waypoints* è l'assenza di un modello di programmazione capace di specificare delle condizioni per il completamento della missione che sfruttano i dati provenienti dai sensori installati a bordo del velivolo; ciò rende gli APR capaci di soddisfare solamente le missioni in cui è richiesto di seguire un percorso definito prima del decollo.

Anche l'utilizzo della stazione di controllo per gestire il drone presenta alcuni svantaggi: il primo è che, dato che essa è pensata per essere utilizzata da un operatore umano, non permette l'interoperabilità con altri sistemi informatici; il secondo è che l'APR non può uscire dall'area di copertura radio poiché, in queste circostanze, la stazione non sarebbe più in grado di inviare i comandi di volo e perderebbe il controllo del velivolo - questo aspetto limita molto l'utilizzo dei droni nelle missioni che richiedono un raggio operativo ampio.

Per quanto riguarda i problemi di interoperabilità tra APR ed altri sistemi informatici, nonostante negli ultimi anni si stia parlando sempre più spesso di *Internet of Things* (IoT) [6] - ovvero di estendere la rete a tutti i dispositivi elettronici di uso comune così da poterli interconnettere l'uno con l'altro e facilitare l'interfacciamento con l'utente, non esistono sistemi che permettono di combinare IoT ed APR al fine di eliminare la stazione di controllo e permettere la comunicazione con gli APR mediante l'utilizzo di Internet.

A sottolineare l'importanza dell'IoT al giorno d'oggi è uno studio effettuato dalla *Gartner* [18], multinazionale leader mondiale nel campo dell'*Information Technology,* secondo il quale, nel 2015, tra le tecnologie emergenti per le quali si hanno più aspettative per il futuro e sulle quali molte aziende stanno investendo i propri capitali c'è appunto l'*Internet of Things.*

1.1 Contributo

Gli obiettivi principali che ci si è voluti prefiggere in questo lavoro di tesi sono i seguenti:

1. Proporre un modello di programmazione per il controllo degli APR mediante tecniche di *Active Sensing*: in tal modo è possibile superare i vincoli imposti dalla navigazione tramite *waypoints* permettendo così ai droni di volare basandosi sui valori letti dai sensori installati a bordo al fine di raggiungere autonomamente gli obiettivi della missione.

2. Proporre un'architettura software per estendere i principi dell'IoT al mondo dei droni; in particolare, il nostro scopo è quello di eliminare la stazione di controllo e sostituirla con delle interfacce basate su protocolli Internet per permettere a qualunque dispositivo connesso alla rete di pilotare gli APR.

L'unione di questi due obiettivi fa sì che l'APR risulti intelligente, autonomo e facilmente interfacciabile con molti sistemi informatici; ad esempio, il drone potrebbe essere integrato in sistemi di *Business Processes* ed essere visto come un componente che espone le proprie funzionalità per svolgere applicazioni più complesse: in tal modo, si potrebbe creare un flusso di lavoro che, ad un certo punto dell'esecuzione, richiede al drone di volare per acquisire dei dati che verranno utilizzati successivamente per compiere delle analisi.

Per raggiungere questi obiettivi abbiamo sviluppato IDrOS - *Internet Drone Operating System*, ovvero un sistema operativo ad alto livello che rende gli APR capaci di utilizzare, tramite un opportuno modello di programmazione, tecniche di *Active Sensing* per gestire la navigazione e che offre delle interfacce Internet per permettere a qualunque dispositivo di controllare l'APR. Per fare ciò IDrOS si occupa di gestire la comunicazione con i dispositivi elettronici che controllano l'assetto del velivolo e tutte le risorse hardware presenti a bordo dell'APR, come i sensori utilizzati per acquisire dati durante l'esecuzione di una missione. Nel raggiungere questi obiettivi, IDrOS deve tenere conto della natura embedded del mondo in cui deve operare: in particolare, dovendo essere eseguito su una piattaforma elettronica installata a bordo del velivolo, non deve consumare troppe risorse hardware durante il suo funzionamento.

Una delle caratteristiche più interessanti del nostro sistema è il fatto di essere una piattaforma *general purpose* che permette agli utenti di utilizzare il drone in missioni diverse tra di loro: per fare ciò, prima di effettuare una richiesta al servizio Internet per la gestione del volo, l'utente deve caricare a bordo del sistema - mediante le interfacce Internet messe a disposizione - le istruzioni necessarie per la gestione autonoma del volo e le regole su come acquisire i dati dai sensori. Dato che missioni diverse richiedono spesso l'utilizzo di sensori diversi, IDrOS prevede che per installare un nuovo sensore l'utente debba solamente fornire al sistema un driver. Questo aspetto sottolinea quanto IDrOS sia versatile: infatti, l'utente non deve modificare il codice relativo alla gestione della missione per utilizzare un nuovo sensore.

Il lavoro di tesi non si è limitato solamente alla progettazione ed alla realizzazione di IDrOS ma, per valutare l'efficacia del sistema proposto, abbiamo costruito un prototipo di APR con il quale abbiamo condotto dei test per simulare alcune applicazioni reali. Durante i voli abbiamo monitorato le prestazioni di IDrOS in

termini di consumo di CPU e memoria e le abbiamo confrontate con quelle di MAVProxy, un programma per il controllo degli APR presente sul mercato che offre molte meno funzionalità rispetto ad IDrOS, ottenendo dei risultati migliori; ad esempio, IDrOS occupa meno del 30% delle risorse di CPU offerte dalla piattaforma embedded utilizzata, mentre MAVProxy ne occupa più del 40%.

1.2 Struttura della Tesi

Nel Capitolo 2 verrà presentato lo stato dell'arte riguardo l'IoT e gli APR.

Quindi, per prima cosa, verrà illustrato il concetto di *Internet of Things* focalizzando l'attenzione all'integrazione di sistemi embedded con Internet. In questa prima parte si descriveranno due dei protocolli di rete maggiormente utilizzati per interfacciare dispositivi dotati di limitate capacità in termini computazionali e di memoria ad Internet: CoAP e MQTT.

Successivamente si parlerà di APR analizzando le caratteristiche principali di questi velivoli e dei sistemi utilizzati per pilotarli; inoltre, verranno illustrati vari esempi di utilizzo dei droni in ambito civile - tali esempi sono tratti da progetti realizzati da università, centri di ricerca o da aziende operanti in questo settore. La fase di studio degli utilizzi dei droni in scenari reali è stata fondamentale per la realizzazione del lavoro di tesi in quanto ci ha permesso di avere una visione globale sui requisiti richiesti dalle applicazioni svolte mediante l'ausilio dei droni.

Nel Capitolo 3, sulla base dello studio riguardante le missioni che sono più comunemente svolte dai droni, siamo andati a classificare l'utilizzo degli APR in ambito civile in tre categorie. Successivamente, partendo da tale classificazione, abbiamo presentato il modello di programmazione da noi proposto per permettere al drone di utilizzare tecniche di *Active Sensing* per gestire la navigazione e per permettere all'utente di fare eseguire al drone svariate tipologie di missioni appartenenti a ciascuna delle tre categorie individuate.

Il modello di programmazione da noi proposto supera lo stato dell'arte perché rende il drone autonomo e capace di gestire le missioni a seconda dei valori letti dai sensori al fine di raggiungere gli obiettivi specificati dall'utente.

Nel Capitolo 4 abbiamo illustrato l'architettura scelta per sviluppare IDrOS al fine di interconnettere il velivolo ad Internet attraverso più protocolli di rete e di gestire le missioni attraverso il modello di programmazione presentato al Capitolo 3. Oltre a questi compiti, l'architettura deve semplificare all'utente la gestione dei sensori installati a bordo del velivolo e garantire la comunicazione con il software di volo fornito dal produttore del velivolo. Il capitolo termina presentando due configurazioni di *deployment* con le quali è possibile utilizzare IDrOS; per ognuna di esse

sono presentati i vantaggi e gli svantaggi.

Nel Capitolo 5 vengono presentati i dettagli riguardanti l'implementazione e le problematiche che abbiamo dovuto affrontare durante lo sviluppo di IDrOS.

Per prima cosa viene illustrata la piattaforma hardware e software che abbiamo scelto di utilizzare come controllore di stabilità. In seguito, viene presentata la configurazione di *deployment* che abbiamo deciso di utilizzare per la nostra implementazione; dato che essa prevede che IDrOS venga installato a bordo dell'APR, abbiamo dovuto affrontare la problematica relativa all'interfacciamento fisico tra il dispositivo su cui è eseguito IDrOS ed il controllore di volo.

Nel capitolo sono poi illustrati i dettagli relativi all'implementazione dell'architettura definita al Capitolo 4 descrivendo come sono state concretamente sviluppate le interfacce Internet ed i componenti per la gestione del volo secondo il modello di programmazione da noi proposto.

Nell'ultima sezione viene presentato il velivolo che abbiamo realizzato discutendo i componenti necessari per farlo volare e per far sì che si potesse utilizzare IDrOS come gestore delle missioni.

Nel Capitolo 6 sono presentati alcuni casi di studio che abbiamo effettuato per valutare le prestazioni del nostro sistema; in particolare, abbiamo eseguito una serie di voli reali: uno che simula una missione di *ricerca di dispersi sotto le valanghe* ed uno in un'applicazione di *videosorveglianza.* Durante questi voli abbiamo monitorato il consumo di CPU e l'occupazione di memoria richiesti da IDrOS per poter funzionare; i dati ottenuti da questi esperimenti possono essere considerati positivi in quanto IDrOS durante l'esecuzione dei voli di test ha richiesto meno del 30% delle risorse offerte dalla CPU ed ha occupato solamente 35MB di RAM dei 512MB offerti dal dispositivo embedded utilizzato.

Nell'ultima sezione del capitolo sono presentati i risultati ottenuti dai test effettuati in una configurazione *Hardware in the loop* - ovvero collegando la board utilizzata dal nostro prototipo per l'esecuzione di IDrOS ad un simulatore - per valutare come scala il sistema al variare del carico computazionale richiesto: sono stati effettuati dei test per valutare la frequenza massima di campionamento dei sensori, il numero massimo di sensori utilizzabili ed il numero massimo di client che il prototipo da noi realizzato può gestire. Anche in questo caso i risultati ottenuti sono positivi in quanto abbiamo riscontrato che IDrOS scala linearmente al variare del carico di lavoro richiesto.

Nell'ultimo capitolo vengono effettuate alcune considerazioni finali sul lavoro di tesi e vengono presentati possibili sviluppi futuri.

Capitolo 2

Stato dell'Arte

Come anticipato nel Capitolo 1 di questo elaborato, uno degli obiettivi del lavoro di tesi è quello di integrare l'utilizzo degli *Aeromobili a Pilotaggio Remoto* (APR) nell'ambito dell'*Internet of Things* (IoT).

Pertanto, come prima cosa, descriveremo il concetto di *Internet of Things* ed illustreremo in dettaglio i protocolli di comunicazione più utilizzati per connettere ad Internet i vari dispositivi che operano in questo contesto.

Successivamente, parleremo degli aeromobili a pilotaggio remoto concentrandoci sulle loro caratteristiche e facendo svariati esempi del loro utilizzo in ambito civile.

2.1 Sistemi Embedded ed Internet

I recenti progressi tecnologici nell'elettronica digitale e nei sistemi di comunicazione hanno portato allo sviluppo di piccoli dispositivi in grado di comunicare con il mondo esterno mediante l'utilizzo di Internet. In questi scenari si parla di *Internet delle Cose* o *Internet of Things* (IoT) [6] ovvero l'estensione della rete a tutti gli apparecchi elettronici di uso comune per poter interagire con essi mediante l'utilizzo di Internet.

Grazie alla IoT l'interazione tra uomo e macchina trae molti benefici in quanto è possibile controllare le proprie apparecchiature elettroniche da qualsiasi parte del mondo. Inoltre, si può parlare anche di interazione macchina-macchina in quanto i vari componenti della rete possono comunicare tra di loro per ottimizzare e automatizzare i compiti da eseguire.

Ad esempio, una lavatrice progettata per operare nel contesto dell'IoT [24] non avrebbe nessun programma di lavaggio salvato localmente ma alla pressione del pulsante d'avvio verrebbe fatta una richiesta ad un server web per ottenere le istruzioni da eseguire per poter procedere al lavaggio.

Questo approccio garantisce molti vantaggi all'utente: ad esempio, i programmi per il lavaggio potrebbero essere continuamente aggiornati offrendo nuove funziona-

lità; inoltre, la lavatrice sarebbe in grado di interagire con gli altri elettrodomestici e/o dispositivi dell'utente per notificare il proprio stato. Ad esempio si potrebbe notificare la fine del lavaggio o un guasto tecnico. In Figura 2.1 è rappresentata l'interfaccia di una lavatrice pensata per operare nel contesto dell'IoT: ogni pulsante, sensore e attuatore è modellizzato come una risorsa che permette l'interazione con altri dispositivi.

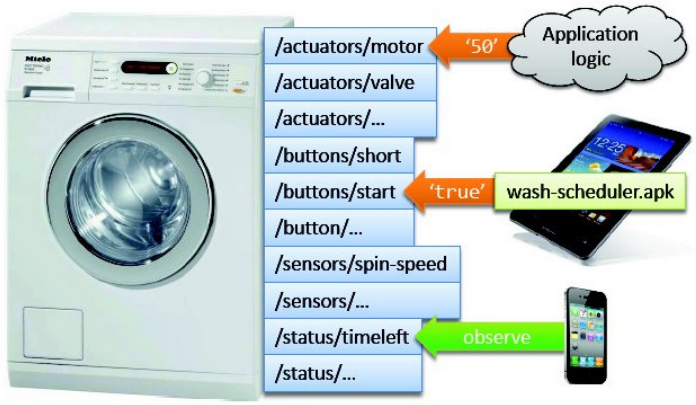


Figura 2.1: Esempio di interfaccia di un sistema operante nel contesto dell'IoT

Dal punto di vista tecnico il paradigma dell'*Internet of Things* fa nascere la necessità di connettere ad Internet dispositivi *resource constrained*, cioè apparecchiature con possibilità di calcolo limitate e con poca memoria; in genere si parla di microcontrollori a 8-16 bit, con frequenze di calcolo intorno ai 16 MHz, con poche decine di KB di RAM e spesso alimentati a batteria.

In questo scenario, date le poche risorse disponibili, l'utilizzo del protocollo HTTP non è adatto per collegare i dispositivi a Internet perché esso è stato progettato per trasferire grosse quantità di dati e per l'interazione con l'utente. Tuttavia, sono stati sviluppati dei protocolli di comunicazione più leggeri che permettono il collegamento ad Internet anche ai dispositivi con minori capacità di calcolo. Esempi di questi protocolli sono CoAP e MQTT.

2.1.1 CoAP

CoAP - *Constrained Application Protocol* [39] è un protocollo web, simile ad HTTP, adatto alle esigenze dei dispositivi con risorse limitate in termini computazionali ed energetiche. Lo scopo di CoAP non è quello di comprimere semplicemente il protocollo HTTP affinché sia utilizzabile dai dispositivi *resource constrained*, ma è quello di creare un sottoinsieme di funzionalità di tipo REST [17] compatibili con HTTP e ottimizzate per la comunicazione macchina-macchina.

Il paradigma REST (*REpresentational State Transfer*) enuncia un insieme di principi su come deve essere strutturata l'architettura di rete; in particolare, le caratteristiche che vengono definite sono le seguenti:

- **Stateless**: la comunicazione tra client e server deve essere vincolata in modo che, tra le varie richieste, nessun'informazione riguardante il client venga memorizzata sul server. Ogni richiesta proveniente dai client deve perciò contenere tutte le informazioni affinché il server sia in grado di erogare il servizio richiesto indipendentemente dalle richieste precedenti.

- **Cachable**: i client possono fare *caching* delle richieste al fine di migliorare le prestazioni. Se una risposta del server è etichettata come *cachable*, il client può memorizzarla e sfruttarla per rispondere alle sue successive richieste; in queste circostanze il traffico di rete diminuisce notevolmente e si ha una latenza minore nell'ottenere le risposte in quanto si sfruttano le informazioni memorizzate localmente.

- **Risorse**: un concetto fondamentale in REST è l'esistenza di risorse, ovvero fonti di informazioni che un server mette a disposizione ai vari client. I client possono accedere alle varie informazioni contenute nelle risorse specificando un identificativo globale (URI) che si riferisce a ciascuna risorsa.

2.1.1.1 Tipologie di Messaggi

L'architettura per il funzionamento del protocollo CoAP, analogamente a quella di HTTP, è basata sul modello *client-server*. Una richiesta CoAP è equivalente ad una richiesta HTTP: essa è inviata da un client ad un server per richiedere a quest'ultimo l'esecuzione di un'azione su una risorsa; il server, dopo aver ricevuto e processato la richiesta, invia al client una risposta contenente un codice di risposta ed un'eventuale rappresentazione della risorsa specificata.

A differenza di HTTP, il protocollo CoAP realizza gli scambi di messaggi tra client e server in maniera asincrona utilizzando il protocollo di trasporto UDP. Dato che UDP non fornisce dei meccanismi che garantiscono l'effettiva consegna dei messaggi, come avviene con TCP, CoAP fornisce un sottolivello tra il livello applicativo e il livello di trasporto che si occupa opzionalmente di garantire un minimo di affidabilità sulla consegna dei messaggi.

In Figura 2.2 vengono mostrati i due sottolivelli che compongono CoAP: il sottolivello *Messages* si occupa di gestire lo scambio di messaggi che avviene in maniera asincrona su UDP, il sottolivello *Requests/Responses* è quello incaricato di elaborare i messaggi.

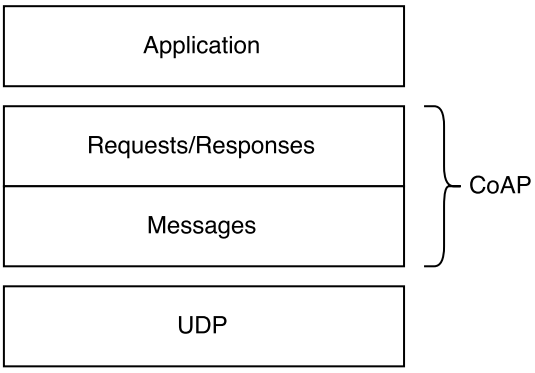


Figura 2.2: Livelli del protocollo CoAP

Per garantire la consegna dei messaggi tra client e server CoAP definisce quattro possibili tipologie di messaggi:

- **Confirmable (CON):** questo tipo di messaggio viene utilizzato quando il mittente vuole avere la garanzia dell'effettiva consegna del messaggio. Quando un pacchetto è contrassegnato *confirmable*, il destinatario deve mandare un messaggio di *acknowledgement* per confermare la ricezione del messaggio. In caso di perdita del pacchetto o dell'ACK, il messaggio deve essere ritrasmesso duplicando ogni volta il tempo tra una trasmissione e l'altra fino al raggiungimento di un numero massimo di ritrasmissioni. Lo standard che definisce il protocollo consiglia di utilizzare 2 secondi come tempo massimo prima della prima ritrasmissione e fissa a 4 il numero massimo di ritrasmissioni. Il messaggio di ACK inviato dal destinatario al mittente deve avere lo stesso identificativo del pacchetto CON che l'ha generato.

- **NON-confirmable (NON):** a differenza dei messaggi *confirmable*, questi tipi di messaggi non richiedono l'invio dell'ACK da parte del destinatario. In questo caso il mittente non avrà la garanzia dell'avvenuta consegna del pacchetto. Generalmente questa tipologia di messaggi viene utilizzata per le trasmissioni che non richiedono affidabilità, ad esempio l'invio continuativo delle rilevazioni fatte da un sensore.

- **Acknowledgement (ACK):** questi messaggi servono per notificare la ricezione dei messaggi *confirmable*. In alcune circostanze l'ACK potrebbe contenere la risposta della risorsa richiesta: in questo caso di parla di *piggybacked response*; questa situazione è comune quando il server ha disponibile la risposta da inviare al mittente al momento dell'arrivo della richiesta. Quando la risposta non è immediatamente disponibile, il messaggio di ACK sarà vuoto e verrà inviato al mittente un ulteriore messaggio contenente la rappresentazione della risorsa richiesta non appena il server avrà le informazioni necessarie per rispondere;

in questo caso si parla di *separate response*. In entrambi i casi, l'ACK dovrà avere lo stesso identificativo del messaggio CON che l'ha generato.

- **Reset (RST):** questo tipo di messaggio viene inviato in risposta ad un pacchetto che per qualche motivo il server non è stato in grado di processare. Il pacchetto RST, come l'ACK, deve avere lo stesso identificativo del messaggio che l'ha generato.

2.1.1.2 Richieste e Risposte

La comunicazione tra i vari host CoAP avviene secondo il modello *query-reply* simile a quello utilizzato da HTTP: un client invia un messaggio di richiesta ad un server il quale, dopo averlo elaborato, invia un messaggio di risposta. Contrariamente ad HTTP, lo scambio di messaggi tra client e server non avviene dopo aver stabilito una connessione, ma avviene in maniera asincrona.

Richieste

Una richiesta CoAP è un messaggio *confirmable* o *NON-confirmable* formato da: un metodo da applicare ad una risorsa, un identificatore (URI) della risorsa, un eventuale *payload* e dei meta-dati riguardanti la richiesta.

I metodi da applicare alla risorsa supportati dal protocollo sono:

- **GET:** tramite il metodo GET si richiede di ottenere una rappresentazione della risorsa identificata dall'URI.
- **POST:** tramite il metodo POST il client richiede che la rappresentazione della risorsa identificata dal campo URI venga elaborata dal server. Solitamente il risultato di questa operazione è l'aggiornamento della risorsa oppure la creazione di una nuova risorsa se questa non esiste.
- **PUT:** tramite il metodo PUT il client richiede al server che la risorsa identificata tramite l'URI presente nel messaggio venga creata oppure aggiornata con la rappresentazione inclusa nel messaggio.
- **DELETE:** il metodo DELETE serve per richiedere che la risorsa identificata dall'URI venga eliminata.

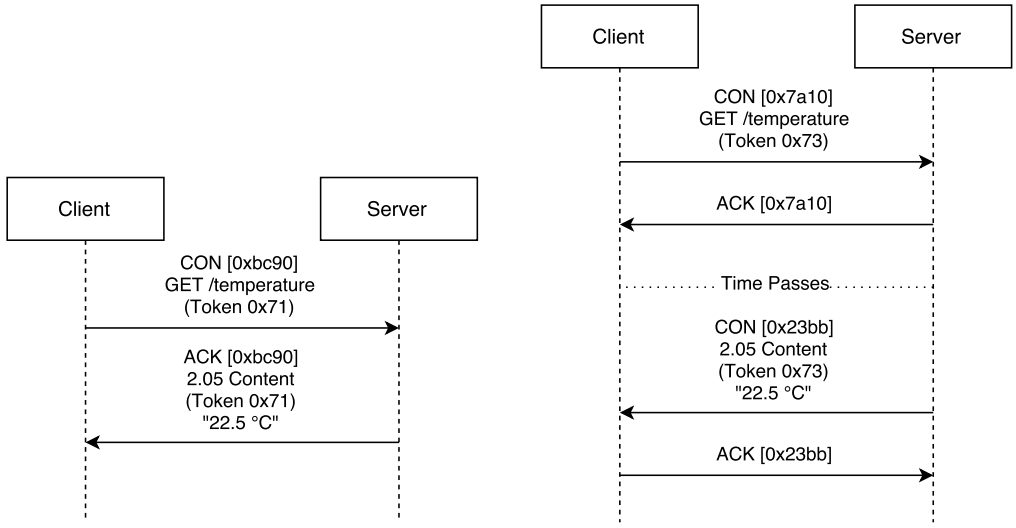
Risposte

Dopo aver ricevuto e processato una richiesta, il server deve inviare al client un messaggio di risposta. Come nel protocollo HTTP, anche in CoAP esistono dei codici di risposta che indicano l'esito dell'azione richiesta; i valori che possono assumere sono:

- **2.xx - Successo:** la richiesta è stata ricevuta, interpretata e correttamente elaborata.
- **4.xx - Errore del client:** la richiesta del client contiene degli errori, perciò non è stata soddisfatta.
- **5.xx - Errore del server:** la richiesta del client era valida, ma il server non è stato in grado di soddisfarla.

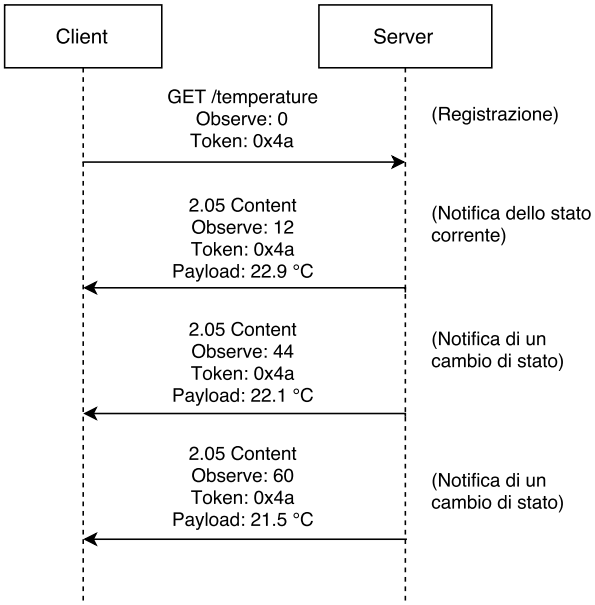
Le modalità con cui il server può rispondere alle richieste si possono suddividere in quattro tipologie:

1. **Piggybacked:** se la risposta ad una richiesta di tipo *confirmable* è disponibile immediatamente, essa può essere subito inserita nel messaggio di *acknowledgement* utilizzato per notificare la ricezione del messaggio, così da non rendere necessario l'invio della risposta in un secondo messaggio. Il codice di risposta di un messaggio *piggybacked* non deve essere necessariamente di tipo successo, ma può contenere un errore. In Figura 2.3a sono mostrati i messaggi scambiati tra un client ed un server nel caso in cui la risposta del messaggio è immediatamente disponibile.
2. **Separate:** se la risposta ad una richiesta di tipo *confirmable* non è immediatamente disponibile, essa è inviata in un messaggio separato. Questo caso si verifica ad esempio quando il server, per rispondere, impiega un tempo maggiore rispetto a quello di attesa per la ritrasmissione della richiesta. La Figura 2.3b mostra un esempio di risposta *separate*.
3. **Observe:** quando in una richiesta è presente l'opzione *observe* [21], il server salva il client nell'elenco degli osservatori della risorsa specificata dall'URI della richiesta ed invia al client la rappresentazione corrente della risorsa. Ogni qualvolta che la risorsa su cui è registrato un *observer* cambia il proprio stato, il server notifica il client fornendogli la nuova rappresentazione della risorsa. In Figura 2.3c è rappresentato lo scambio di messaggi tra client e server nel caso in cui venga registrato un *observer* sulla risorsa *temperature*.
4. **NON-confirmable:** se la richiesta è di tipo *NON-confirmable*, il server una volta ottenuta la rappresentazione della risorsa specificata risponderà a sua volta con un messaggio *NON-confirmable*.



(a) Richiesta *confirmable* seguita da risposta *piggybacked*

(b) Richiesta *confirmable* seguita da risposta *separate*



(c) Scambio di messaggi nel caso di richiesta con opzione *observe*

Figura 2.3: Modalità di risposta previste dal protocollo CoAP

2.1.1.3 Discovery

CoAP fornisce un servizio di *discovery* per dare ai client la possibilità di conoscere le risorse che mette a disposizione un server; questo servizio è molto importante specialmente nelle applicazioni in cui è richiesta l'interazione macchina-macchina.

CoAP permette inoltre ai client di utilizzare un indirizzo *multicast* per scoprire quali sono i server presenti nella rete; in questo caso, i server che offrono il servizio

di *discovery* devono supportare una connessione sulla porta 5683, che è la porta di default per il protocollo CoAP.

2.1.2 MQTT

MQTT - *Message Queuing Telemetry Transport* [19] è un protocollo di rete particolarmente leggero basato sul paradigma architetturale *publish/subscribe*; esso è stato progettato da IBM nel 1999 per essere utilizzato da dispositivi embedded per la comunicazione macchina-macchina. Data la sua semplicità, il suo uso è adatto in ambienti che impongono dei vincoli prestazionali come ad esempio:

- quando l'infrastruttura di rete offre una banda limitata;
- quando i dispositivi che si vogliono utilizzare hanno capacità limitate in termini di memoria e potenza di calcolo.

La caratteristica che contraddistingue MQTT da CoAP è il fatto di utilizzare un'architettura *publish/subscribe* per gestire lo scambio di messaggi tra i vari dispositivi che compongono la rete. In questo schema architetturale i mittenti e i destinatari dei messaggi dialogano attraverso un tramite, detto dispatcher o broker.

Il mittente di un messaggio (detto *publisher*) non deve essere consapevole dell'identità dei destinatari (detti *subscriber*); esso si limita a pubblicare il proprio messaggio al broker specificando il *topic* che contraddistingue il suo contenuto.

I destinatari si rivolgono a loro volta al broker sottoscrivendosi alla ricezione di messaggi etichettati con un certo *topic*. Il compito del broker è quello di inoltrare ogni messaggio inviato da un *publisher* a tutti i *subscriber* interessati al *topic* di quel messaggio.

In Figura 2.4 è mostrato il funzionamento del paradigma *publish/subscribe* utilizzato dal protocollo MQTT: dopo la sottoscrizione ad un determinato *topic* da parte di un *subscriber,* il broker deve recapitargli tutti i messaggi che i *publisher* pubblicano contenenti quel *topic*.

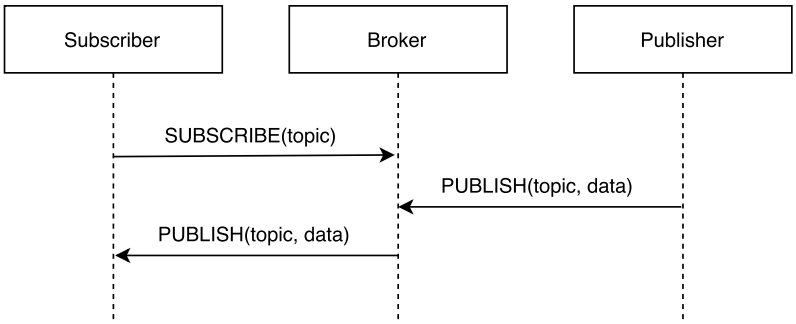


Figura 2.4: Esempio di funzionamento del paradigma *publish/subscribe* utilizzato da MQTT

Un'ulteriore differenza rispetto a CoAP è che l'implementazione di MQTT è basata sul livello protocollare TCP/IP: questo semplifica di molto la struttura del protocollo in quanto TCP, contrariamente ad UDP, è un livello di trasporto affidabile che permette di ricevere i vari segmenti TCP come un flusso ordinato di Byte senza doversi preoccupare della gestione di eventuali segmenti persi o arrivati fuori ordine.

Inoltre, MQTT fornisce la possibilità di utilizzare per lo scambio di messaggi tre differenti tipologie di *Quality of Service* (QoS), ovvero degli accordi tra mittente e destinatari riguardanti la garanzia di consegna di un messaggio; i tre tipi di QoS che vengono messi a disposizione sono:

- **At most once:** è il livello minimo di garanzia di consegna, esso fornisce le stesse garanzie di consegna del protocollo TCP.
- **At least once:** in questo caso è garantito che al destinatario arrivi il messaggio almeno una volta; tuttavia, potrebbero esserci dei duplicati.
- **Exactly once:** con questo livello è garantito che il messaggio arrivi senza duplicati al destinatario.

2.1.2.1 Tipologie di Messaggi

Per implementare il modello *publish/subscribe* e la gestione della *Quality of Service*, il protocollo MQTT mette a disposizione una serie di tipologie di messaggi che i client, sia *publisher* che *subscriber*, possono scambiare con il broker. Di seguito sono descritti i vari messaggi disponibili:

- **CONNECT:** questo messaggio è inviato da un client al broker per inizializzare la connessione tra i due; la connessione tra un client ed il broker viene lasciata aperta finché il client non invia un messaggio per disconnettersi.
- **CONNACK:** questo è il messaggio di *acknowledgement* che il broker invia al client dopo aver ricevuto il messaggio CONNECT; esso contiene il codice di risposta che specifica se la connessione è stata stabilita o meno.
- **PUBLISH:** dopo aver stabilito una connessione con il broker, il client può pubblicare messaggi inerenti ad uno specifico *topic*. Questo messaggio contiene le informazioni riguardanti il *topic* ed il contenuto che il client vuole pubblicare.
- **PUBACK:** questo è il messaggio di *ack* inviato dal broker per notificare la ricezione di un messaggio PUBLISH. Questa tipologia di messaggi serve per implementare il meccanismo di QoS *at least once.*
- **PUBREC, PUBREL, PUBCOMP:** questi sono una serie di messaggi che vengono generati dal client e dal broker dopo l'invio di un messaggio di PUBLISH per implementare il meccanismo di QoS *exactly once.* Nella sezione successiva è illustrata nel dettaglio la sequenza di invio di questi messaggi.

- **SUBSCRIBE:** questo messaggio è inviato da un client al broker per sottoscriversi alla ricezione di messaggi di un determinato *topic.*
- **SUBACK:** questo tipo di messaggio viene inviato dal broker per notificare di aver ricevuto la sottoscrizione ad un *topic.*
- **UNSUBSCRIBE:** questo messaggio viene inviato dal client al broker per richiedere l'eliminazione della sottoscrizione ad uno o più *topic.*
- **UNSUBACK:** questo è l'*ack* inviato dal broker per confermare la ricezione del messaggio UNSUBSCRIBE.
- **PINGREQ:** questa tipologia di messaggio è un *ping* inviato dal client al broker. Tramite questo messaggio il client può sapere se il broker è ancora raggiungibile.
- **PINGRESP:** questo è il messaggio di risposta che invia il broker al client dopo la ricezione del PINGREQ.
- **DISCONNECT:** questo è il messaggio utilizzato dal client per indicare al broker la volontà di chiudere la comunicazione.

2.1.2.2 Flusso di Messaggi

Come accennato in precedenza, MQTT mette a disposizione diversi livelli di *Quality of Service* per garantire la consegna dei messaggi. Il fatto di poter scegliere diversi livelli di QoS è una delle principali caratteristiche del protocollo MQTT perché permette ai diversi nodi della rete di comunicare tra di loro utilizzando delle infrastrutture di rete poco affidabili.

Per garantire che un messaggio inviato arrivi al destinatario, il protocollo prevede la possibilità di inviare delle notifiche (*ack*) al mittente secondo il livello di qualità utilizzato durante la comunicazione. Di seguito vengono illustrati i messaggi scambiati tra client e broker a seconda del livello di QoS utilizzato.

Qos 0 - at most once: questo è il minimo livello di garanzia che è possibile utilizzare; i messaggi che sono inviati da un client non sono seguiti da un *ack* da parte del broker, per questo motivo non c'è il concetto di ritrasmissione dei messaggi. In questo caso la garanzia che un messaggio venga recapitato al broker segue la politica *best effort* offerta dal livello di rete TCP.

In Figura 2.5 è rappresentato il flusso di messaggi scambiati tra client e broker nel caso in cui il messaggio PUBLISH sia inviato con QoS al livello *at most once.*

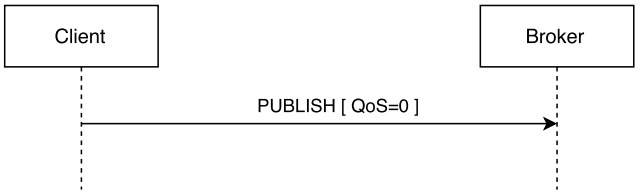


Figura 2.5: Flusso di messaggi nel caso in cui si utilizza QoS al livello *at most once*

Qos 1 - at least once: rispetto al caso precedente, dopo la ricezione di un messaggio PUBLISH, il broker deve inviare un messaggio di *ack* al client per confermare di aver ricevuto il messaggio. Se durante la comunicazione un messaggio o *l'acknowledgement* viene perso, il client - dopo lo scadere di un timeout - dovrà inviare nuovamente il messaggio. Utilizzando questo livello di QoS è garantito che al broker venga recapitato almeno un messaggio: nel caso in cui viene perso il messaggio di *ack* esso riceverà più volte lo stesso messaggio.

In Figura 2.6 viene mostrato il flusso di messaggi scambiati tra client e broker nel caso in cui venga effettuata una PUBLISH con QoS settata al livello *at least once*.

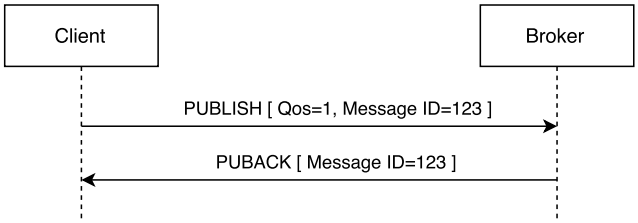


Figura 2.6: Flusso di messaggi nel caso in cui si utilizza QoS al livello *at least once*

Qos 2 - exactly once: questo livello di QoS garantisce che al broker non vengano consegnati messaggi duplicati; per fare ciò - come mostrato in Figura 2.7 - il client attraverso il messaggio PUBREL deve confermare la ricezione del messaggio di *ack* inviato dal broker. Ovviamente, utilizzando questo livello di QoS il canale di comunicazione sarà più trafficato; questo aspetto può essere accettato per le applicazioni che, per l'importanza del contenuto dell'informazione da trasmettere, richiedono che il messaggio sia recapitato correttamente al broker.

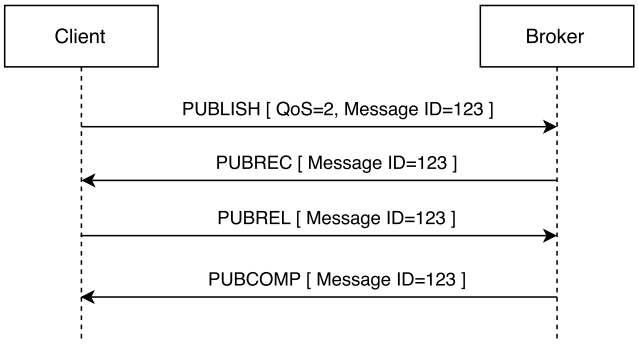


Figura 2.7: Flusso di messaggi nel caso in cui si utilizza QoS al livello *exactly once*

2.2 Aeromobili a Pilotaggio Remoto (APR)

Gli *Aeromobili a Pilotaggio Remoto* (APR), comunemente noti come droni, sono dei velivoli caratterizzati dall'assenza del pilota umano a bordo che volano autonomamente oppure sono pilotati a distanza da una stazione di controllo. Inizialmente questi aeromobili furono utilizzati a scopo militare: il primo tentativo di costruire un APR risale al 1849 quando gli Austriaci attaccarono la città di Venezia utilizzando dei palloni caricati di esplosivo [28]; successivamente durante le due guerre mondiali lo sviluppo di questi velivoli subì un forte impulso ma solo durante i conflitti del Vietnam, della guerra del Golfo e in Afghanistan, grazie allo sviluppo tecnologico, si riuscirono a produrre soluzioni compatte ed efficienti.

A differenza degli aerei tradizionali, gli APR possono essere utilizzati per raggiungere aree inaccessibili o impervie oppure in tutte quelle missioni caratterizzate da un elevato pericolo per la vita umana.

Con il termine APR si fa riferimento ad un'ampia gamma di velivoli che presentano caratteristiche tecnologiche differenti, le più importanti sono: il raggio operativo, la quota di volo, la durata del volo e il peso. Sulla base di questi parametri la Federazione Internazionale UVS (*Unmanned Vehicle System*) ha stilato la classificazione mostrata in Tabella 2.1 (tratta da [2]).

Come si può notare, esistono svariate tipologie di droni: si va dai velivoli più piccoli che pesano qualche decina di grammo, hanno un'autonomia di volo inferiore all'ora ed un raggio operativo inferiore al chilometro, agli APR più grandi che possono avere un peso superiore alle dieci tonnellate, avere un'autonomia di volo di oltre un giorno, raggiungere quote di volo superiori a 10000 metri e operare a più di 2000 chilometri di distanza dalla stazione di controllo.

Per questo lavoro di tesi sono stati considerati gli aeromobili più piccoli - come quelli classificati Nano, Micro e Mini - in quanto, per motivi di carattere economico,

sono quelli destinati ad applicazioni civili; gli APR più grandi sono invece utilizzati in applicazioni militari.

Tabella 2.1: Classificazione degli APR

Categoria	**Raggio operativo [km]**	**Quota di volo [m]**	**Durata del volo [h]**	**Peso [Kg]**
Tactical UAV				
Nano	< 1	100	< 1	$< 0{,}0250$
Micro	< 10	250	1	< 5
Mini	< 10	150 - 300	< 2	< 30
Close Range	10 - 30	3 000	2 - 4	150
Short Range	30 - 70	3 000	3 - 6	200
Medium Range	70 - 200	5 000	6 - 10	1 250
Medium Range Endurance	> 500	8 000	10 - 18	1 250
Low Altitude Deep Penetration	> 250	50 - 9 000	0,5 - 1	350
Low Altitude Long Endurance	> 500	3 000	> 24	< 30
Medium Altitude Long Endurance	> 500	14 000	24 - 48	1500
Strategic UAV				
High Altitude Long Endurance	$> 2\,000$	20 000	24 - 48	12 000
Special purpose UAV				
Unmanned combat aerial vehicle	1 500	10 000	2	10 000
Lethal	300	4 000	3 - 4	250
Decoy	$0 - 500$	5 000	< 4	250
Stratospheric	$> 2\,000$	$> 20\,000$ & $< 30\,000$	> 48	Da definire
Exo – stratospehric	Da definire	$< 30\,000$	Da definire	Da definire
Space	Da definire	Da definire	Da definire	Da definire

2.2.1 Sistemi di Pilotaggio Remoto

Gli aeromobili a pilotaggio remoto costituiscono il componente principale di un ampio sistema denominato *Sistema di Pilotaggio* R*emoto* o *Unmanned Aircraft System* (UAS) [7] il quale, per poter operare correttamente, necessita di una stazione di controllo, un payload, un sistema di navigazione, un sistema di comunicazione ed un equipaggiamento di lancio e recupero (Figura 2.8).

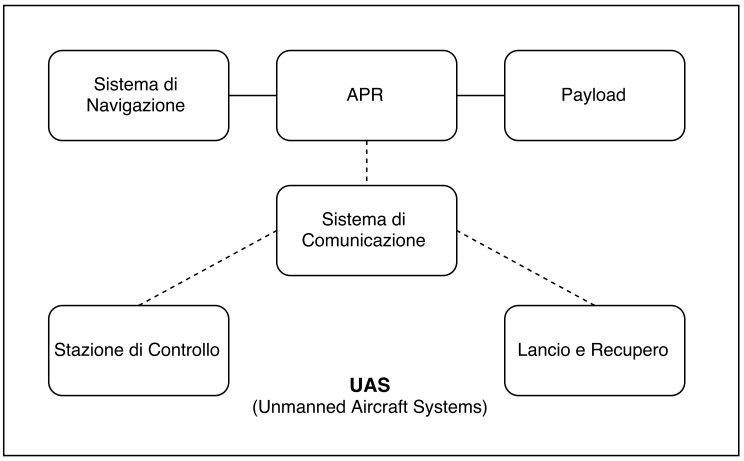


Figura 2.8: Principali componenti che costituiscono un UAS (*Unmanned Aircraft System*)

Stazione di Controllo

È il centro di controllo della missione e l'interfaccia tra l'uomo ed il drone. Dalla stazione di controllo gli operatori possono pianificare e monitorare l'andamento della missione eseguita dall'APR. Generalmente si trova a terra e può essere costituita da piccole attrezzature portatili oppure può assumere dimensioni simili alle stazioni di controllo degli aerei di linea.

Ad esempio, per l'APR utilizzato durante questo lavoro di tesi (Sezione 5.3), la stazione di controllo è costituita da un laptop con installato a bordo il software Mission Planner tramite il quale è possibile monitorare tutti i dati di volo provenienti dal drone ed inviare vari comandi per pilotarlo. In Figura 2.9 viene illustrata una schermata di Mission Planner.

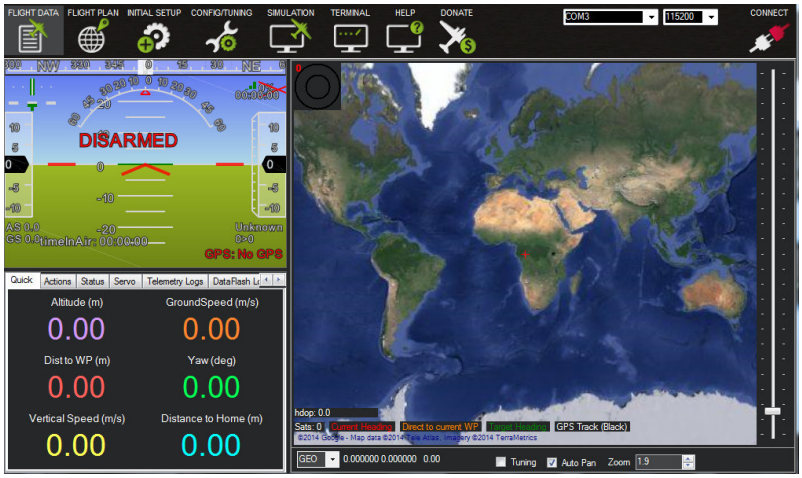


Figura 2.9: Schermata principale del software Mission Planner

Payload

Il payload è l'equipaggiamento a bordo del velivolo che serve per svolgere la missione; esso può essere costituito da sensori di vario genere (umidità, temperatura, pressione atmosferica, ...), videocamere oppure da attrezzature più complesse come radar.

Sistema di Navigazione

Ha il compito di tracciare posizione, orientamento e velocità dell'aeromobile. Per ricavare questi dati, il sistema di navigazione si può avvalere di un sistema di posizionamento globale come il GPS oppure può essere costituito da sistemi inerziali che sfruttano i dati provenienti da accelerometri e giroscopi per calcolare la posizione dell'aeromobile.

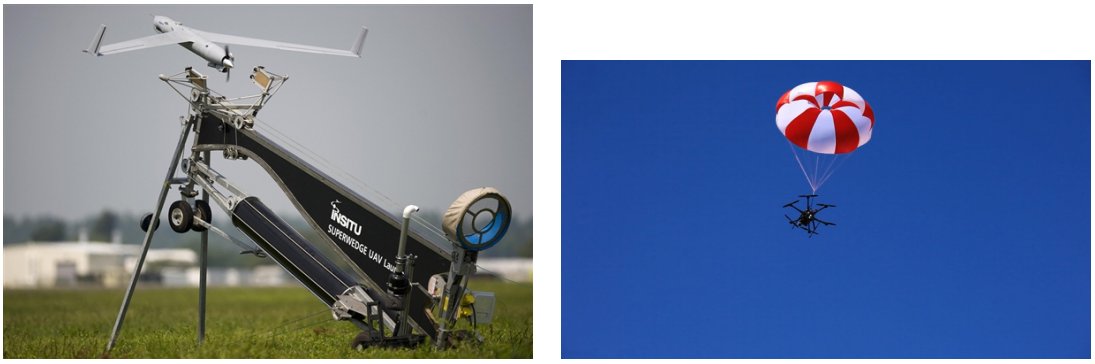
Sistema di Comunicazione

Il suo obiettivo è quello di permettere la comunicazione tra la stazione di controllo ed il velivolo; il mezzo trasmissivo più utilizzato è costituito dalle radiofrequenze ma esistono anche dei sistemi di comunicazione satellitari che permettono di avere una copertura più ampia.

Equipaggiamento di Lancio e di Recupero

L'equipaggiamento di lancio è necessario per quegli aeromobili che non hanno la capacità di volare verticalmente; di solito esso è costituito da una rampa dove il velivolo viene accelerato fino alla velocità necessaria per sostenere il volo. L'equipaggiamento di recupero è utilizzato in quelle situazioni in cui non è possibile effettuare un normale atterraggio; di norma è costituito da paracaduti e da airbag necessari ad

assorbire gli impatti. In Figura 2.10 è mostrato l'esempio di un equipaggiamento di lancio e di un equipaggiamento di recupero.



(a) Equipaggiamento di lancio (b) Equipaggiamento di recupero

Figura 2.10: Esempi di equipaggiamenti di lancio e di recupero

2.2.2 Utilizzi

Nei primi anni di sviluppo della tecnologia la maggior parte degli APR sono stati utilizzati per scopi militari. In genere gli aeromobili utilizzati per scopi bellici sono attrezzati con armamenti o, più semplicemente, con sensori di ripresa che permettono l'invio in tempo reale di immagini alla stazione di controllo posta a decine di chilometri di distanza.
Negli ultimi anni è in continuo aumento l'impiego degli APR anche in ambito civile. Le missioni che essi possono svolgere dipendono dalle caratteristiche degli aeromobili utilizzati e dai sensori installati a bordo. Di seguito verranno descritti alcuni esempi di utilizzo dei droni in ambito civile.

2.2.2.1 Operazioni di Ricerca e Soccorso

Come studiato nell'ambito del progetto SHERPA [40], coordinato dall'Università di Bologna, gli APR possono svolgere un ruolo importante durante le operazioni di ricerca e soccorso in ambienti ostili, come può essere quello alpino.

Il progetto SHERPA è nato per proporre una soluzione che permettesse di minimizzare i tempi di soccorso in caso di valanghe; in questo scenario, la celerità dell'intervento è strategica per salvare i dispersi, in quanto una persona travolta da una valanga può sopravvivere solo per pochi minuti. Il progetto prevede l'utilizzo di APR di piccole dimensioni dotati di ricevitore ARTVA [1] per sorvolare la zona interessata dalla valanga. I test condotti hanno mostrato che un drone è in grado

[1] ARTVA - Apparecchio di Ricerca dei Travolti in VAlanga, è uno strumento ricetrasmittente che viene utilizzato prevalentemente in modalità trasmittente e commutato in ricezione nel momento in cui occorre cercare dispersi; la localizzazione dei dispersi avviene grazie al fatto che il segnale è più intenso man mano che ci si avvicina allo strumento in trasmissione.

di identificare un segnalatore ARTVA nascosto in una zona dal raggio di 300 metri sotto un metro di neve in meno di un minuto: è stata quindi evidenziata l'efficacia di tale mezzo in questo contesto; inoltre, in aggiunta al breve tempo di ritrovamento dei dispersi, l'utilizzo degli APR permette di ridurre i rischi legati all'ambiente di lavoro per i soccorritori impegnati nelle attività di ricerca e diminuire drasticamente i costi rispetto all'utilizzo di un elicottero per sondare la zona dell'evento - in caso di valanga, l'elisoccorso aiuta il personale a terra calando un'antenna in grado di ricevere il segnale emesso dagli ARTVA e sorvolando così il luogo in cui vengono condotte le ricerche.

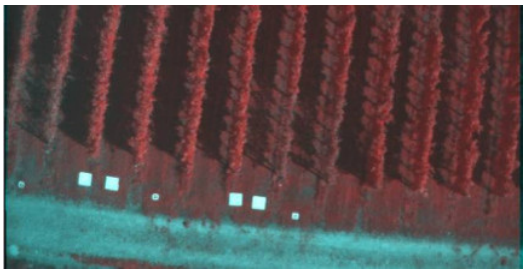
2.2.2.2 Monitoraggio Ambientale

Grazie alla possibilità di volare a quote molto basse ed al costo piuttosto accessibile, gli APR vengono sempre più spesso utilizzati per svolgere applicazioni di monitoraggio ambientale quali la creazione di mappe di vigore di colture agricole [30], la creazione di cartografie dell'uso del suolo, il monitoraggio delle attività forestali illegali [33] e l'individuazione e la mappatura della presenza di amianto e di dispersioni termiche nelle copertura degli edifici come nel caso del progetto «Progetto Amianto e Termografia» avviato dal Comune di Fidenza.

Un esempio di monitoraggio ambientale tramite l'uso dei droni è quello descritto in uno studio pubblicato dall'Università di Scienze Applicate del Nordovest della Svizzera nel 2008 [30] che ha mostrato l'ottimo risultato ed i vantaggi che derivano dall'utilizzo di APR nella creazione di mappe di vigore delle colture agricole.

Le tecniche di telerilevamento si sono da sempre mostrate molto utili per monitorare lo stato di salute della vegetazione: elaborando immagini riprese nell'infrarosso, grazie alla riflessività della clorofilla a queste lunghezze d'onda, è possibile distinguere piante sane - che possiedono una maggiore quantità di clorofilla e pertanto riflettono maggiormente l'infrarosso - da piante malate o in sofferenza - che hanno una minore concentrazione di questo elemento e quindi sono meno riflettenti (Figura 2.11a). Tuttavia, fino ad oggi, le immagini erano riprese da aerei o satelliti; tale mezzi hanno delle forti limitazioni quali il costo e il massimo livello di dettaglio raggiungibile.

L'utilizzo degli APR, come quello mostrato in Figura 2.11b, per svolgere questo tipo di applicazione permette di superare queste limitazioni fornendo uno strumento relativamente semplice da utilizzare e in grado di acquisire immagini con una risoluzione spaziale e temporale più elevata. In particolare, la maggiore risoluzione spaziale permette di identificare il livello di vigore delle colture in maniera quasi equivalente a quello determinato dal personale specializzato tramite osservazioni dirette sul campo.



(a) Piantagione ripresa con sensore multi-spettrale; le aree rosse sono le più attive vegetativamente, seguite da quelle gialle e poi dalle verdi.

(b) APR *microdrones md4-200* con sensore multi-spettrale utilizzato dall'Università di Scienze Applicate del Nordovest della Svizzera.

Figura 2.11: APR per il monitoraggio ambientale

2.2.2.3 Aerofotogrammetria

La fotogrammetria è una tecnica di rilievo che permette di ottenere dati metrici di un oggetto tramite l'acquisizione e l'analisi di immagini. È utilizzata principalmente in architettura, per rilevare la forma, le dimensioni e il posizionamento di un oggetto architettonico, e per il rilevamento topografico del territorio; nel primo caso vengono di norma utilizzate immagini riprese da terra, nel secondo caso si ricorre soprattutto ad immagini acquisite da aerei o satelliti. I rilievi fotogrammetrici sono impiegati poi per la creazione di modelli 3D di edifici, modelli digitali del terreno ed orto-foto[2].

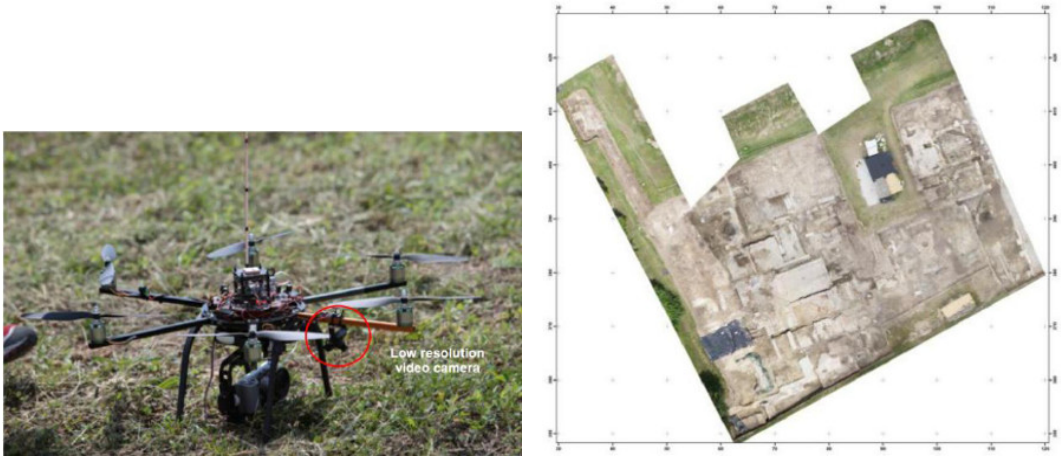
Come descritto in [31], la diffusione sempre più ampia degli APR ha aperto nuove strade anche nell'ambito della fotogrammetria, introducendo un'alternativa economica alla fotogrammetria aerea e una soluzione complementare alle acquisizioni da terra.

L'archeologia è uno dei campi di applicazione più interessanti per l'utilizzo di droni nella fotogrammetria [31,38]. Durante le campagne di scavi è molto importante monitorare con accuratezza le nuove aree scavate e i ritrovamenti compiuti, sia per poter pianificare le attività da svolgere giornalmente sia per scopi di documentazione e conservazione. Grazie al costo ridotto, alla velocità di acquisizione delle immagini e all'alta risoluzione spaziale garantita dalla possibilità di volare a basse quote, gli APR possono efficacemente essere utilizzati nel monitoraggio, anche giornaliero.

Questo specifico utilizzo degli APR è stato dimostrato essere efficace da una campagna di test svolta nel sito archeologico *Domus dei putti danzanti* ad Aquileia (UD) [38]. Il test è stato svolto impiegando un drone (Figura 2.12a) che, seguendo un piano di volo pre-programmato, ha acquisito in breve tempo decine di fotografie

[2] orto-foto: un orto-foto è una fotografia aerea che è stata geometricamente corretta e georeferenziata in modo tale che la scala di rappresentazione sia uniforme. In questo modo la foto può essere considerata equivalente ad una mappa così da poter essere utilizzata per misurare distanze reali.

aeree dell'intero sito archeologico; le immagini acquisite durante il volo sono state poi processate da un software specifico al fine di produrre l'orto-mappa del sito, mostrata in Figura 2.12b.

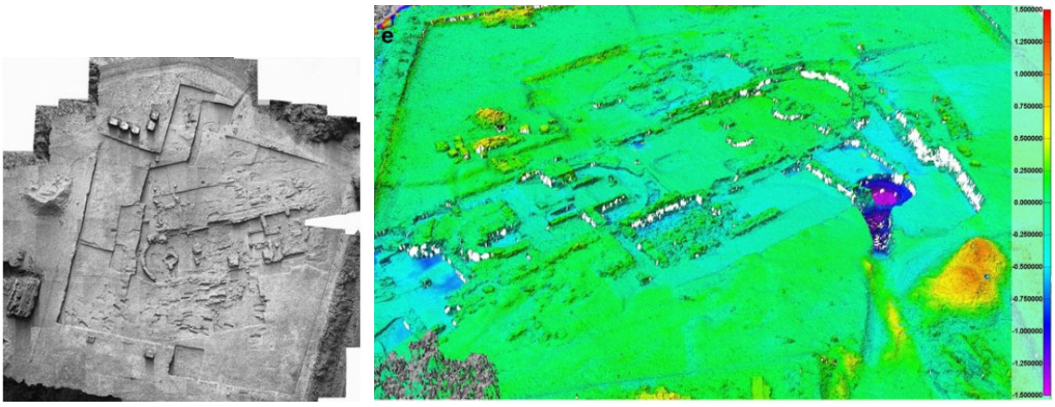


(a) APR utilizzato negli scavi archeologici *Domus dei putti danzanti*

(b) Orto-mappa del sito archeologico di Aquileia

Figura 2.12: Aerofotogrammetria nel sito archeologico *Domus dei putti danzanti* ad Aquileia (UD)

Un altro esempio, descritto in [31], è l'utilizzo di un drone nell'area archeologica di Pava (SI): ogni anno, all'inizio e alla fine del periodo di scavi, vengono compiuti dei voli durante i quali sono raccolte immagini utilizzate per creare modelli digitali della superficie (DSM) (Figura 2.13a) ed orto-foto del sito. La disponibilità di DSM relativi a diversi periodi temporali permette anche di calcolare una stima del volume del materiale estratto durante gli scavi (Figura 2.13b).



(a) Modello digitale della superficie (DSM) del sito

(b) Modello digitale multi-temporale della superficie: le aree blu mostrano dove è stato asportato del materiale, mentre quelle rosso-gialle indicano dove tale materiale è stato accumulato.

Figura 2.13: Aerofotogrammetria nell'area archeologica di Pava (SI)

2.2.2.4 Monitoraggio di Aree Colpite da Calamità Naturali

L'obiettivo principale dopo una calamità naturale (terremoto, alluvione o frana) è quello di produrre dati geo-referenziati sulle zone colpite per sostenere l'azione umanitaria. Le informazioni essenziali da conoscere per organizzare al meglio i soccorsi sono: le aree colpite, il numero di persone coinvolte e lo stato di sicurezza degli edifici. Le immagini satellitari non sono sufficienti per soddisfare queste necessità perché molto spesso hanno scale troppo grandi per fare delle analisi accurate del territorio; inoltre, queste immagini vengono rese disponibili solo dopo qualche giorno dall'evento catastrofico. Gli APR invece sono perfetti in queste circostanze perché in poco tempo possono sorvolare le aree coinvolte ed acquisire immagini molto più dettagliate rispetto alle riprese satellitari. Inoltre, tutti i dati raccolti dai droni possono essere utilizzati immediatamente dai soccorritori.

Per questo motivo l'associazione ITHACA - *Information Technology for Humanitarian Assistance, Cooperation and Action* [10] in collaborazione con il Politecnico di Torino ha realizzato un APR di basso costo per svolgere questo tipo di missioni. Il velivolo sviluppato (Figura 2.14a) è in grado di volare autonomamente ed è equipaggiato con sensori digitali per l'acquisizione di immagini e video. Il sistema di navigazione include un'unità GPS, un giroscopio a tre assi, un accelerometro ed un sensore ad ultrasuoni per rilevare l'altitudine dal suolo. La stazione di controllo (Figura 2.14b) è invece composta da un laptop con installato il software di controllo HORIZON in grado di programmare piani di volo specificando un lista di *waypoints* - ovvero una lista coordinate GPS che il drone deve raggiungere in sequenza. Dalla stazione di controllo è inoltre possibile leggere in tempo reale i valori di tutti i sensori installati a bordo.
Questo sistema di pilotaggio remoto è in grado di fornire in tempi rapidi una cartografia delle zone soggette a calamità ed inviare in tempo reale le immagini registrate dalla fotocamera alla stazione di controllo.

(a) Velivolo sviluppato per il progetto ITHACA

(b) Stazione di controllo utilizzata nel progetto ITHACA

Figura 2.14: APR per il monitoraggio di aree colpite calamità da naturali

2.2.2.5 Meteorologia

Le misurazioni compiute tramite mezzi aerei sono molto utilizzate nell'ambito degli esperimenti meteorologici, grazie alla possibilità di investigare grandi aree in tempi relativamente brevi. Come in altri ambiti, anche nel caso della meteorologia, l'utilizzo di droni presenta dei vantaggi rispetto all'uso di velivoli con pilota a bordo. Soprattutto, gli APR risultano più semplici da manovrare, hanno costi di gestione e operativi nettamente inferiori e possono sorvolare aree normalmente interdette o rischiose; inoltre, l'utilizzo simultaneo di più APR, abilitato anche dal costo ridotto di ogni singolo velivolo, permette di incrementare drasticamente l'area di analisi rispetto agli aeromobili tradizionali. Tuttavia, considerando gli APR maggiormente utilizzati in ambito civile, cioè quelli di tipo Micro o Mini, si possono evidenziare anche degli svantaggi rispetto ai velivoli con pilota a bordo; i maggiori sono certamente le limitazioni per quanto riguarda le dimensioni ed il peso del payload trasportabile dai suddetti APR, che spesso costringono ad utilizzare sensori con caratteristiche inferiori.

L'Istituto di Sistemi Aerospaziali dell'Università Tecnica di Braunschweig ha però mostrato, negli studi compiuti tramite prototipi di aeromobili da loro sviluppati [12, 41](Figura 2.15), che è comunque possibile raccogliere dati di buona qualità, nonostante i limiti attuali. In particolare, gli APR sono stati impiegati in missioni che avevano come obiettivo quello di raccogliere dei parametri atmosferici fondamentali come la pressione, la temperatura, l'umidità e le componenti tridimensionali del vento ad una altitudine compresa tra i dieci e i mille metri. Le missioni compiute durante i test sono state pianificate dagli utenti prima del decollo, impostando per mezzo della stazione di controllo a terra una serie di *waypoints* su una mappa e determinando di conseguenza il piano di volo dell'APR. Una volta caricato il piano di volo, i velivoli dovevano risultare totalmente autonomi, dovevano cioè poter compiere la missione specificata anche in assenza di contatto radio con la stazione di controllo a terra; tale caratteristica è essenziale per permettere agli APR di coprire aree molto vaste, così come necessario per gli studi meteorologici. I dati campionati sono stati archiviati su una memory card installata a bordo del drone, che è stata poi recuperata e letta una volta che l'APR è atterrato.

(a) Carolo P50 (b) Carolo T140

Figura 2.15: APR usati per studi meteorologici

2.2.2.6 Ispezione di Oleodotti e Gasdotti

Oltre agli esempi di utilizzo degli APR fino a qui descritti, un'altra applicazione di grande interesse è il monitoraggio di oleodotti e gasdotti al fine di identificare eventuali perdite di idrocarburi.

In tutto il mondo vi sono più di tre milioni di chilometri di condotte per il trasporto di idrocarburi; a causa di danni nella rete possono essere disperse enormi quantità di gas o petrolio con conseguenti gravi danni ambientali e perdite economiche. Pertanto, sviluppare ed implementare dei sistemi di monitoraggio che verifichino continuamente lo stato delle condotte è essenziale.

Ad oggi, le soluzioni maggiormente utilizzate sono rappresentate da controlli visivi effettuati attraverso ispezioni periodiche da terra o per mezzo di aerei leggeri od elicotteri. Tali soluzioni sono estremamente costose e hanno lo svantaggio di individuare eventuali perdite solitamente in maniera tardiva. Gli APR vengono quindi considerati un'alternativa allettante rispetto ai sistemi di monitoraggio tradizionali. L'impiego dei droni per l'ispezione degli oleodotti e dei gasdotti [8] prevede che i velivoli, equipaggiati con telecamere termiche, vengano fatti volare autonomamente lungo le condotte; eventuali perdite sono facilmente identificabili dalle immagini raccolte grazie alla differenza di temperatura tra gli idrocarburi fuoriusciti dalle condotte e il suolo circostante. Alcuni dei vantaggi che derivano dall'utilizzo di APR sono la riduzione dei costi operativi e l'aumento della sicurezza delle missioni di ispezione.

2.2.3 Piattaforme Software per APR

Dagli esempi appena citati emerge che gli APR, per raggiungere gli obiettivi della missione, sono spesso pilotati a distanza tramite un radiocomando oppure devono seguire piani pre-programmati. Questo aspetto distingue gli APR dai robot. Gli APR infatti non devono esprimere comportamenti *human like* e non devono avere capacità di *machine learning* al fine di apprendere informazioni in maniera automatica dall'ambiente in cui operano. Per tale motivo gli APR, diversamente dai

robot, hanno limitate capacità decisionali poiché sono utilizzati prevalentemente per sorvolare un'area al fine di acquisire dati utili agli operatori umani.

Inoltre, nella maggioranza delle applicazioni in cui si utilizzano i droni non è necessario utilizzare più di un velivolo per portare a termine gli obiettivi della missione quindi non sorgono problematiche relative al coordinamento tra gli APR e non è necessario utilizzare tecniche di *swarm behaviour* [11] per coordinare i velivoli durante il volo.

Per questi motivi, le piattaforme software utilizzate dagli APR per la gestione del volo sono diverse dalla quelle utilizzate dalla robotica tradizionale. Infatti, per gestire gli APR non è necessario utilizzare delle piattaforme come ROS - *Robot Operating System* [36], le quali permettono di avere un sistema completamente autonomo e capace di cooperare con altri robot al fine di raggiungere degli obiettivi comuni, ma è sufficiente utilizzare dei controllori più semplici, come ArduCopter [5] (Sezione 5.1), in grado di garantire agli APR la capacità di volare, comunicare con la stazione di controllo ed eseguire dei semplici compiti in maniera automatica, ad esempio, l'esecuzione di piani di volo specificati tramite una lista di *waypoints*.

2.3 Gap nello Stato dell'Arte

Quello che emerge dagli esempi di applicazioni mostrati nella Sezione 2.2.2 è che spesso la programmazione degli APR avviene tramite la specifica di una serie di *waypoints*; questa tecnica fa sì che il velivolo sia poco intelligente e non permette l'utilizzo di tecniche di *Active Sensing* per gestire la navigazione. In tal modo gli APR non hanno la capacità di modificare dinamicamente il piano di volo in base ai valori letti dai sensori al fine di raggiungere autonomamente gli obiettivi della missione.

Inoltre, non è possibile far volare gli APR senza l'ausilio di una stazione di controllo o di un radiocomando. Infatti, ad oggi, non esistono dei sistemi che permettono il controllo degli APR mediante l'utilizzo di interfacce basate su protocolli Internet. Questo aspetto non permette l'utilizzo dei droni nel contesto dell'IoT limitando quindi le loro possibilità di interfacciamento con dispositivi e sistemi informatici.

Capitolo 3

Modello di Programmazione

In questo capitolo viene presentato un modello per la programmazione e l'esecuzione delle missioni per far sì che gli APR possano superare i limiti imposti dalla navigazione tramite *waypoints*. Lo scopo di questo modello è quello di proporre una serie di astrazioni per la gestione delle missioni che permettano ai droni di essere autonomi ed intelligenti; in particolare, vogliamo che gli APR possano utilizzare tecniche di *Active Sensing* per gestire la navigazione, ovvero che i velivoli possano interagire con i sensori installati a bordo al fine modificare dinamicamente il proprio piano di volo a seconda degli obiettivi della missione. Questo approccio supera lo stato dell'arte attuale in quanto i droni non sono più costretti a seguire dei piani pre-programmati, come accadeva nel caso di programmazione con *waypoints*, ma sono in grado di modificare il proprio piano di volo durante l'esecuzione della missione grazie alla capacità di elaborare in tempo reale i dati acquisiti dai sensori.

Inoltre, il modello da noi proposto deve poter semplificare l'integrazione degli APR nel contesto dell'IoT così da garantire ai velivoli di essere indipendenti dalla stazione di controllo. A tal proposito vogliamo che i velivoli possano offrire delle interfacce basate su protocolli Internet per permettere a qualunque dispositivo connesso alla rete di pilotare gli APR.

In questo capitolo focalizzeremo la nostra attenzione sugli aspetti legati al modello astratto per permettere agli APR di utilizzare tecniche di *Active Sensing* per la gestione della missione; le problematiche relative all'integrazione degli APR nell'ambito dell'IoT sono discusse nel Capitolo 4.

Inizialmente siamo andati a suddividere i vari utilizzi degli APR in ambito civile in tre classi di applicazioni. Per ogni classe abbiamo definito gli obiettivi e gli aspetti principali che le caratterizzano.

Successivamente abbiamo illustrato il modello di programmazione proposto discutendo come lo si può impiegare nelle varie applicazioni appartenenti alle classi

individuate.

Infine abbiamo presentato degli esempi di come questo modello si può utilizzare in alcuni casi reali.

3.1 Applicazioni

Partendo dagli esempi di utilizzo degli APR illustrati al Capitolo 2, abbiamo classificato le varie tipologie di missioni nelle seguenti categorie: campionamento, ricerca ed inseguimento.

Tale classificazione è stata di ausilio per la definizione del modello di programmazione degli APR, illustrato nella Sezione 3.2, poiché ci ha permesso di avere una visione globale dei requisiti che le varie applicazioni necessitano per soddisfare i propri obiettivi.

Di seguito vengono illustrate le classi di applicazioni che abbiamo individuato, esse sono presentate in ordine crescente rispetto al livello di *Active Sensing* richiesto da ciascuna di esse.

3.1.1 Campionamento

In questa classe ricadono tutte le applicazioni in cui il drone è utilizzato per trasportare i sensori installati a bordo in luoghi di particolare interesse per l'utente con l'obiettivo di acquisire informazioni riguardanti l'ambiente o il territorio sorvolato. Normalmente i dati acquisiti durante la missione vengono salvati su dispositivi installati a bordo dell'APR o vengono trasmessi in tempo reale alla stazione di controllo; l'elaborazione delle informazioni raccolte viene effettuata a *missione conclusa*.

Generalmente, i dati acquisiti vengono poi utilizzati per fare statistiche, previsioni, costruzioni di mappe e modelli 3D.

Nella maggioranza delle applicazioni le informazioni raccolte sono immagini, valori di temperatura, pressione e umidità dell'aria; in alcuni casi sui droni sono installati sensori più sofisticati come ad esempio quelli per valutare la presenza di polveri sottili o per monitorare la presenza di gas inquinanti nell'atmosfera [20].

In questa categoria ricadono le applicazioni di: monitoraggio ambientale, aerofotogrammetria, monitoraggio di aree colpite da calamità naturali e meteorologia citate nel Capitolo 2 di questo elaborato. È importante osservare che ciò che accomuna queste applicazioni è il fatto che il percorso che segue l'APR durante l'esecuzione della missione non è condizionato dai dati letti dai sensori utilizzati per lo svolgimento della missione stessa: infatti, per svolgere questo tipo di applicazioni non è necessario utilizzare tecniche di *Active Sensing*. Come si può notare nel caso dell'applicazione in cui il drone veniva utilizzato in meteorologia (Sezione 2.2.2.5), il percorso che doveva compiere l'aeromobile veniva programmato specificando una lista di punti da seguire, mentre l'acquisizione dei dati atmosferici veniva fatta da un processo parallelo.

3.1.2 Ricerca

Queste tipologie di missioni si prefiggono l'obiettivo di sorvolare una determinata area per verificare la presenza di oggetti, persone o anomalie.

Esempi di applicazioni elencate al Capitolo 2 di questo elaborato che ricadono in questa categoria sono l'ispezione di oleodotti per verificare perdite di sostanze inquinanti e tutte le applicazioni in cui gli APR sono utilizzati per operazioni di ricerca e soccorso di persone disperse, come la ricerca di sciatori travolti da valanghe.

A differenza delle missioni di campionamento, i dati acquisiti durante il volo devono essere analizzati *immediatamente* per valutare se la ricerca è andata a buon fine o meno: per questo motivo è necessario utilizzare tecniche di *Active Sensing* durante la navigazione. L'analisi delle informazioni raccolte durante la missione può essere fatta a bordo dell'aeromobile oppure a terra; nel primo caso l'APR dovrà solamente comunicare all'utente la posizione dell'oggetto qualora questo venga individuato all'interno dell'area di ricerca, nel secondo caso i dati raccolti dai sensori utilizzati per effettuare la ricerca dovranno essere inviati in tempo reale alla stazione di controllo la quale li analizzerà per verificare se l'obiettivo è stato trovato.

La maggioranza di queste missioni terminano non appena la ricerca ha esito positivo; si pensi alla ricerca di un disperso sotto una valanga: non avrebbe senso continuare la perlustrazione una volta che la vittima è stata individuata. Altre applicazioni, come l'ispezione di oleodotti per verificare la presenza di perdite, necessitano invece che l'APR utilizzato sorvoli l'intera area per verificare la presenza di uno o più riscontri.

I sensori più utilizzati nelle operazioni di ricerca sono camere digitali e ricevitori di segnali radio. Nel caso di camere digitali per capire se l'obiettivo è stato identificato vengono analizzate le immagini raccolte utilizzando degli algoritmi di riconoscimento di oggetti; per quanto riguarda i segnali radio quello che in genere si vuole trovare è il punto in cui la potenza del segnale è più alta, come nel caso della ricerca di dispersi sotto le valanghe (Sezione 2.2.2.1) dove lo scopo è quello di identificare il luogo dove il segnale emesso dal dispositivo ARTVA posseduto dalla vittima è maggiore.

3.1.3 Inseguimento

Lo scopo delle missioni di inseguimento è quello di utilizzare gli APR per identificare un obiettivo e successivamente seguire tutti gli spostamenti che questo compie. Esempi di questo tipo di applicazione sono la ripresa dall'alto di soggetti che praticano attività sportive oppure il tracciamento degli spostamenti di alcune specie animali per studiare il loro comportamento [14].

I dati acquisiti durante il volo possono essere salvati su dispositivi di archiviazione installati a bordo dell'APR oppure possono essere trasmessi in tempo reale alla stazione di controllo.

Le tecniche usate per far sì che l'aeromobile segua ogni spostamento dell'obiettivo sono tendenzialmente due: la prima consiste nell'equipaggiare l'obiettivo mobile con un dispositivo che notifica continuamente la sua posizione all'APR, quest'ultimo dovrà solamente spostarsi nel punto indicato; la seconda è quella di utilizzare le immagini acquisite da una telecamera installata a bordo dell'APR per rilevare gli spostamenti dell'obiettivo e far muovere l'aeromobile di conseguenza.

Analogamente al caso precedente anche per svolgere questo tipo di applicazioni è necessario utilizzare tecniche di *Active Sensing* durante la navigazione perché è necessario identificare la posizione dell'obiettivo da seguire prima di far spostare il velivolo.

3.2 Gestione delle Missioni

Ad oggi, la gestione degli APR richiede l'utilizzo di una stazione di controllo per programmare i percorsi di volo da far eseguire ai velivoli e per analizzare i dati raccolti durante la missione.

La programmazione del piano di volo di solito consiste nella definizione di una lista di punti, detti *waypoints*, che il velivolo deve seguire in sequenza per formare un percorso; come si può intuire, con questo approccio il drone è poco intelligente: esso infatti deve solamente eseguire dei piani pre-programmati e non ha la capacità di raggiungere autonomamente gli obiettivi della missione.

Inoltre, l'utilizzo della stazione di controllo impone grandi limiti per quanto riguarda l'integrazione degli APR con altri sistemi informatici: essa, infatti, è pensata per semplificare l'interazione tra l'uomo ed il velivolo e non per l'interazione tra il velivolo e sistemi informatici.

Questi limiti vogliono essere superati dal modello di programmazione per la gestione degli APR proposto di seguito, i cui obiettivi sono:

- rendere i velivoli più autonomi: il drone deve utilizzare tecniche di *Active Sensing,* ovvero deve avere la possibilità di interagire con i sensori installati a bordo per poter modificare dinamicamente il proprio piano di volo al fine di raggiungere gli obiettivi della missione. Così facendo il drone risulta intelligente ed in grado di gestire il volo in maniera autonoma.

- rendere il drone un componente della IoT: la programmazione della logica di gestione della missione e l'accesso ai dati raccolti dai sensori deve essere effettuata mediante l'utilizzo dei protocolli di rete utilizzati nel contesto dell'IoT.

3.2.1 Componenti

Per soddisfare gli obiettivi sopra elencati, abbiamo deciso di proporre un modello per la gestione di volo in cui la logica applicativa per il controllo dell'APR è suddi-

visa in due componenti: *1) modulo di navigazione; 2) modulo di campionamento ed elaborazione dati.* Tale scelta è stata fatta per rendere il drone flessibile: in tal modo è infatti possibile riutilizzare parte della logica applicativa per svolgere più tipi di missioni.

Navigazione

Il modulo di navigazione contiene le istruzioni che specificano come si deve muovere l'aeromobile mentre la missione è in esecuzione. Queste istruzioni possono riguardare vari aspetti del volo come: il decollo, l'atterraggio, le coordinate GPS che devono essere raggiunte, le aree che devono essere perlustrate, le traiettorie che devono essere seguite, le quote che devono essere raggiunte oppure gli assetti che devono essere mantenuti.

Il modulo di navigazione deve quindi poter agire sugli attuatori che permettono al drone di volare; dato che la maggioranza dei velivoli è dotata di controllori di stabilità necessari per il volo, per poter agire sugli attuatori bisognerà interfacciarsi con questi controllori che solitamente sono componenti hardware forniti dal produttore dell'aeromobile. In Figura 3.1 è illustrato un esempio di interfacciamento tra il modulo di navigazione e l'APR mediante il controllore di stabilità.

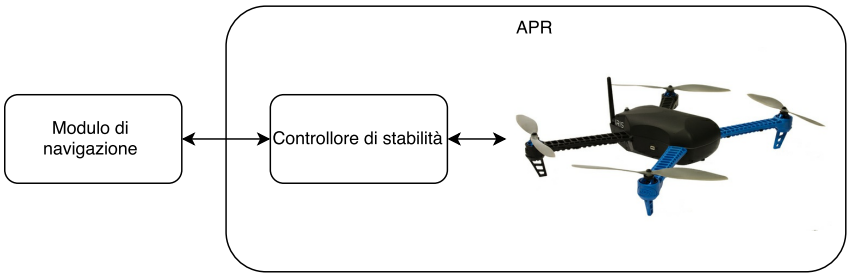


Figura 3.1: Interfacciamento tra modulo di navigazione e APR

Per far sì che il drone si comporti nella maniera specificata dall'utente è importante che, durante l'esecuzione di una missione, il modulo di navigazione sia l'unico componente in grado di inviare al controllore di stabilità comandi per modificare l'assetto di volo del velivolo.

Dato che, come detto precedentemente, uno degli obiettivi di questo lavoro è quello di permettere agli APR di modificare il piano di volo a seconda dei valori letti dai sensori durante la missione, ovvero di utilizzare tecniche di *Active Sensing*, è importante che il modulo di navigazione possa interfacciarsi con tutti i sensori installati a bordo del drone. In questo modo l'utente può creare dei piani di volo dinamici che permettono al velivolo di volare a seconda dei dati letti dai sensori utilizzati per lo svolgimento della missione.

Di seguito viene illustrato come varia il comportamento del modulo di navigazione nelle tre classi di applicazioni individuate nella Sezione 3.1:

- **Missioni di campionamento:** in questo caso, il modulo di navigazione deve solamente far volare l'APR nei punti indicati dall'utente, cioè deve permettere la navigazione tramite *waypoints*. Non è necessario che la navigazione dipenda dai valori letti dai sensori durante la missione (*Active Sensing*) perché il velivolo è solo il mezzo che l'utente utilizza per portare i vari sensori nei punti di interesse. Si pensi al caso del monitoraggio ambientale ed in particolare al caso in cui l'APR viene utilizzato per monitorare lo stato di salute delle piantagioni mediante l'acquisizione di immagini multi-spettrali (Sezione 2.2.2.2): in questo caso il drone deve solamente portare la camera digitale utilizzata per l'acquisizione sopra le piantagioni; al fine della missione non è necessario che il modulo di navigazione analizzi tali immagini in tempo reale per modificare il piano di volo.

- **Missioni di ricerca:** in questo caso, il drone è utilizzato per ricercare un'evidenza all'interno di un'area più o meno vasta. Per tali missioni è importante che il modulo di navigazione possa interagire con i sensori utilizzati (*Active Sensing*) per due motivi: il primo è per valutare se l'obiettivo della missione è stato raggiunto e quindi interrompere la ricerca facendo atterrare il velivolo; il secondo è perché in alcuni casi la navigazione è guidata proprio dai valori letti dai sensori. Un esempio di quest'ultimo caso è la ricerca di un disperso sotto le valanghe (Sezione 2.2.2.1) dove per localizzare la vittima viene analizzata la potenza del segnale radio che lo sciatore emette tramite un apposito dispositivo che porta con sé: in questa situazione è opportuno che il modulo di navigazione piloti l'APR in modo da fargli seguire il gradiente del segnale radio fino al raggiungimento della massima intensità così da individuare il punto dove è localizzata la vittima.

- **Missioni di inseguimento:** in questo caso, il modulo di navigazione deve pilotare il velivolo facendo riferimento ai dati provenienti dai sensori utilizzati per il riconoscimento della posizione in cui si trova l'obiettivo da seguire. Per questa tipologia di missioni è necessario utilizzare tecniche di *Active Sensing*, le quali permettono di avere un velivolo in grado di decidere autonomamente il percorso da eseguire poiché non è possibile sapere a priori gli spostamenti che farà l'obiettivo da seguire. Per questo tipo di missioni, infatti, la programmazione tramite *waypoints* risulta inefficace.

Campionamento ed Elaborazione Dati

Lo scopo di questo modulo è quello di fornire al sistema delle regole su come devono essere acquisiti i dati dai sensori installati a bordo dell'aeromobile e su come tali dati devono essere elaborati prima di essere inviati all'utente.

Dato che uno degli obiettivi del nostro lavoro è quello di realizzare una piattaforma che si adatti a vari tipi di missione, le modalità con cui vengono fatte acquisizione ed elaborazione dei dati devono essere personalizzabili dall'utente.

In Figura 3.2 è illustrato il processo di acquisizione ed elaborazione dati svolto dal sistema tramite le regole specificate dal modulo di campionamento ed elaborazione dati.

Per l'acquisizione dei dati, l'utente dovrà specificare una condizione che verrà valutata continuamente durante l'esecuzione della missione; quando questa sarà verificata il sistema dovrà acquisire un campione da un sensore e dovrà memorizzarlo insieme agli altri dati raccolti durante la missione. È importante osservare che, per l'acquisizione, l'utente deve solamente specificare una condizione poiché la parte di acquisizione è gestita in automatico dal sistema. Il sensore utilizzato per il campionamento deve essere specificato dall'utente all'avvio della missione.

Inoltre, per personalizzare il processo di elaborazione dei campioni acquisiti l'utente dovrà specificare le procedure da applicare ai dati raccolti per ottenere i risultati voluti. Esempi di elaborazioni dati possono essere: la creazione di mappe partendo dalle singole immagini oppure la ricerca del valore massimo o minimo tra tutti i campioni acquisiti.

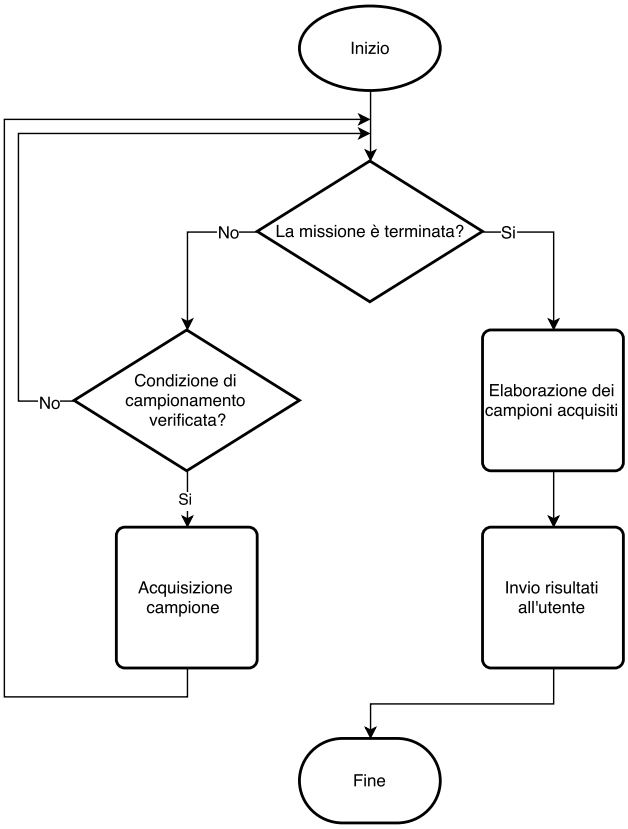


Figura 3.2: Processo di acquisizione ed elaborazione dati

Come nel caso precedente illustriamo come varia il comportamento di questo modulo nelle classi di applicazioni individuate nella Sezione 3.1:

- **Missioni di campionamento:** in questo caso il modulo di campionamento ed elaborazione dati svolge un ruolo fondamentale perché deve informare il sistema al fine di fargli acquisire dei campioni, dal sensore utilizzato per lo svolgimento della missione, quando l'APR raggiunge le posizioni indicate dall'utente. Inoltre, al termine della missione deve, se necessario, elaborare i dati raccolti per fornire all'utente il risultato della missione. Il caso tipico di missioni di campionamento dove è richiesta sia l'acquisizione che l'elaborazione dei dati raccolti è costituito dalle applicazioni che utilizzano il drone per costruire delle mappe del suolo: in queste circostanze, il velivolo deve volare al di sopra dell'area da mappare e scattare una fotografia ogni qualvolta la condizione di campionamento ha esito positivo, ovvero quando l'APR raggiunge determinate coordinate. Una volta che è stata coperta tutta l'area, la procedura di elaborazione, partendo dalle immagini acquisite, dovrà costruire la mappa del suolo e fornirla all'utente.

- **Missioni di ricerca:** anche in queste missioni, il modulo di campionamento ed elaborazione dati svolge un ruolo importante. In questo caso la condizione

da verificare per effettuare il campionamento è la presenza o meno dell'evidenza da ricercare; tale condizione è molto più complessa rispetto al caso delle missioni di campionamento dove per effettuare un'acquisizione bisognava solamente verificare il raggiungimento o meno di una coordinata. Tuttavia, rispetto al caso precedente, nelle missioni di ricerca, di norma non bisogna effettuare nessuna elaborazione sui dati prima di presentarli all'utente perché i campioni acquisiti contengono già le informazioni sulla posizione in cui è stata rilevata la presenza dell'evidenza. Si pensi allo scenario della ricerca di guasti degli oleodotti (Sezione 2.2.2.6) dove l'obiettivo della missione è quello di sorvolare le condotte per valutare la presenza di perdite; in questo caso la procedura di acquisizione dovrà analizzare continuamente le immagini provenienti dalle telecamera di bordo e, applicando delle tecniche per l'analisi delle immagini, dovrà valutare se è presente o meno una perdita: nel caso affermativo dovrà essere acquisito un campione contenente la posizione di dove è stato rilevato il guasto.

- **Missioni di inseguimento:** l'utilizzo di questo modulo è indispensabile quando si vuole valutare lo spostamento che ha fatto l'oggetto mentre era seguito dall'APR: in questa situazione, la procedura per l'acquisizione dati deve avere sempre esito positivo in modo che si acquisiscano continuamente i dati relativi alla posizione dell'oggetto inseguito. Come nel caso precedente non è necessario elaborare i campioni prima di ritornarli all'utente. Nel caso in cui non si desidera monitorare gli spostamenti effettuati dall'oggetto seguito dal drone non è necessario specificare alcuna condizione di campionamento ed alcuna procedura di elaborazione perché per seguire un *target* è necessario l'utilizzo del solo modulo di navigazione.

3.2.2 Interfaccia Sensori

Uno dei limiti del sistema proposto fino a questo momento è quello che l'utente ha accesso ai dati raccolti durante la missione solamente quando quest'ultima termina.

Questo comportamento è accettabile in alcune applicazioni, come la creazione di mappe, dove il modulo di campionamento ed elaborazione dati necessita di avere tutti i campioni prima di poter elaborare i dati da inviare all'utente. Per altre tipologie di missioni, invece, può costituire un enorme svantaggio: si pensi ad esempio al caso di monitoraggio di aree colpite da calamità naturali dove si ha la necessità di visualizzare in tempo reale le immagini provenienti dalla telecamera installata a bordo dell'APR per valutare la presenza di dispersi o per analizzare la situazione di stabilità degli edifici. In situazioni di questo tipo non è possibile pensare di fornire agli operatori le immagini provenienti dal drone solamente a missione terminata, ovvero quando un'intera area è stata perlustrata, ma per facilitare il lavoro dei soccorritori si ha la

necessità di fornire dei metodi per analizzare in tempo reale i dati provenienti dai sensori installati a bordo dell'APR.

Per questo motivo abbiamo deciso che il nostro sistema dovesse mettere a disposizione dell'utente, tramite l'*interfaccia sensori*, la possibilità di visualizzare il valore istantaneo di tutti i sensori installati a bordo, indipendentemente dal fatto che la missione sia in corso di esecuzione o meno.

3.2.3 Tipologie di Missioni

In alcune applicazioni può sorgere la necessità di eseguire in maniera ripetuta la stessa missione. In genere questo comportamento è richiesto per le applicazioni che devono analizzare come variano i dati raccolti in un arco di tempo: un esempio di questo caso può essere l'utilizzo del drone per la raccolta di dati meteorologici prolungata nel tempo; un secondo esempio può essere l'utilizzo del drone in applicazioni di videosorveglianza dove l'APR deve continuare a sorvolare un perimetro e acquisire le immagini provenienti dalla telecamera installata a bordo.

Per semplificare la gestione del volo in queste missioni, il modello per il controllo dell'APR proposto permette di specificare se una missione deve essere eseguita una sola volta oppure se deve essere eseguita in maniera continua finché l'utente non decide di interromperla. Pertanto sono state definite due tipologie di missioni: *standard* e *cicliche*.

Missioni Standard

Sono le missioni più semplici: la missione inizia quando il modulo di navigazione viene mandato in esecuzione e termina dopo che l'ultima istruzione che esso contiene viene eseguita. Durante l'esecuzione della missione il modulo di campionamento si occupa di acquisire i dati dai sensori. L'utente può valutare in tempo reale lo stato dei sensori attraverso la loro interfaccia. Quando una missione termina i dati raccolti vengono elaborati e vengono mostrati all'utente.

Le fasi principali di questo tipo di missioni sono:

1. Esecuzione del modulo di navigazione: l'APR volerà seguendo le istruzioni specificate dall'utente.

2. Acquisizione dati: mentre il piano di volo è in esecuzione, il sistema controlla continuamente se la condizione di campionamento specificata dall'utente è verificata. In caso affermativo viene acquisito e memorizzato un campione dal sensore utilizzato per la missione.

3. Elaborazione campioni: una volta terminata l'esecuzione del piano di volo, i dati raccolti vengono elaborati secondo le necessità dell'utente.

4. Invio dati all'utente: conclusa l'elaborazione i dati ottenuti vengono inviati all'utente.

Le missioni standard sono utilizzate in tutte le applicazioni di campionamento dove non si ha la necessità di monitorare costantemente l'ambiente o il suolo sorvolato, ad esempio nelle applicazioni per la realizzazione di mappe o per il monitoraggio di calamità naturali. Anche le operazioni di ricerca generalmente utilizzano le missioni standard perché una volta individuato l'oggetto ricercato non ha senso rilanciare la missione.

Missioni Cicliche

Queste missioni, a differenza di quelle standard, non terminano dopo il completamento dell'ultima istruzione presente nel modulo di navigazione, ma devono essere interrotte dall'utente.

Le fasi principali rimangono invariate rispetto a quelle delle missioni standard con la differenza che queste vengono ripetute continuamente finché l'utente non decide di terminare la missione.

Come citato precedentemente questa tipologia di missione è utilizzata per le applicazioni di campionamento che richiedono di monitorare costantemente i dati provenienti dai sensori, ad esempio per le applicazioni di videosorveglianza.

Inoltre, questa tipologia di missione può essere utilizzata per realizzare applicazioni di inseguimento riutilizzando il codice delle applicazioni di ricerca. Si pensi ad esempio all'utilizzo degli APR per seguire dei soggetti che praticano attività sportiva: in questo caso, il modulo di navigazione deve implementare la procedura per identificare il soggetto e raggiungerlo (algoritmo di ricerca); utilizzando una missione ciclica per l'esecuzione di questa applicazione, l'APR continuerebbe a ricercare e a raggiungere il soggetto facendo si che il velivolo segua l'obiettivo mobile nei suoi spostamenti. Alla Sezione 3.3.4 è illustrato in maniera più dettagliata questo caso.

3.3 Esempi

In questa sezione verrà illustrato come il modello di programmazione proposto per la gestione degli APR si può adattare ad alcune situazioni reali. Gli esempi scelti per questa analisi appartengono alle tre classi di applicazioni individuate nella Sezione 3.1 così da mostrare come la soluzione proposta per la gestione dell'APR è utilizzabile in diversi contesti. Per ogni scenario illustrato saranno spiegati, tramite l'ausilio di segmenti di pseudocodice, come devono essere strutturati i moduli di navigazione e di campionamento ed elaborazione dati.

3.3.1 Creazione di Mappe

Classe di applicazione: Campionamento

Descrizione: Lo scopo di questa applicazione è quello di utilizzare un APR per sorvolare un'area ai fini di creare una mappa del suolo. Durante il volo il drone dovrà scattare delle fotografie le quali, al termine della missione, verranno unite per creare la mappa.

Dati in input: Area da mappare

Dati in output: Mappa del suolo

Tipologia di missione: Standard

Modulo di navigazione: Come mostrato dall'Algoritmo 3.1, il modulo di navigazione, tramite la funzione *generateWaypoints*, dovrà generare una lista di coordinate geografiche per poter coprire tutto il sito da mappare e successivamente pilotare l'APR facendogli raggiungere tutti i punti generati.

Algoritmo 3.1 Modulo di navigazione per lo scenario: *Creazione di Mappe*

```
1: a ← AREA                                  ▷ Area da sorvolare
2: wpList ← generateWaypoints(a)             ▷ Genera lista di waypoint da raggiungere
3: takeOff()                                 ▷ Decolla
4: for all w ∈ wpList do
5:     fightTo(w)                            ▷ Raggiungi la coordinata specificata
6: end for
7: land()                                    ▷ Atterra
```

Modulo di campionamento: La condizione che il sistema verifica per fare l'acquisizione delle immagini (Algoritmo 3.2) dovrà avere esito positivo ogni qualvolta l'APR si trova in una delle coordinate geografiche generate dalla funzione *generateWaypoints* per coprire l'aera da mappare. La procedura da applicare ai dati raccolti dovrà usare degli strumenti grafici per l'elaborazione delle immagini acquisite al fine di produrre la mappa (Algoritmo 3.3).

Algoritmo 3.2 Condizione di campionamento per lo scenario: *Creazione di Mappe*

```
1: a ← AREA                                  ▷ Area da sorvolare
2: wpList ← generateWaypoints(a)             ▷ Genera lista di waypoint
3: pos ← getCurrentPosition()
4: if pos ∈ wpList then
5:     return True
6: else
7:     return False
8: end if
```

Algoritmo 3.3 Elaborazione dei dati acquisiti per lo scenario: *Creazione di Mappe*

1: $samples \leftarrow ACQUIRED\_DATA$
2: $map \leftarrow generateMap(samples)$
3: **return** map

3.3.2 Ricerca Guasti Oleodotti

Classe di applicazione: Ricerca

Descrizione: La ricerca dei guasti su una rete di oleodotti prevede che il drone sorvoli le condotte ed attraverso l'analisi delle immagini catturate rilevi la presenza o meno di perdite di sostanze inquinanti. Al termine della missione l'APR deve comunicare all'utente le coordinate geografiche dei punti dove sono stati localizzati eventuali guasti.

Dati in input: Percorso da seguire espresso come lista di *waypoints*

Dati in output: Coordinate geografiche di eventuali guasti

Tipologia di missione: Standard

Modulo di navigazione: Data la lista di *waypoints* rappresentante il percorso da seguire, il modulo di navigazione deve pilotare l'APR facendogli raggiungere in sequenza ogni *waypoints* (Algoritmo 3.4). Il modulo di navigazione, per svolgere questo tipo di applicazione è molto simile a quello utilizzato al caso precedente (Algoritmo 3.1): quello che cambia è che in questo caso la lista di *waypoints,* che indica il percorso da seguire per sorvolare le condotte, è fornita direttamente dall'utente mentre nel caso precedente essa doveva essere ricavata dai parametri che specificavano l'area da campionare.

Algoritmo 3.4 Modulo di navigazione per lo scenario: *Ricerca Guasti Oleodotti*

1: $wpList \leftarrow$ `PATH` ▷ Lista di waypoint
2: $takeOff()$ ▷ Decolla
3: **for all** $w \in wpList$ **do**
4: $\quad fightTo(w)$ ▷ Raggiungi la coordinata specificata
5: **end for**
6: $land()$ ▷ Atterra

Modulo di campionamento: In questo caso la condizione che l'utente deve specificare affinché venga registrato un campione contenente le coordinate del guasto deve avere esito positivo ogni qualvolta l'analisi dell'immagine rilevi la presenza di una perdita (Algoritmo 3.5). Per questo scenario non è necessario specificare alcuna procedura di elaborazione dei campioni acquisiti.

Algoritmo 3.5 Condizione di campionamento per lo scenario: *Ricerca Guasti Oleodotti*

```
1: image ← takePicture()                    ▷ Acquisisci un'immagine
2: if oilLeak(image) then
3:     return True
4: else
5:     return False
6: end if
```

3.3.3 Ricerca Sciatori Travolti da Valanghe

Classe di applicazione: Ricerca

Descrizione: La ricerca di dispersi sotto le valanghe è eseguita tramite l'utilizzo di un particolare dispositivo denominato ARTVA il quale riceve un segnale radio da un trasmettitore che la vittima porta con sé. Per identificare la posizione dello sciatore disperso è sufficiente identificare il luogo dove la potenza del segnale radio è massima. Per ottimizzare i tempi di ricerca il drone deve muoversi seguendo il gradiente del segnale spostandosi verso valori crescenti.

Dati in input: Nessuno

Dati in output: Coordinata geografica del punto in cui è stato rilevato lo sciatore

Tipologia di missione: Standard

Modulo di navigazione: Analizzando i dati ricevuti dal ricevitore ARTVA, il modulo di navigazione dovrà far muovere l'APR verso valori crescenti del segnale radio fino al raggiungimento del punto in cui il valore è massimo (Algoritmo 3.6). In questo esempio, il modulo di navigazione utilizza la funzione *getNextPoint* per far muovere il drone seguendo il gradiente del segnale emesso dall'ARTVA. L'utilizzo di questa procedura è necessaria perché il sensore ARTVA impiegato per la ricerca restituisce un valore di intensità e non la direzione dove il segnale è più forte; dato che noi siamo interessati alla direzione di massima crescita del segnale, questa funzione deve far volare il drone in modo da perlustrare una piccola area attorno al punto in cui si trova l'APR e restituire la posizione in cui il segnale è più alto così da ottenere lo stesso output che fornirebbe un sensore direzionale.

Algoritmo 3.6 Modulo di navigazione per lo scenario: *Ricerca Sciatori Travolti da Valanghe*

1: $takeOff()$ ▷ Decolla
2: $w \leftarrow getNextPoint()$
3: **while** $w \neq getCurrentPosition()$ **do**
4: $\quad fightTo(w)$ ▷ Raggiungi la coordinata specificata
5: $\quad w \leftarrow getNextPoint()$
6: **end while**
7: $land()$ ▷ Atterra

Modulo di campionamento: La condizione di campionamento dovrà avere sempre esito positivo così da permettere il campionamento continuo della coppia di valori formata da coordinata geografica e intensità del segnale. La procedura di elaborazione dei dati, Algoritmo 3.7, dovrà individuare tra i campioni acquisiti qual è la coppia con il valore di intensità di segnale più alta.

Algoritmo 3.7 Elaborazione dei dati acquisiti per lo scenario: *Ricerca Sciatori Travolti da Valanghe*

1: $samples \leftarrow ACQUIRED\_DATA$
2: $maxValue \leftarrow 0$
3: $coordinate \leftarrow null$
4: **for all** $s \in samples$ **do**
5: $\quad$ **if** $s[artvaSignal] > maxValue$ **then**
6: $\quad\quad maxValue \leftarrow s[artvaSignal]$
7: $\quad\quad coordinate \leftarrow s[coordinate]$
8: $\quad$ **end if**
9: **end for**
10: **return** $coordinate$

3.3.4 Inseguimento Obiettivi Mobili

Classe di applicazione: Inseguimento

Descrizione: Questo tipo di applicazione ha lo scopo di identificare un obiettivo all'interno di un'area e di seguire tutti i suoi spostamenti.

Dati in input: Obiettivo che si vuole seguire

Dati in output: Posizioni dell'obiettivo inseguito

Tipologia di missione: Ciclica

Modulo di navigazione: Il modulo di navigazione deve localizzare l'oggetto da seguire e pilotare il drone affinché questo voli nel punto in cui l'obiettivo è stato identificato (Algoritmo 3.8). Per fare in modo che l'APR segua continuamente l'obiettivo mobile basta specificare che la missione deve essere ciclica: in questa

maniera una volta raggiunto l'oggetto che si vuole seguire il modulo di navigazione viene mandato nuovamente in esecuzione per pilotare il drone nel punto in cui si è spostato l'obiettivo.

Algoritmo 3.8 Modulo di navigazione per lo scenario: *Inseguimento Obiettivi Mobili*

1: $t \leftarrow TARGET$
2: $w \leftarrow getPositionOf(t)$ ▷ Identifica l'obiettivo da seguire
3: $fightTo(w)$ ▷ Raggiungi la coordinata specificata

Modulo di campionamento: La condizione di campionamento dovrà avere sempre esito positivo così da permettere il campionamento continuo della posizione del velivolo. Per questo scenario non è necessario specificare alcuna procedura di elaborazione dei campioni acquisiti.

Capitolo 4

IDrOS: Architettura

In questo capitolo verrà presentato IDrOS - *Internet Drone Operating System* ovvero la piattaforma software che abbiamo sviluppato a supporto del modello di programmazione illustrato al Capitolo 3 e per rendere possibile l'utilizzo di interfacce basate su protocolli Internet per il controllo degli APR. Grazie ad IDrOS è possibile pilotare i droni tramite Internet senza l'ausilio della stazione di controllo; in questo modo è possibile superare i vincoli di interoperabilità presenti nelle soluzioni attuali garantendo ad ogni sistema informatico la possibilità di interfacciarsi con gli APR mediante Internet.

Inizialmente verranno elencati i requisiti funzionali che IDrOS deve soddisfare.

Successivamente si passerà alla descrizione dell'architettura software proposta per la realizzazione.

Infine verranno illustrate due configurazioni di *deployment* con le quali è possibile utilizzare IDrOS, evidenziando i vantaggi e gli svantaggi derivanti dall'utilizzo dell'una o dell'altra soluzione.

4.1 Requisiti Funzionali

Prima di elencare i requisiti funzionali che IDrOS deve avere, vogliamo focalizzare l'attenzione sui limiti dei sistemi di pilotaggio remoto presenti al giorno d'oggi sul mercato.

In primis, possiamo affermare che molti prototipi di APR utilizzati in ambito civile sono pensati per essere utilizzati localmente: ovvero sono pilotati tramite un radiocomando oppure necessitano la presenza di una stazione di controllo per programmare il percorso di volo e per analizzare i dati raccolti durante la missione. Questo approccio ha degli enormi svantaggi per quanto riguarda l'interoperabilità dell'APR con sistemi informatici più complessi perché richiede la presenza di un operatore umano che coordini l'andamento della missione. Nel Capitolo 2 è stato citato l'esempio d'utilizzo degli APR per la raccolta e l'analisi dei dati meteorologici

(Sezione 2.2.2.5): in questo caso la programmazione del piano di volo veniva fatta attraverso una stazione di controllo con la quale l'operatore umano doveva specificare i *waypoints* per far seguire al drone il percorso desiderato. Inoltre, i dati raccolti durante il volo non venivano inviati alla stazione di controllo al termine della missione, ma doveva essere l'utente a doverli reperire dalla memory card installata sul drone prima di poterli analizzare. In questo caso, dato che per la gestione della missione è necessario l'intervento di un operatore umano, risulta evidente che il drone non è integrabile in sistemi informatici o in sistemi di *Workflow Management*, i quali potrebbero essere utilizzati per automatizzare l'esecuzione della missione.

Se questa applicazione fosse stata realizzata nel contesto IoT, un qualsiasi componente operante nella rete dell'APR avrebbe potuto inviare una richiesta al velivolo specificando i compiti da eseguire (ad esempio: il percorso da seguire e i dati meteorologici da acquisire), successivamente il drone in maniera autonoma sarebbe decollato per acquisire le informazioni richieste dall'utente ed avrebbe inviato una risposta contenente i risultati della missione, dopodiché sarebbe stato disponibile per soddisfare nuove richieste.

Un altro svantaggio che presentano molti prototipi di APR è il fatto che essi sono strettamente dipendenti dall'applicazione di utilizzo. Capita spesso che il software per gestire le missioni è sviluppato tenendo in considerazione solamente un contesto di utilizzo; pertanto, non è possibile utilizzare il medesimo velivolo per svolgere dei compiti diversi da quelli per cui è stato progettato senza modificare la logica di controllo della missione. Ad esempio, alcune applicazioni di utilizzo citate nel Capitolo 2, come il monitoraggio ambientale (Sezione 2.2.2.2) e l'aerofotogrammetria (Sezione 2.2.2.3), presentano caratteristiche simili poiché il drone è utilizzato per sorvolare un'area ai fini di acquisire delle fotografie del suolo; tuttavia, non è possibile utilizzare lo stesso APR per svolgere entrambe le applicazioni perché la logica di controllo delle missioni è pensata per utilizzare uno specifico sensore: nel primo caso una camera multi-spettrale, nel secondo caso una fotocamera digitale.

Fatte queste premesse, i requisiti funzionali che IDrOS deve garantire sono:

- Offrire un'interfaccia che permetta di utilizzare l'APR tramite Internet: il requisito fondamentale è che IDrOS offra la possibilità di controllare il velivolo tramite Internet; in particolare, deve mettere a disposizione varie interfacce per poter utilizzare i protocolli di rete dell'IoT illustrati al Capitolo 2 così da poter integrare il drone in ambienti più complessi come *Business Processes* o sistemi di *Workflow Management*.

- Supportare il modello di programmazione definito al Capitolo 3: un altro aspetto fondamentale che il nostro sistema deve garantire è quello di permettere all'APR di utilizzare tecniche di *Active Sensing* per gestire la navigazione così

da poter modificare dinamicamente il piano di volo a seconda degli obiettivi specificati dalla missione. Così facendo il velivolo risulta intelligente, autonomo ed in grado di adattarsi a più tipologie di missioni.

- Compatibilità con diverse tipologie di sensori: l'utente deve poter installare a bordo del drone diverse tipologie di sensori per l'esecuzione di più categorie di applicazioni senza dover modificare il codice che si occupa della logica applicativa necessaria per lo svolgimento della missione.

4.2 Architettura

Definiamo ora l'architettura software che soddisfa i requisiti illustrati nella sezione precedente ed utilizzata per implementare IDrOS. Come mostrato in Figura 4.1, abbiamo scelto di utilizzare un modello stratificato composto da tre livelli: interfaccia Internet, logica applicativa ed astrazione hardware. Le motivazioni che hanno spinto la scelta di quest'architettura sono dovute alla flessibilità che questa configurazione offre: infatti, una modifica fatta ad un livello rimane localizzata e non impatta sugli altri strati; viene così garantita all'utente la possibilità di estendere facilmente la piattaforma secondo le proprie necessità.

Di seguito descriveremo nei dettagli le funzionalità che devono offrire i tre strati che compongono IDrOS.

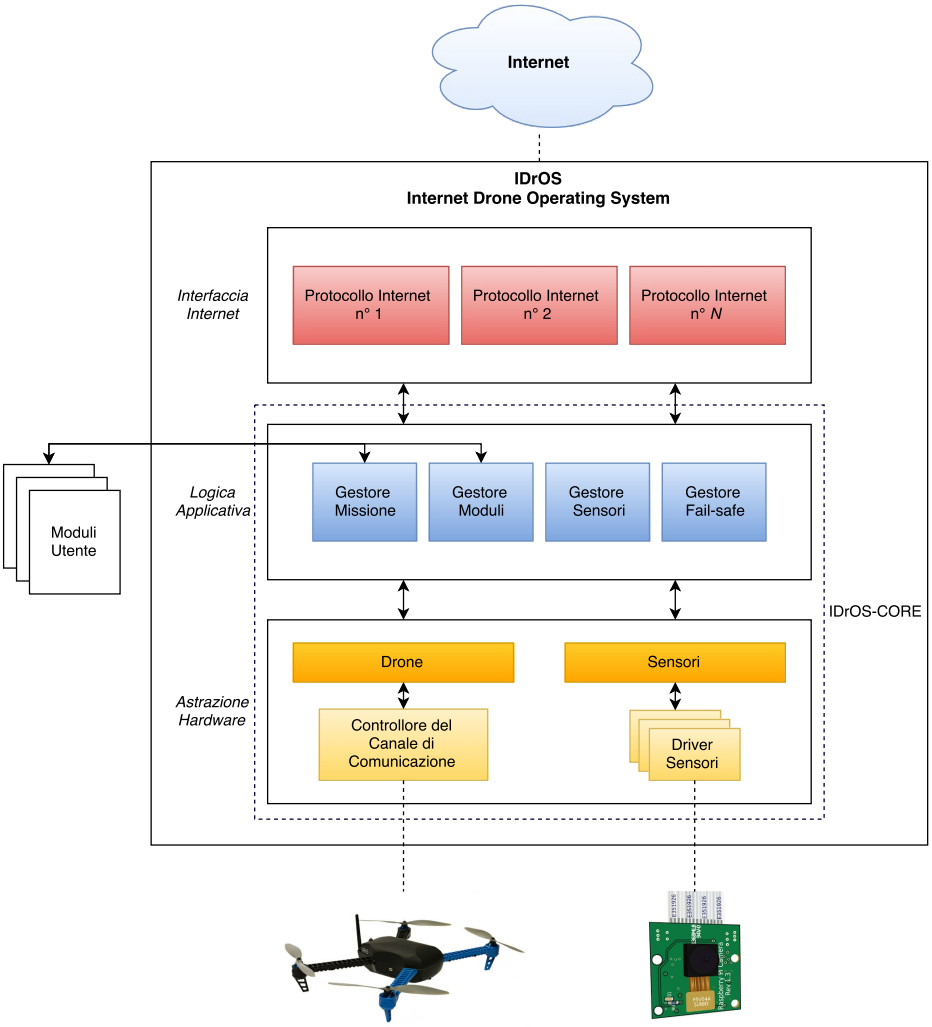


Figura 4.1: Architettura di IDrOS

4.2.1 Astrazione Hardware

Questo livello ha il compito di fornire delle interfacce ad alto livello per comunicare con il drone e con i sensori installati a bordo dell'aeromobile; pertanto, questo strato è composto da due componenti: *Drone* e *Sensori* (Figura 4.1). Questi due componenti devono nascondere i dettagli implementativi riguardanti la gestione dell'hardware al livello superiore, così da semplificare l'accesso da parte dei moduli di navigazione e di campionamento ed elaborazione ai dispositivi utilizzati per pilotare l'APR e per acquisire i dati.

In particolare, il componente *Sensori* deve fornire un'interfaccia che permetta alla logica applicativa di:

- conoscere quali sensori sono installati a bordo;

- fornire delle funzioni ad alto livello per poter leggere la misura registrata dai sensori.

Dato che ogni sensore presenta caratteristiche differenti, al momento dell'installazione, l'utente deve fornire un *driver* per permettere al componente *Sensori* di utilizzare il sensore installato senza conoscere come questo opera e come rappresentare i dati che esso fornisce. In commercio, infatti, esistono svariate tipologie di sensori con caratteristiche totalmente diverse l'uno dall'altro, ciò vale anche per sensori utilizzati per registrare la stessa grandezza fisica; ad esempio, esistono sensori analogici, come il sensore di temperatura LM35 [25], che come output forniscono un valore di tensione elettrica proporzionale alla grandezza rilevata, oppure sensori digitali, come il sensore di temperatura DS18B20 [16], che per essere letti devo essere collegati ad un bus seriale (I2C o il CAN) e forniscono in formato digitale la rappresentazione del valore letto. In questo caso il *driver* utilizzato per gestire il sensore deve preoccuparsi di reperire le informazioni provenienti da esso, interpretare tali valori e fornire all'utilizzatore un risultato contenete la misura eseguita. Ad esempio, nel caso dei due sensori di temperatura illustrati precedentemente, saranno presenti due *driver* per gestire il fatto che un sensore è analogico e l'altro è digitale; entrambi, dovranno fornire all'utente il valore di temperatura letto mascherando tutte le operazioni fatte per acquisire ed interpretare i valori provenienti dal relativo sensore.

Per quanto riguarda il componente *Drone*, esso dovrà fornire alla logica applicativa delle API per pilotare il drone ad alto livello mascherando i dettagli riguardanti il protocollo di comunicazione utilizzato per interfacciarsi con il controllore di stabilità che pilota il velivolo.

Generalmente i comandi utilizzati per pilotare gli APR riguardano la potenza da erogare ai motori per far sì che il velivolo si sposti o mantenga l'assetto desiderato. Questo livello di astrazione è però troppo basso per lo sviluppatore del modulo di navigazione perché richiede di inviare al drone numerosi comandi per effettuare dei compiti anche semplici. Ad esempio, per far decollare il velivolo bisognerebbe inviare il comando di accensione dei motori, successivamente bisognerebbe aumentare la potenza dei motori finché il velivolo si solleva da terra ed una volta avvenuto il decollo bisognerebbe impostare la potenza dei motori ad un valore tale per cui il velivolo rimanga ad una quota costante. Le API che deve fornire il componente *Drone* devono essere ad alto livello così da nascondere alla logica applicativa, incaricata di gestire la missione, tutti questi aspetti legati alla gestione del velivolo. Ad esempio, per il decollo sarà fornito un solo metodo incaricato di far decollare il velivolo e di portarlo alla quota indicata.

Il componente *Drone*, per poter funzionare ed interfacciarsi con il velivolo, deve utilizzare un sotto-componente per la gestione del canale di comunicazione tra le API ed il controllore di stabilità: questo componente dovrà, ad esempio, inizializzare la

comunicazione con il velivolo, inviare i comandi a basso livello per la gestione del aeromobile e ricevere le risposte per valutare se i comandi sono stati eseguiti o meno.

4.2.2 Logica Applicativa

Questo livello ha il compito di gestire la logica applicativa per svolgere le missioni; in particolare, deve fornire svariate funzionalità per fare in modo che il modello di programmazione, proposto alla Sezione 3.2 del capitolo precedente, possa essere implementato su un sistema reale. Per fare ciò bisogna tenere in considerazione alcuni aspetti tecnici che non sono emersi durante la presentazione del modello di programmazione, come ad esempio la gestione dell'APR quando si verificano situazioni di pericolo o malfunzionamenti che potrebbero compromettere il volo.

Pertanto, la logica applicativa è divisa in quattro componenti software, ognuno dei quali è incaricato di offrire delle funzionalità all'utente per poter utilizzare concretamente il modello proposto al Capitolo 3 e per comunicare con l'APR.

Gestore dei Moduli

Gli elementi principali che caratterizzano il modello per la gestione dell'APR illustrato al Capitolo 3 sono: il *modulo di navigazione* ed il *modulo di campionamento ed elaborazione dati*; essi sono dei file contenenti codice eseguibile che permettono rispettivamente di pilotare l'aeromobile e di gestire l'acquisizione e l'elaborazione dei dati provenienti dai sensori. L'utente deve poter caricare su IDrOS i propri moduli così da poter eseguire svariate tipologie di missioni utilizzando lo stesso APR.

Il gestore delle missioni dovrà fornire all'utente la possibilità di:

- Caricare nuovi moduli nel sistema.
- Eliminare dal sistema i moduli che non intende più utilizzare.
- Visualizzare l'elenco dei moduli presenti nel sistema.
- Visualizzare i dettagli riguardanti il comportamento di un modulo.

Gestore della Missione

Il gestore della missione è il componente software in cui risiede gran parte della logica applicativa per la gestione del volo, l'acquisizione dei dati dai sensori e l'elaborazione dei dati raccolti; in particolare, questo gestore si occupa di implementare il modello proposto nella Sezione 3.2. Esso deve: mandare in esecuzione il modulo di navigazione fornito dall'utente per permettere al drone di volare, valutare ciclicamente la condizione - contenuta nel modulo di campionamento ed elaborazione - per l'acquisizione dei dati dai sensori ed eseguire la procedura di elaborazione fornita

dall'utente per processare i dati raccolti prima di inviare la risposta contenente i risultati della missione.

Il gestore della missione, nel caso in cui l'utente specifica di voler eseguire una missione ciclica, è il componente che deve mandare continuamente in esecuzione i moduli di navigazione e di campionamento ed elaborazione.

Questo gestore deve permettere al modulo di navigazione di interfacciarsi con il livello di astrazione hardware per poter inviare i comandi necessari a pilotare l'APR e per accedere ai sensori installati a bordo per l'esecuzione della missione. Anche il modulo di campionamento ed elaborazione dati necessita di interfacciarsi con il livello di astrazione hardware ma, in questo caso, solamente per leggere i dati provenienti dai sensori.

Gestore dei Sensori

Nel modello illustrato al Capitolo 3 è presente anche un componente detto *interfaccia sensori* che ha il compito di offrire un'interfaccia per visualizzare in tempo reale lo stato dei sensori (Sezione 3.2.2).

Il gestore dei sensori soddisfa questi requisiti; in particolare, il suo compito è quello di fornire all'utente dei metodi per permettergli di conoscere quali sono i sensori installati a bordo del sistema e di ottenere in tempo reale la rappresentazione del dato letto dal sensore richiesto.

Questo gestore utilizza il livello di astrazione hardware ed in particolare il componente *Sensori* (Figura 4.1) per poter comunicare con i vari sensori installati a bordo del drone.

Inoltre, per arricchire le funzionalità messe a disposizione dell'utente si è scelto di fornire due modi diversi per accedere alle informazioni fornite dai sensori:

1. Tramite una richiesta per visualizzare il valore attuale di un sensore: in questo caso l'utente dovrà effettuare una richiesta specificando quale sensore intende leggere, dopodiché il gestore dei sensori dovrà reperire tale informazione dal livello di astrazione hardware e rispondere all'utente con il dato richiesto.

2. Sottoscrivendosi alla ricezione dei dati da parte di un sensore: dopo avere ricevuto una richiesta di sottoscrizione da parte dell'utente, il gestore dei sensori dovrà comunicare al livello di astrazione hardware l'interesse a ricevere i dati di un sensore ogni qualvolta questo cambi stato. Ad ogni cambiamento di stato il gestore dei sensori dovrà notificare all'utente il nuovo valore. Questo procedimento deve andare avanti finché l'utente non cancella la sottoscrizione.

La seconda tipologia di utilizzo è molto utile quando si vuole visualizzare costantemente ed in tempo reale lo stato di un sensore, ad esempio si potrebbe utilizzare per fornire all'utente un flusso di immagini per mostrargli quello che vede il drone mentre è in volo.

Gestore Fail-safe

La presenza di questo componente all'interno dell'architettura di IDrOS è giustificata dal fatto che durante l'esecuzione di una missione si possono verificare delle situazioni, come la presenza di vento in quota oppure il malfunzionamento dell'elettronica di bordo, che potrebbero compromettere la stabilità del volo e far precipitare l'aeromobile. In queste situazioni l'utente può fare una richiesta a questo gestore per far entrare il drone in modalità *fail-safe*: così facendo, l'esecuzione della missione viene interrotta e il velivolo viene fatto atterrare immediatamente.

Si può quindi intuire che questo componente non serve per implementare parte della logica di esecuzione della missione, ma ha il compito di offrire all'utente delle funzionalità accessorie da utilizzare nei momenti di criticità per evitare di causare danni al velivolo.

Questo componete dovrà interfacciarsi con il livello di astrazione hardware per inviare al velivolo i comandi per farlo atterrare.

4.2.3 Interfaccia Internet

L'interfaccia Internet è lo strato software che permette ad IDrOS di offrire all'utente ed ai vari dispositivi operanti nel contesto dell'IoT la possibilità di comunicare con l'APR. Dato che non esiste una soluzione generale per gestire la comunicazione tra due o più dispositivi, in quanto oggi giorno è possibile utilizzare molti protocolli di comunicazione spesso operanti con diversi *pattern*, ci sembrava limitante offrire un solo modo per interconnettere il drone ad Internet. Perciò, abbiamo deciso che questo livello software doveva garantire la possibilità all'utente di utilizzare vari protocolli Internet per dialogare con l'aeromobile; per questo motivo, lo strato interfaccia Internet è composto da vari componenti software che rappresentano i binding necessari all'utente per utilizzare le funzionalità messe a disposizione dal livello della logica applicativa attraverso l'utilizzo di diversi protocolli di rete.

La scelta di offrire vari protocolli di comunicazione per dialogare con altri dispositivi ha il vantaggio di garantire un alto livello di interoperabilità; in tal modo IDrOS risulta molto flessibile ed utilizzabile in svariati contesti. Ad esempio, come sarà mostrato nel Capitolo 5, la nostra implementazione offrirà la possibilità di utilizzare sia il protocollo CoAP che il protocollo MQTT per comunicare con l'APR.

Inoltre, l'utente - senza modificare la logica di gestione della missione - può decidere di implementare un nuovo binding per permettere all'APR di comunicare utilizzando altri protocolli come HTTP.

4.3 Configurazioni di Deployment

Proponiamo ora due configurazioni di *deployment* con le quali è possibile utilizzare IDrOS.

La prima tipologia di configurazione è quella che abilita l'utilizzo degli APR nel contesto dell'IoT; come mostrato in Figura 4.2, essa prevede che IDrOS sia installato a bordo dell'APR.

Grazie a questa soluzione non è necessario utilizzare la stazione di controllo per pilotare il velivolo; IDrOS infatti, come detto precedentemente, deve mettere a disposizione delle interfacce che permettono di controllare il velivolo mediante Internet.

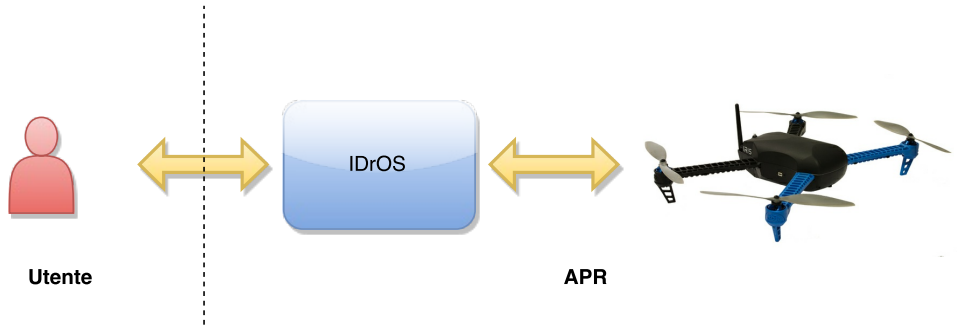


Figura 4.2: Configurazione di *deployment* con IDrOS installato a bordo dell'aeromobile

È importante osservare che in questa situazione lo scambio di messaggi tra l'utente e IDrOS avviene ad un livello di astrazione alto: l'utente o il dispositivo che vuole utilizzare il drone deve solamente effettuare una richiesta ad IDrOS per richiedere che venga svolta una determinata azione; il sistema dopo avere elaborato la richiesta deve eseguirla e ritornare all'utente una risposta contenente i risultati dell'operazione.

Le fasi principali dell'esecuzione di una missione ed i messaggi scambiati tra la stazione di controllo e l'APR sono illustrati in Figura 4.3

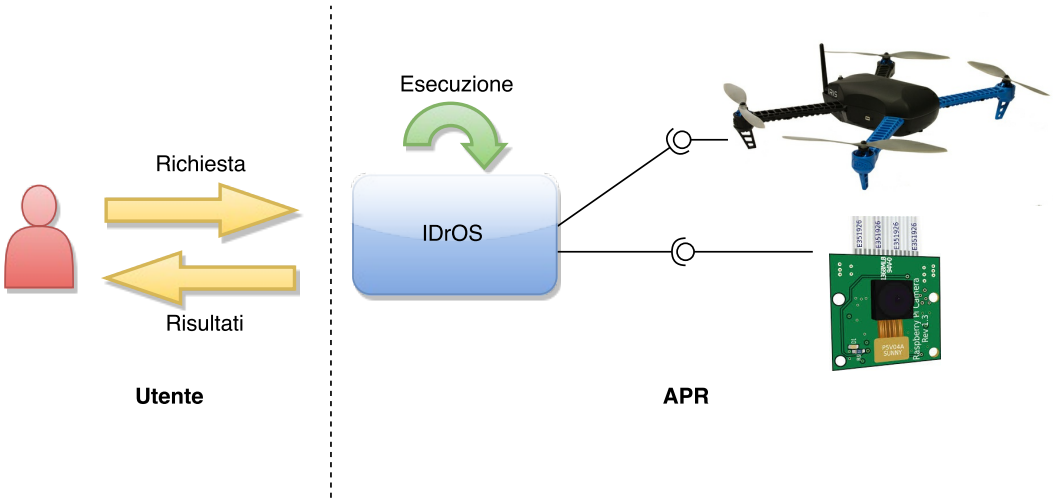


Figura 4.3: Fasi dell'esecuzione di una missione nella configurazione di *deployment* con IDrOS installato a bordo dell'aeromobile

I vantaggi derivanti dall'utilizzo di questa soluzione sono i seguenti:

1. Non è necessario utilizzare una stazione di controllo per pilotare l'APR; grazie alle interfacce Internet ogni dispositivo dotato di una scheda di rete può connettersi con il velivolo così da poterlo controllare.

2. Il canale di comunicazione per interagire con IDrOS può avere latenze alte; infatti, una volta ricevuta una richiesta, IDrOS gestisce tutte le fasi della missione. Quindi non è un problema se i messaggi scambiati con l'utente impiegano del tempo per giungere a destinazione.

3. L'APR non necessita di rimanere in contatto con i dispositivi di controllo durante l'esecuzione della missione; ciò permette all'APR di potersi spostare al di fuori dell'area di copertura radio rendendo così possibile l'esecuzione di missioni che richiedono al velivolo di compiere grandi spostamenti.

Gli svantaggi dovuti all'utilizzo di questa soluzione sono essenzialmente dovuti alle capacità di calcolo degli APR: in questa configurazione infatti il velivolo deve avere abbastanza capacità di elaborazione per poter eseguire IDrOS. Tuttavia, come vedremo nel Capitolo 5, il nostro sistema è pensato per poter operare su dispositivi dotati di limitate capacità di calcolo, quindi quest'ultimo aspetto può essere trascurato.

La seconda tipologia di configurazione che proponiamo è speculare rispetto al caso precedente ed è quella più classica in quanto è utilizzata in molti sistemi già esistenti; essa consiste nel dislocare gran parte della logica applicativa nella stazione di controllo e nell'avere un APR con ridotte capacità di elaborazione, in grado solo di eseguire semplici comandi inviati dalla stazione di terra. In Figura 4.4 è mostrato un esempio di questa configurazione.

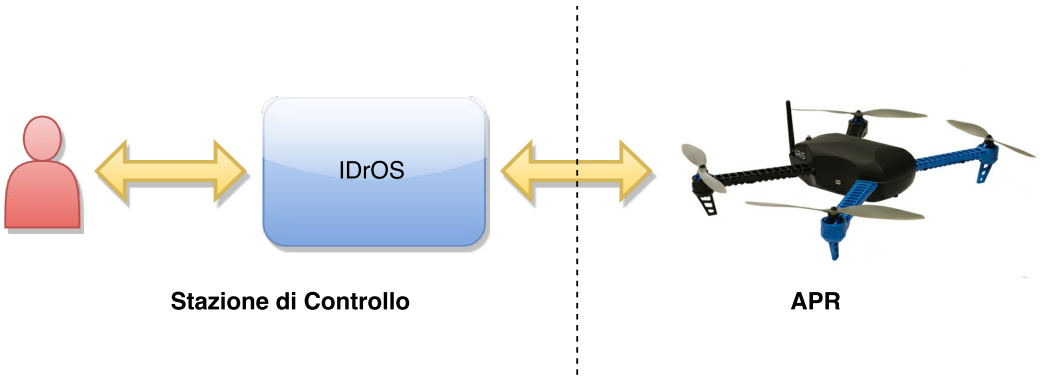


Figura 4.4: Configurazione di *deployment* con IDrOS installato nella stazione di controllo

In questa situazione il controllo della missione è gestito a terra ed il velivolo deve solamente eseguire i comandi per la navigazione che riceve dalla stazione di controllo e comunicare i parametri di volo e i dati letti dai sensori utilizzati per lo svolgimento della missione.

In Figura 4.5 sono rappresentate le fasi principali per l'esecuzione di una missione ed i messaggi scambiati con l'aeromobile.

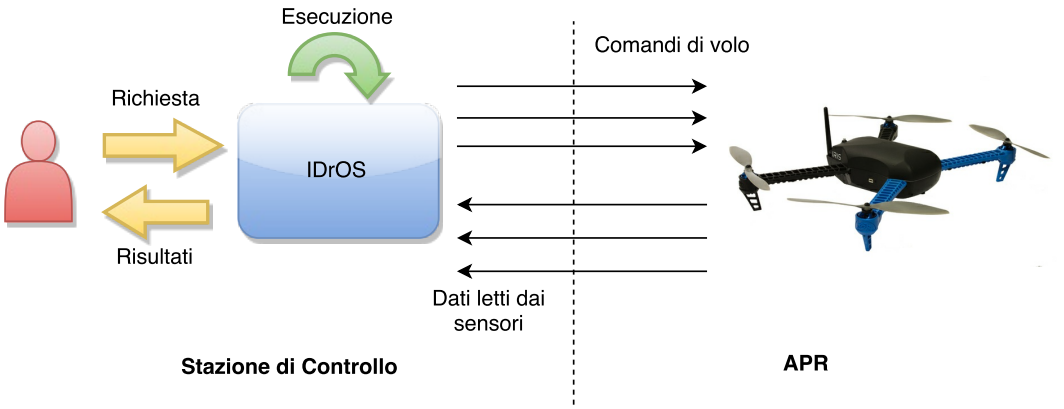


Figura 4.5: Fasi dell'esecuzione di una missione nella configurazione di *deployment* con IDrOS installato nella stazione di controllo

Il vantaggio che si trae dall'utilizzo di questo approccio risiede nel fatto che l'elettronica di controllo a bordo dell'APR non deve essere molto performante in quanto la gestione della missione e l'elaborazione dei dati provenienti dai sensori vengono eseguite nei calcolatori che compongono la stazione di controllo.

Tuttavia, in questa configurazione si hanno i seguenti svantaggi:

1. Durante l'esecuzione della missione non si può perdere il collegamento radio tra la stazione di controllo e l'APR; in queste circostanze non sarebbe più

possibile inviare i comandi di volo necessari per l'esecuzione della missione e si perderebbe il controllo del velivolo.

2. Il canale di comunicazione deve avere poca latenza: la stazione di controllo, per riuscire a coordinare la missione, deve sapere in tempo reale quali sono le informazioni riguardanti l'assetto di volo del velivolo.

3. Il canale di comunicazione radio deve gestire un gran numero di messaggi, oltre ai comandi di volo esso deve trasmettere tutti i dati acquisiti dai sensori necessari per l'avanzamento della missione.

4. L'utente avrà poca mobilità sul campo perché la stazione di controllo, dovendo gestire l'esecuzione della missione e garantire la presenza del canale di comunicazione, sarà sofisticata e pertanto renderà complicati gli spostamenti dell'utente.

Tra queste due configurazioni di *deployment*, quella più interessante è la prima - ovvero quella che prevede di installare IDrOS a bordo del velivolo - perché presenta meno vincoli per il controllo e la gestione dell'APR. Inoltre, data l'assenza della stazione di controllo essa è anche quella più flessibile poiché permette a qualunque dispositivo di connettersi al velivolo.

Capitolo 5

Implementazione

In questo capitolo focalizzeremo l'attenzione sui dettagli, le scelte effettuate e le problematiche riscontrate durante lo sviluppo di IDrOS.

Inizialmente verrà presentato il software di volo e la relativa scheda di supporto che abbiamo utilizzato come controllore di volo per lo sviluppo del nostro sistema.

Successivamente verranno messe in luce le scelte implementative per realizzare l'architettura proposta al Capitolo 4: in particolare verranno ripresi i livelli software che la costituiscono e verranno illustrati i dettagli che caratterizzano ciascuno di essi.

Infine verrà presentato il velivolo che abbiamo costruito elencando i componenti necessari per la sua realizzazione e per far sì che esso possa eseguire IDrOS.

Lo sviluppo di IDrOS e la realizzazione del velivolo ci hanno permesso di ottenere un prototipo hardware/software funzionante su cui è stato possibile mettere in pratica il modello astratto per la programmazione degli APR (Capitolo 3) ed eseguire alcuni test al fine di valutare le performance del sistema (Capitolo 6).

5.1 ArduCopter

Prima di focalizzare l'attenzione sui dettagli riguardanti l'implementazione di IDrOS è doveroso illustrare il controllore di volo necessario per far sì che il velivolo, sul quale vogliamo sviluppare il nostro sistema, sia in grado di volare. Il controllo di stabilità del velivolo infatti è un task complementare, svolto da una board dedicata, che IDrOS non deve preoccuparsi di gestire.

Per la realizzazione del nostro lavoro abbiamo deciso di utilizzare come APR di riferimento un quadricottero, ovvero un velivolo multirotore che viene sollevato e mosso grazie all'ausilio di quattro motori. Per coordinare i motori che compongono il velivolo, abbiamo deciso di utilizzare come controllore di stabilità ArduCopter [5]: la scelta di questo firmware è stata dettata dal fatto che esso è un software open source, con una ricca community di sviluppo e che si adatta facilmente ad

un'ampia gamma di velivoli multirotori; infatti, è possibile utilizzare ArduCopter per controllare tricotteri, quadricotteri, esacotteri ed ottocotteri.

ArduCopter, oltre a coordinare i motori che compongono il velivolo affinché questo sia in grado di volare, permette l'utilizzo di svariati sensori, tra cui GPS, magnetometro, barometro e sonar, per fare in modo che il velivolo mantenga stabile la posizione e l'orientamento durante il volo.

Per pilotare il velivolo ArduCopter offre all'utente la possibilità di selezionare una modalità di volo, ogni modalità presenta delle caratteristiche differenti su come il controllore di stabilità interpreta i comandi ricevuti per regolare l'assetto del velivolo. Di seguito sono illustrate alcune modalità di volo disponibili per controllare il velivolo:

- **Stabilize:** questa modalità permette di pilotare il velivolo manualmente, ArduCopter si preoccupa solamente di stabilizzare il velivolo permettendo al pilota di specificare i parametri di beccheggio, rollio ed imbardata.

- **AltHold:** se è specificata questa modalità di volo, il controllore di volo oltre a stabilizzare il velivolo permette di mantenere un'altezza fissa durante il volo.

- **Loiter:** quando il velivolo è in questa modalità di volo, se il pilota non invia comandi al drone, ArduCopter si preoccupa di mantenere stabile la posizione, l'orientamento e l'altitudine dal suolo. Per utilizzare questa modalità è necessario che il velivolo sia dotato di GPS, magnetometro, barometro ed eventualmente di un sonar.

- **Land:** quando viene impostata questa modalità di volo il velivolo atterra.

- **RLT:** la modalità RTL (*Return To Launch*) quando utilizzata guida il drone fino alla *Home Position* e lo fa atterrare. Generalmente la *Home Position* è la posizione in cui il velivolo è decollato; essa può anche essere impostata dall'utente al momento della configurazione di ArduCopter. Per utilizzare questa modalità è necessario che sul drone sia installato il sensore GPS.

- **Guided:** grazie a questa modalità è possibile indicare al drone una coordinata GPS da raggiungere e l'altitudine da mantenere. Una volta raggiunto il punto stabilito, ArduCopter mantiene stabile la posizione, l'orientamento e l'altitudine dal suolo finché non viene indicato un altro punto da raggiungere. Come si può intuire, questa modalità di volo necessita di poter utilizzare il sensore GPS.

Nell'elenco non sono presenti tutte le modalità di volo che ArduCopter mette a disposizione, quelle elencate sono quelle usate durante il lavoro di tesi. Le modalità che non sono state elencate non sono necessarie ad IDrOS per pilotare il velivolo.

Per poter essere utilizzato correttamente, ArduCopter necessita di una piattaforma hardware su cui essere eseguito; nel nostro caso la scheda che abbiamo scelto è la board Pixhawk [34] (Figura 5.1): essa è quella consigliata dalla comunità che sviluppa il software di stabilità. Le caratteristiche principali di questa scheda sono:

- Processore ARM Cortex M4 a 32 bit.
- 168 MHz/256 KB RAM.
- 2 MB Flash.
- 14 uscite PWM.
- Giroscopio a 3 assi integrato.
- Accelerometro/Magnetometro a 3 assi integrato.
- Barometro integrato.
- Porte: SPI, I2C, CAN e UART per interagire con sensori e/o altri sistemi.

Figura 5.1: Scheda Pixhawk

5.2 Implementazione di IDrOS

In questa sezione verranno illustrate le scelte implementative ed i vari problemi riscontrati durante lo sviluppo di IDrOS.

Dopo aver esaminato il problema della comunicazione tra IDrOS ed ArduCopter, verrà ripresa l'architettura proposta nel Capitolo 4 descrivendo nel dettaglio come sono stati implementati i vari livelli software che compongono IDrOS.

Come discusso nella Sezione 4.3 è possibile utilizzare IDrOS in due configurazioni di *deployment* differenti: la prima prevede che IDrOS sia installato a bordo del

velivolo permettendo all'utente di collegarsi all'APR senza l'ausilio della stazione di controllo; la seconda soluzione prevede che esso sia installato nei dispositivi dislocati a terra ed operi come una stazione di controllo.

Tra le due soluzioni, abbiamo deciso di concentrarci su quella che prevede che IDrOS sia installato a bordo del velivolo. La ragione che ha motivato questa scelta risiede essenzialmente nel fatto che in questo modo si rispetta la filosofia dell'IoT: in questo caso, il drone è un nodo attivo della rete che per operare non ha bisogno di ulteriori componenti, come la stazione di controllo, ma fornisce delle interfacce Internet che espongono le sue funzionalità a tutti i nodi che operano nella stessa rete.

Questa scelta, tuttavia, prevede che a bordo dell'APR sia installata una piattaforma hardware per gestire l'esecuzione di IDrOS e che essa possa comunicare con la scheda Pixhawk per permettere l'interfacciamento tra IDrOS ed ArduCopter.

I problemi legati all'interfacciamento tra i due sistemi (IDrOS e ArduCopter) sono descritti alla Sezione 5.2.1; mentre alla Sezione 5.3.1 verrà presentata la scheda di supporto che abbiamo utilizzato per eseguire IDrOS.

5.2.1 Interfacciamento con ArduCopter

Il primo problema che abbiamo dovuto affrontare durante lo sviluppo di IDrOS è stato quello di trovare un modo semplice ed efficiente per poter comunicare con ArduCopter ai fini di pilotare il velivolo.

ArduCopter presuppone che l'interfacciamento con l'utente o con la stazione di controllo possa essere effettuato in uno dei seguenti modi:

1. Tramite un radiocomando: questa è la tecnica più semplice per avere il controllo del velivolo; essa prevede che i comandi riguardanti gli angoli di rollio, beccheggio e imbardata e la potenza da erogare ai motori vengano inviati dall'utente tramite un radiocomando. A seconda dei valori inviati dall'utente, ArduCopter modifica l'assetto del velivolo portandolo nella configurazione desiderata.

2. Tramite la telemetria: la telemetria è un collegamento radio tramite il quale il velivolo comunica alla stazione di controllo i dati provenienti dalla sensoristica di bordo, come ad esempio i valori letti dagli accelerometri, dal giroscopio, dal barometro e dal GPS. Generalmente la telemetria è utilizzata per il monitoraggio del velivolo e non per il controllo; tuttavia, ArduCopter prevede che il collegamento utilizzato per la telemetria sia bidirezionale offrendo così all'utente la possibilità di inviare comandi per modificare l'assetto di volo. Così facendo è possibile pilotare il velivolo senza l'ausilio del radiocomando. Questa soluzione è quella usata dalle stazioni di controllo sviluppate per comunicare con ArduCopter.

Per utilizzare la prima soluzione bisognerebbe fare in modo che IDrOS emuli la presenza del radiocomando e che invii i comandi relativi agli angoli di rollio, beccheggio e imbardata e la potenza da erogare ai motori ad ArduCopter. La seconda soluzione richiede invece di utilizzare una delle due porte seriali predisposte sulla scheda Pixhawk e di comunicare con l'APR seguendo le direttive specificate dal protocollo MAVLink.

Data la maggiore facilità di impiego, abbiamo deciso di utilizzare il canale dedicato alla telemetria per interfacciare IDrOS con ArduCopter; in particolare, abbiamo collegato tramite un cavo seriale, appositamente costruito, le due schede: quella di supporto per eseguire IDrOS e la board Pixhawk.

Ora illustreremo brevemente il protocollo MAVLink, facendo particolare attenzione alle librerie presenti online per poterlo utilizzare agevolmente.

MAVLink - *Micro Air Vehicle Link* [26] è un protocollo di comunicazione utilizzato in molti sistemi di pilotaggio remoto per permettere lo scambio di messaggi tra la stazione di controllo ed il velivolo.

Per utilizzare questo protocollo esistono svariate librerie e framework che mascherano al programmatore i dettagli tecnici necessari al funzionamento del protocollo fornendo delle API di alto livello per l'invio e la ricezione dei messaggi.

Gli strumenti che abbiamo preso in considerazione per l'utilizzo di questo protocollo al fine di comunicare con il velivolo sono i seguenti:

pymavlink: Questa libreria [35] è l'implementazione di riferimento per il protocollo MAVLink. Essa è scritta in Python e fornisce svariati metodi per inviare comandi all'APR e ricevere i dati della telemetria. Nonostante semplifichi di molto l'utilizzo del protocollo MAVLink, essa non fornisce un elevato grado di astrazione perché permette solo la gestione dei messaggi senza offrire dei metodi ad alto livello per pilotare l'aeromobile. Ad esempio, non è presente un metodo che permette al velivolo di decollare: per fare ciò bisogna inviare dei messaggi per variare la potenza erogata ai motori e monitorare i dati relativi all'altezza dal suolo per capire quando il decollo è avvenuto. Pertanto, per utilizzare questa libreria bisogna avere una buona conoscenza del protocollo MAVLink e dei messaggi che è possibile inviare ad ArduCopter.

MAVProxy: Questo strumento [43] è un'applicazione che si comporta da proxy per il protocollo MAVLink, esso si interpone tra il velivolo e l'utente facendo da tramite fra i due. Per poter pilotare l'APR, l'utente deve inviare i comandi di volo al proxy invece che al velivolo, il proxy a sua volta si collega con l'aeromobile e inoltra la richiesta del client utilizzando il protocollo MAVLink. MAVProxy maschera così tutti i dettagli riguardanti il protocollo di comunicazione con il velivolo fornendo all'utente un'interfaccia a linea di comando per

pilotare l'APR. MAVProxy può essere visto come una stazione di controllo a linea di comando per interagire con gli APR.

DroneKit: Questa libreria [15] è scritta in Python e fornisce all'utente un'interfaccia di programmazione ad oggetti per pilotare l'APR. Il pregio di questa libreria è che nasconde l'esistenza del protocollo MAVLink e fornisce all'utente dei metodi ad alto livello per interfacciarsi con l'aeromobile. Tuttavia, DroneKit necessita di interfacciarsi con MAVProxy per poter controllare il velivolo; perciò, l'utilizzo di questa libreria implica l'utilizzo di MAVProxy.

A prima vista il modo più semplice per interfacciarsi con ArduCopter è quello di usare la libreria DroneKit: così facendo il controllo dell'APR risulterebbe molto semplice perché tutti i dettagli relativi al protocollo MAVLink vengono mascherati da questa libreria.

Nonostante questo vantaggio abbiamo deciso di non utilizzare DroneKit per lo sviluppo di IDrOS perché vogliamo che il nostro sistema, dovendo essere in grado di funzionare su una piattaforma embedded installata a bordo del velivolo, sia minimale e non consumi troppe risorse. L'utilizzo delle API DroneKit, come detto precedentemente, necessita l'utilizzo di MAVProxy il quale, offrendo le stesse funzionalità di una stazione di controllo, richiede molte risorse hardware per poter essere eseguito.

Perciò, la soluzione che abbiamo deciso di adottare per interfacciare IDrOS ad ArduCopter è stata quella di utilizzare la libreria pymavlink e di creare da zero delle API, pensate appositamente per IDrOS, per gestire la comunicazione con il velivolo; tale scelta richiede uno sforzo implementativo maggiore per la realizzazione del livello di astrazione hardware che compone IDrOS ma, come vedremo dagli esperimenti di utilizzo effettuati al Capitolo 6, garantisce un minor consumo di risorse hardware rispetto a MAVProxy.

5.2.2 Astrazione Hardware

Analizziamo ora i dettagli implementativi riguardati lo sviluppo del livello più basso che costituisce IDrOS. Come anticipato nel Capitolo 4, il livello di astrazione hardware deve fornire al livello superiore delle interfacce per pilotare il velivolo e per leggere i sensori installati a bordo dell'APR. Di seguito è presentato nel dettaglio come sono costituite queste due interfacce.

5.2.2.1 Gestione del Velivolo

Per gestire la comunicazione tra ArduCopter e IDrOS, come citato nella Sezione 5.2.1, abbiamo deciso di utilizzare la libreria pymavlink [35] e di realizzare da zero delle API per permettere di pilotare agevolmente il velivolo tramite dei metodi ad alto livello così da nascondere agli strati superiori i dettagli riguardanti la comunicazione con il protocollo MAVLink.

La soluzione da noi utilizzata per implementare queste API è composta da due componenti software: *MavLayer* e *Drone*.

MavLayer: Questo componente è il *controllore del canale di comunicazione* descritto nel capitolo dedicato all'architettura (Figura 4.1). Esso svolge un ruolo fondamentale per la comunicazione con il velivolo perché è il responsabile dell'invio e della ricezione dei messaggi verso e dall'APR utilizzando le specifiche del protocollo MAVLink.

In particolare i compiti che questo componente deve svolgere sono:

1. Inizializzare la connessione con l'APR.
2. Inviare periodicamente il messaggio HEARTBEAT: tale messaggio è necessario ad ArduCopter per autorizzare il drone a volare; in assenza di questo messaggio, infatti, ArduCopter crede di aver perso la comunicazione con la stazione di controllo e non permette il volo.
3. Leggere i messaggi della telemetria ed aggiornare lo stato del velivolo. MavLayer memorizza infatti tutte le variabili riguardanti l'assetto di volo e i dati provenienti dai sensori di bordo così da permettere al componente Drone, illustrato di seguito, di conoscere lo stato dell'APR controllato.
4. Verificare periodicamente lo stato del canale di comunicazione.
5. Inviare i comandi di volo.

In Figura 5.2 è rappresentato il flusso di informazioni che vengono scambiate tra MavLayer e ArduCopter dopo l'inizializzazione della comunicazione. Il primo messaggio che viene inviato è il messaggio di HEARTBEAT; la risposta che invia ArduCopter dopo la ricezione di questo messaggio è un pacchetto dello stesso tipo che contiene le informazioni sulla tipologia del velivolo utilizzato - ad esempio, se un quadricottero o un esacottero. Una volta avvenuto questo scambio di messaggi MavLayer deve richiedere ad ArduCopter di ricevere i dati relativi alla telemetria: per fare questo, deve invocare la funzione *set_stream_rate* indicando la frequenza con cui desidera ricevere questi dati; il rate massimo con cui è possibile ricevere informazioni dalla telemetria è 5Hz. Conclusa questa prima fase, MavLayer dovrà inviare periodicamente il messaggio di HEARTBEAT, valutare lo stato del canale di comunicazione e processare i messaggi relativi alla telemetria per aggiornare le informazioni sullo stato del velivolo.

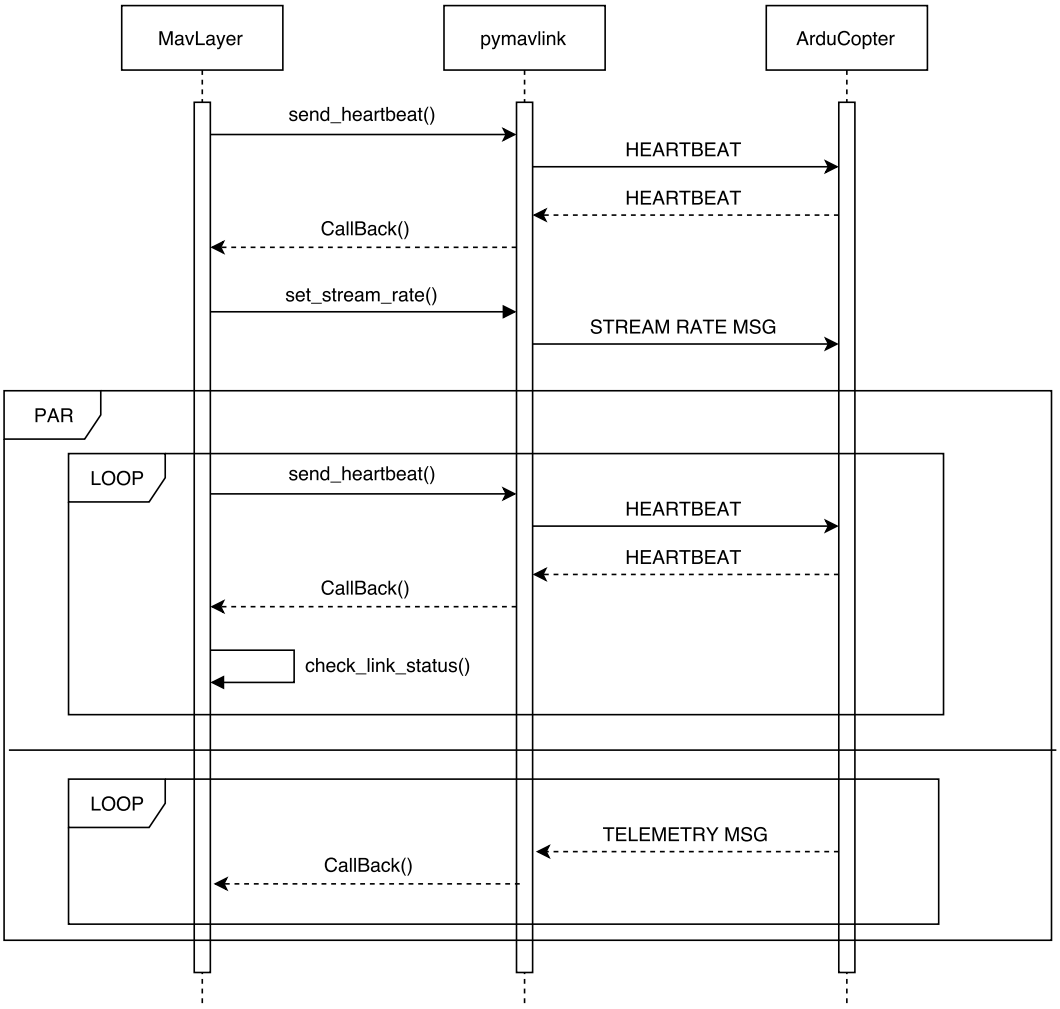


Figura 5.2: Flusso di messaggi scambiati tra il componente MavLayer e ArduCopter

Drone: Questo componente ha il compito di fornire alla logica applicativa delle API per pilotare il velivolo; in particolare, i metodi che esso dovrà mettere a disposizione sono quelli per conoscere lo stato del velivolo e per inviare i comandi di volo per pilotarlo.

Questo componente è stato realizzato tenendo in considerazione solamente la famiglia di APR che utilizzano come software di volo ArduCopter, pertanto costituisce uno dei limiti di IDrOS in quanto non permette di eseguire il nostro sistema su tutti i velivoli presenti sul mercato.

5.2.2.2 Gestione dei Sensori

Come illustrato nel Capitolo 4, una delle caratteristiche principali di IDrOS è quella di gestire molteplici tipologie di sensori attraverso un'interfaccia comune; per fare ciò, al momento dell'installazione di un sensore a bordo del sistema, l'utente

deve fornire ad IDrOS un driver per specificare come leggere i dati registrati dal sensore.

Nella nostra implementazione, ogni driver è un modulo Python contenente la definizione di una classe ed un metodo denominato *init*. La classe definita da questo modulo deve estendere la classe astratta *Sensor* messa a disposizione da IDrOS; in particolare deve essere definito il metodo *_read* , il quale contiene le istruzioni per leggere il sensore. Il metodo *init* deve istanziare e ritornare un oggetto della classe definita nel medesimo file.

Nel Listato 5.1 è illustrato un esempio di driver; in particolare, è mostrato quello per leggere la camera digitale utilizzata dalla board Raspberry Pi, una delle schede utilizzate come piattaforma embedded per eseguire IDrOS durante la fase di test (Sezione 5.3.1).

Per installare un driver su IDrOS è necessario salvare il modulo Python nella cartella contenente tutti i driver, tale cartella è identificata dal parametro *sensors_package* definito nel file di configurazione del sistema.

Listato 5.1: Esempio di un driver utilizzabile con IDrOS

```
1  class RaspiCam(Sensor):
2
3    def __init__(self):
4      Sensor.__init__(self,'raspicam')
5
6    def _read(self):
7      fname = path.dirname(sys.argv[0]) + "/tmp/img.jpg"
8      system("raspistill -w 960 -h 540 -o " + fname)
9      img = open(fname)
10     imagestring = img.read()
11     datapayload = base64.b64encode(imagestring)
12     img.close()
13     return datapayload
14
15 def init():
16   return RaspiCam()
```

IDrOS prevede che la lettura dei dati provenienti da un sensore possa essere effettuata quando richiesto dall'utente oppure periodicamente. Nel primo caso la gestione risulta semplice perché per leggere il sensore bisognerà utilizzare lo specifico driver invocando il metodo *_read*; nel secondo caso invece, per ogni sensore, IDrOS mette a disposizione una procedura che periodicamente provvede ad aggiornare lo stato del sensore e notificare attraverso una callback tale aggiornamento al livello

superiore (quello della logica applicativa) - l'intervallo di tempo con cui campionare periodicamente il sensore deve essere specificato nel file di configurazione di IDrOS.

Per semplificare l'interazione con il livello superiore abbiamo deciso che la logica applicativa non deve utilizzare direttamente i driver per leggere i sensori ma abbiamo implementato un ulteriore componente software per svolgere questi compiti.

Tale componente è denominato *Sensori* (Figura 4.1): il suo compito è quello di gestire tutti i driver installati a bordo del sistema ed esporre delle API per poterli utilizzare correttamente. In particolare, tale componente, all'avvio di IDrOS, dovrà preoccuparsi di importare tutti i moduli presenti nella cartella dei driver. Una volta conclusa questa prima fase, dovrà fornire al livello che implementa la logica applicativa i metodi per conoscere quali sono i sensori installati a bordo e per leggere i dati da uno specifico sensore.

5.2.3 Logica Applicativa

Un elemento di forza di IDrOS è quello di fornire delle interfacce Internet per permettere all'utente di comunicare direttamente con l'APR: è infatti proprio questa caratteristica quella che permette al nostro sistema di poter essere integrato con molti sistemi informatici. Dato che desideriamo fornire all'utente la possibilità di creare dei nuovi binding protocollari rispetto a quelli da noi sviluppati ed illustrati alla Sezione 5.2.4, ci è sembrato limitante vincolare l'utente a sviluppare questi binding con lo stesso linguaggio di programmazione utilizzato per implementare IDrOS. Pertanto abbiamo dovuto trovare una soluzione che permettesse a diversi binding, scritti con linguaggi di programmazione differenti, di interagire con il livello della logica applicativa.

Questo è stato il primo problema che abbiamo dovuto affrontare prima di sviluppare lo strato software contenente la logica applicativa; la soluzione da noi individuata è stata quella di far comunicare i due livelli (logica applicativa ed interfaccia Internet) attraverso l'interfaccia di rete in *loopback* utilizzando un protocollo di rete da noi sviluppato. L'utilizzo dell'interfaccia *loopback* per mettere in comunicazione i binding con la logica applicativa è la soluzione più semplice ma sopratutto è la più portabile, essa infatti si può utilizzare con la maggioranza dei sistemi operativi presenti sul mercato.

I messaggi che vengono utilizzati dal protocollo da noi sviluppato contengono informazioni in formato *json* per permettere la comunicazione tra i due livelli (logica applicativa ed interfaccia Internet). In particolare, il formato dei messaggi per accedere alle funzionalità che offre la logica applicativa prevede due campi:

handler: Questo campo contiene una stringa che rappresenta il nome del gestore (Figura 4.1) che eroga la funzionalità richiesta alla logica applicativa; i valori che può assumere sono le stringhe: *mission*, *module*, *sensor*, *failsafe*.

payload: Il *payload* contiene ulteriori campi che variano a seconda del gestore richiesto; nelle sezioni qui di seguito verranno illustrati nel dettaglio i valori che possono essere presenti in questo campo.

Una volta ricevuto un messaggio, il gestore specificato dal campo *handler* del messaggio deve eseguire le operazioni richieste e rispondere al mittente con un messaggio contenente i risultati dell'operazione.

Di seguito sono descritti i quattro gestori che compongono lo strato della logica applicativa facendo particolare attenzione alle funzionalità che devono erogare ed ai campi che è necessario specificare nel campo *payload* del messaggio per poterli utilizzare. Come già anticipato nel Capitolo 4 la scelta di creare questi quattro componenti software è stata fatta per gestire separatamente le varie funzionalità messe a disposizione da IDrOS per il controllo del velivolo.

5.2.3.1 Gestore dei Moduli

Per utilizzare le funzionalità di questo componente è necessario inserire nel campo *handler* del messaggio la stringa *module*. Le funzionalità che esso offre sono quelle per la gestione dei moduli di navigazione e di campionamento ed elaborazione dati installati su IDrOS; in particolare, la nostra implementazione prevede che il gestore dei moduli permetta all'utente eseguire le seguenti operazioni:

- Caricare un modulo.
- Eliminare un modulo.
- Ottenere l'elenco dei moduli installati.
- Ottenere informazioni sul funzionamento di un modulo.

Ogni funzionalità sarà ora descritta con maggiore precisione.

Caricare un modulo: L'utente deve avere la possibilità, in qualsiasi momento, di caricare su IDrOS nuovi moduli di navigazione e di campionamento ed elaborazione dati al fine di poter fare eseguire al velivolo i compiti necessari per portare a termine una missione.

In Tabella 5.1 sono descritti i campi che devono essere specificati nel *payload* del messaggio che il livello delle interfacce Internet deve inviare al gestore dei moduli per poter caricare un nuovo file su IDrOS.

Tabella 5.1: Parametri da specificare per caricare un modulo su IDrOS

Nome del campo	Descrizione
action	Per effettuare l'upload di un modulo questo campo deve assumere il valore: *upload*
module_type	Questo campo deve assumere il valore *navigation* se si vuole caricare un modulo di navigazione oppure il valore *sampling* se si vuole caricare un modulo di campionamento ed elaborazione
filename	Questo campo indica il nome con cui verrà identificato il modulo da caricare
file	Questo campo contiene il modulo da caricare

Una volta ricevuta una richiesta per il caricamento di un modulo, il gestore dei moduli dovrà salvare il file ricevuto in una delle due cartelle destinate ad ospitare i moduli presenti nel sistema. Le cartelle dove salvare i moduli possono essere scelte dall'utente modificando il file di configurazione di IDrOS; in particolare dovranno essere settati i parametri *sampling_modules_package* e *navigation_modules_package* per indicare rispettivamente dove salvare i moduli di campionamento ed elaborazione dati ed i moduli di navigazione.

Nel Listato 5.2 è illustrata la struttura che deve essere utilizzata per la definizione dei moduli di navigazione mentre il Listato 5.3 mostra la struttura che devono avere i moduli di campionamento ed elaborazione per essere accettati da IDrOS.

Listato 5.2: Struttura del modulo di navigazione

```
1  class VisitAndReturn(Navigation):
2
3    def __init__(self, location):
4      Navigation.__init__(self)
5      loc_arr = location.split(',')
6      self.loc = Location(loc_arr[0], loc_arr[1], loc_arr[2])
7
8    def start(self):
9      self.drone.arm():
10     self.drone.takeoff(5)
11
12     self.drone.change_flightmode("guided")
13     self.drone.flight_mode.goto(self.loc)
14
15     self.drone.change_flightmode("rtl"):
16
17 def init(**kwargs):
```

```
18    return VisitAndReturn(**kwargs)
19
20  def description():
21    return "Module description..."
```

Come si può notare dal Listato 5.2, il modulo di navigazione deve essere composto dalla definizione di una classe e di due metodi: *init* e *description*.

La classe deve estendere la classe astratta *Navigation* definita da IDrOS; in particolare, l'utente dovrà implementare il metodo *start* contenente le istruzioni di volo da inviare all'APR. Nell'esempio sono presenti le istruzioni per far decollare il velivolo, fargli raggiungere una coordinata GPS e farlo atterrare.

Il metodo *init* deve istanziare e ritornare un oggetto della classe definita nel medesimo file.

Il metodo *description* deve ritornare una stringa contenente la descrizione del modulo di navigazione.

Listato 5.3: Struttura del modulo di campionamento ed elaborazione dati

```
1  class SamplingModuleExample(Sampling):
2
3    def __init__(self):
4      Sampling.__init__(self)
5
6    def condition(self):
7      return True
8
9    def compute_result(self):
10     return self.samples
11
12 def init(**kwargs):
13   return SamplingModuleExample(**kwargs)
14
15 def description():
16   return "Module description..."
```

Analogamente al modulo di navigazione, anche il modulo di campionamento ed elaborazione - di cui un esempio è riportato nel Listato 5.3 - deve contenere la definizione di una classe ed i metodi *init* e *description*.

In questo caso però la classe deve estendere la classe astratta *Sampling* e l'utente dovrà definire il metodo *condition,* il quale rappresenta la condizione da valutare per l'acquisizione dei campioni, e il metodo *compute_result,* che serve per elaborare i campioni raccolti. Nell'esempio mostrato nel Listato 5.3 la condizione ritorna sempre il valore *True* - ciò significa che IDrOS acquisirà continuamente i valori dal sensore

utilizzato per la missione; la funzione per l'elaborazione ritorna all'utente la lista di campioni raccolti senza effettuare nessuna elaborazione.

I metodi *init* e *description* sono analoghi a quelli usati per il modulo di navigazione.

Eliminare un modulo: L'utente può decidere in un qualsiasi momento di cancellare un modulo installato su IDrOS; in questo caso dovrà solamente inviare un messaggio al gestore dei moduli contenente, nel proprio *payload*, i campi illustrati in Tabella 5.2.

Tabella 5.2: Parametri da specificare per eliminare un modulo installato su IDrOS

Nome del campo	Descrizione
action	Per cancellare un modulo questo campo deve assumere il valore: *delete*
module_type	Questo campo deve assumere il valore *navigation o sampling* a seconda del fatto che il modulo da cancellare sia un modulo di navigazione oppure di campionamento ed elaborazione
filename	Questo campo indica il nome del modulo da eliminare

Ottenere l'elenco dei moduli installati: Per ottenere la lista di tutti i moduli installati su IDrOS, l'utente dovrà specificare nel campo *payload* i campi: *action* e *module_ type.* In particolare, il campo *action* dovrà contenere la stringa *list*, mentre il campo *module_ type* dovrà contenere una stringa per indicare la tipologia dei moduli a cui si è interessati: *navigation* o *sampling.*

Ottenere informazioni sul funzionamento di un modulo: Qualora l'utente fosse interessato a conoscere il comportamento di un modulo può effettuare una richiesta al gestore dei moduli per ottenere informazioni su come utilizzare il modulo e sul suo comportamento. Il formato del messaggio che dovrà essere inviato per ottenere queste informazioni dovrà contenere nel *payload* i seguenti campi: *action*, *module_ type* e *filename*; in particolare, *action* dovrà contenere la stringa *info, module_ type* dovrà contenere la stringa *navigation* o *sampling* a seconda della tipologia di modulo e *filename* dovrà contenere la stringa contenente il nome del modulo di cui si vogliono ottenere informazioni.

5.2.3.2 Gestore della Missione

Come citato nel Capitolo 4 il gestore della missione è il componente responsabile di eseguire la logica applicativa per permettere all'APR di volare, di acquisire dati dai sensori e di compiere delle elaborazioni sui dati raccolti.

Per poter utilizzare le funzionalità messe a disposizione da questo gestore bisogna settare il campo *handler* del messaggio del nostro protocollo di comunicazione tra logica applicativa e binding protocollari con la stringa *mission*, mentre i campi da specificare nel campo *payload* sono elencati in Tabella 5.3.

Tabella 5.3: Formato del messaggio accettato dal gestore della missione

Nome del campo	Descrizione
navigation_module	Stringa contenente il nome del modulo di navigazione da utilizzare
navigation_params	Lista di parametri da passare al modulo di navigazione
sampling_module	Stringa contenente il nome del modulo di campionamento ed elaborazione dati da utilizzare
sampling_params	Lista di parametri da passare al modulo di campionamento ed elaborazione dati
sensor	Stringa contenente il nome del sensore da utilizzare per il campionamento
loop	Se posto a *true* indica che la missione deve essere ciclica; *false* altrimenti

Dopo aver ricevuto un messaggio dall'utente, questo gestore deve preoccuparsi di mandare in esecuzione la missione; in particolare, la gestione dell'APR deve essere effettuata secondo il modello presentato nel Capitolo 3. In Figura 5.3 è mostrato un diagramma di flusso che illustra nei dettagli le operazioni svolte dal gestore delle missioni.

Come si può notare la prima operazione che viene eseguita è quella di inizializzare la missione: tramite questa operazione vengono caricati i moduli di navigazione e di campionamento ed elaborazione richiesti dall'utente, inoltre, viene verificato che non sia in esecuzione alcuna missione. Nel caso in cui uno dei due moduli non è presente oppure IDrOS sta eseguendo un'altra missione viene ritornato un errore all'utente.

L'inizializzazione dei moduli viene fatta specificando i parametri passati dall'utente relativi al modulo da caricare; tale caratteristica è fondamentale per il riuso del codice. Ad esempio, per i nostri test (Sezione 6.2) abbiamo creato un modulo di navigazione per permettere di pilotare il drone specificando, attraverso il parametro *navigation_params*, una lista di *waypoints*; così facendo è possibile riutilizzare lo stesso modulo in più circostanze.

Il controllo che non sia in esecuzione un'altra missione è fondamentale per non creare conflitti: infatti se venissero lanciate due missioni contemporaneamente sarebbero in esecuzione due moduli di navigazione che invierebbero comandi contraddittori all'APR, ciò causerebbe sicuramente dei problemi durante il volo.

Se la procedura di inizializzazione della missione termina senza errori viene mandato in esecuzione il modulo di navigazione il quale inizia ad inviare i comandi di volo

all'APR. Finché il modulo di navigazione è in esecuzione il sistema verifica continuamente la condizione contenuta nel modulo di campionamento ed elaborazione; ogni qualvolta questa ha esito positivo viene acquisito un campione dal sensore specificato dall'utente attraverso il parametro *sensor*. I campioni acquisiti sono costituiti da tre valori: *timestamp*, coordinata geografica e valore letto dal sensore.

Una volta che il *thread* che esegue il modulo di navigazione termina, il sistema applica ai dati riaccolti la procedura di elaborazione contenuta nel modulo di campionamento ed elaborazione ed invia all'utente i risultati della missione.

Se l'utente setta il parametro *loop* con il valore *True* significa che desidera eseguire una missione ciclica perciò IDrOS, dopo aver inviato i risultati della missione, rilancia la missione.

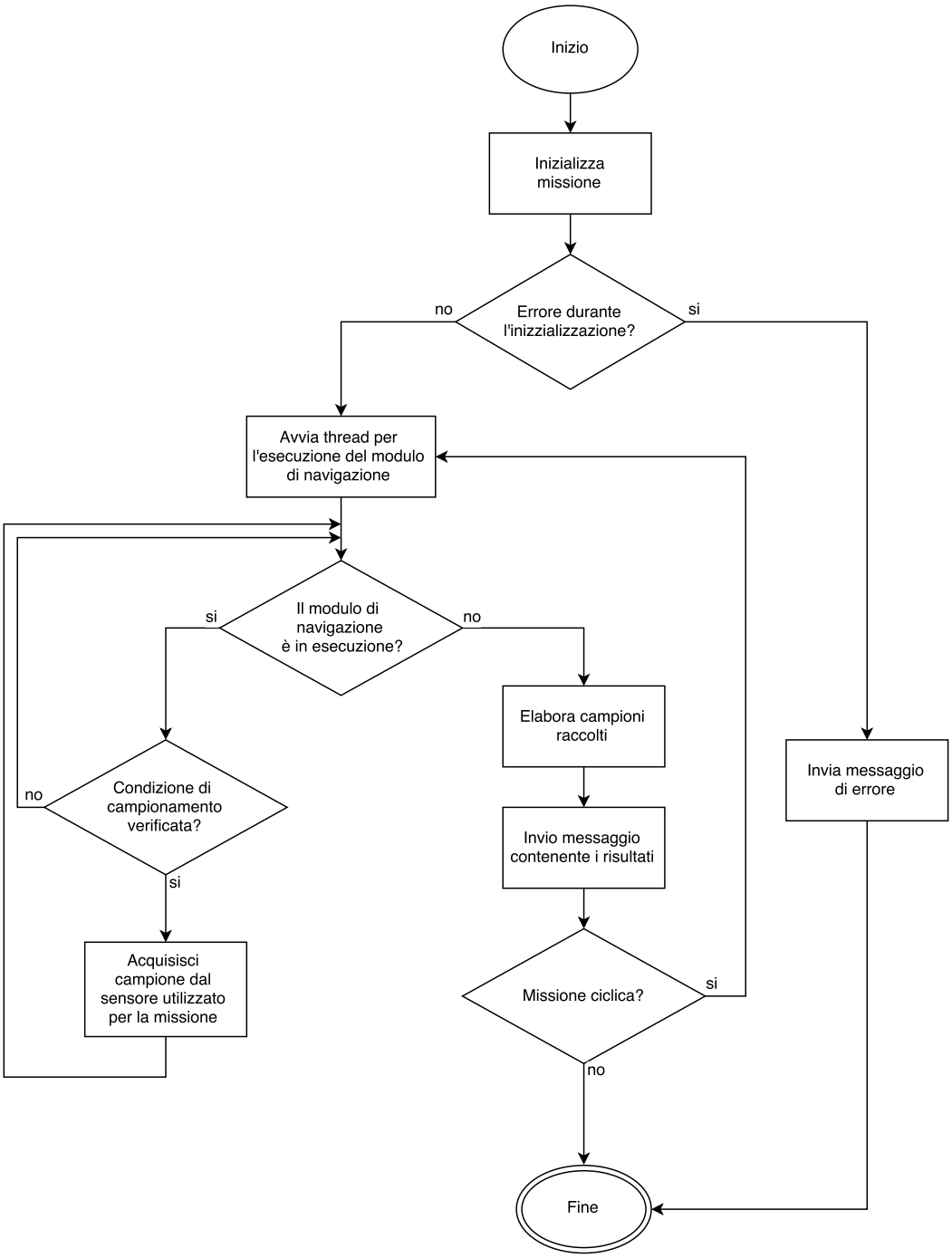


Figura 5.3: Diagramma di flusso relativo al funzionamento del gestore della missione

5.2.3.3 Gestore dei Sensori

Questo gestore, identificato con la stringa *sensor* del campo *handler* del messaggio, permette di visualizzare i valori letti da un sensore installato a bordo dell'APR. Come citato alla Sezione 4.2.2 del capitolo precedente, è possibile visualizzare i dati registrati in due modi differenti: il primo attraverso una normale richiesta per visua-

lizzare la misura istantanea, il secondo sottoscrivendosi alla ricezione della misura registrata ogni qualvolta il sensore cambia il proprio stato.

Oltre a fornire questi due metodi per poter accedere ai sensori, abbiamo deciso che questo componente dovesse fornire anche delle procedure per informare l'utente sull'elenco dei sensori installati su IDrOS.

Di seguito vengono descritti i campi da specificare nel *payload* del messaggio per utilizzare le varie funzionalità messe a disposizione da questo componente.

Lettura di un sensore: In questo caso l'utente è interessato a conoscere il valore attuale di un sensore, per far ciò dovranno essere specificati i seguenti parametri:

- *action:* questo campo dovrà contenere la stringa *read.*
- *sensor:* questo campo dovrà contenere la stringa che indica il nome del sensore a cui si è interessati.

Monitoraggio di un sensore: A differenza del caso precedente l'utente è interessato a ricevere la misura registrata da un sensore ogni qualvolta questo venga aggiornato; in questo caso i campi che dovranno essere specificati nel *payload* sono i seguenti:

- *action:* questo campo dovrà contenere la stringa *observe.*
- *sensors:* questo campo contiene una lista di sensori che l'utente desidera monitorare; in particolare la lista è formata da coppie di valori indicanti il nome del sensore e l'intervallo di tempo, espresso in millisecondi, che deve passare tra un aggiornamento e l'altro. Ad esempio questo campo può assumere il valore *[['gps',1000], ['temperature',2000]]* per indicare che l'utente è interessato a monitorare il sensore GPS e il sensore di temperatura e ricevere le notifiche ogni 1000ms nel caso del GPS e 2000ms nel caso del sensore di temperatura.

Ottenere l'elenco dei sensori installati a bordo: Per ottenere la lista di tutti i sensori che possono essere utilizzati, il campo *payload* del messaggio inviato a questo gestore dovrà contenere solamente il campo *action* contenente la stringa *list.*

Una volta ricevuto questo messaggio il gestore dei sensori effettuerà una richiesta al livello di astrazione hardware per conoscere quali sono i sensori installati a bordo e inoltrerà il risultato all'utente.

5.2.3.4 Gestore Fail-safe

L'ultimo gestore che compone il livello della logica applicativa è quello che offre le funzionalità per gestire l'aeromobile in situazioni di pericolo per il volo. Questo componente, una volta ricevuto un messaggio dall'utente, interrompe l'esecuzione

della missione e fa atterrare immediatamente l'APR. Il *payload* del messaggio deve contenere solamente il campo *action* il quale può contenere una delle seguenti stringhe:

- *land*: in questo caso l'APR viene fatto atterrare nel punto in cui si trova dopo aver interrotto l'esecuzione della missione.
- *rtl* (*return to launch*): specificando questo valore, una volta interrotta la missione, l'aeromobile viene fatto atterrare nel punto in cui è decollato.
- *reset*: questo paramento serve per interrompere la procedura di *fail-safe* attivata tramite i comandi *land* o *rtl* così da non far atterrare il drone.

5.2.4 Interfaccia Internet

L'ultimo strato software che compone IDrOS è quello incaricato di fornire delle interfacce Internet per permettere all'APR di interagire con tutti i dispositivi presenti nella rete in cui opera. Questo livello è caratterizzato dalla presenza di vari componenti che rappresentano i binding tra diversi protocolli di rete e il livello in cui risiede la logica applicativa necessaria a pilotare il velivolo. La nostra implementazione è caratterizzata dalla presenza di due binding protocollari di esempio: CoAP e MQTT. La scelta di implementare i binding di questi due protocolli - oltre al fatto che essi sono quelli più utilizzati nel contesto dell'IoT - risiede nel fatto che essi operano con dei paradigmi totalmente diversi: *client-server* per CoAP e *publish/subscribe* per MQTT; pertanto, riuscire ad utilizzarli entrambi per interfacciarsi IDrOS mostra come il nostro sistema risulti flessibile e facilmente integrabile con una grande varietà di sistemi informatici.

Di seguito descriveremo come abbiamo realizzato il binding per questi due protocolli.

5.2.4.1 CoAP

Come descritto nel Capitolo 2 (Sezione 2.1.1) la caratteristica principale di CoAP è quella di essere un protocollo basato sul paradigma *client-server*: il server mette a disposizione delle risorse, ovvero delle fonti di informazioni, a cui i client possono accedere attraverso vari metodi.

La presenza di risorse è stato un elemento chiave per lo sviluppo di questo binding: infatti, quello che abbiamo sviluppato altro non è che un server CoAP che eroga ai vari client svariate risorse per interagire con IDrOS; in particolare, le risorse che il server deve offrire sono quattro, una per ogni componente della logica applicativa. Come vedremo in seguito, le quatto risorse implementate possono contenere delle sotto-risorse così da semplificare l'interazione con l'utente.

Prima di descrivere nel dettaglio le varie risorse offerte da questo binding, vogliamo focalizzare l'attenzione sul framework che abbiamo utilizzato per implementare il binding CoAP; in particolare, per il nostro progetto abbiamo deciso di utilizzare Californium (Cf) [22].

Californium è un framework scritto in java che implementa le specifiche CoAP relative all'RFC 7252 [39]; la scelta di utilizzare questo framework per realizzare il binding CoAP risiede nel fatto che Californium implementa tutte le funzionalità descritte dal documento di specifica del protocollo, lasciando all'utente il compito di sviluppare solamente le risorse che dovranno essere disponibili ai client.

Di seguito sono illustrare le risorse sviluppate per realizzare il binding CoAP.

drone: questa è la risorsa che permette ai vari dispositivi operanti nella stessa rete dell'APR di usufruire delle funzionalità messe a disposizione dal *gestore delle missioni*, ovvero permette di pilotare il drone per svolgere i compiti richiesti dall'utente. Nella nostra implementazione, come mostrato in Figura 5.4, questa risorsa è composta da un dato numero di sotto-risorse, una per ogni sensore installato su IDrOS.

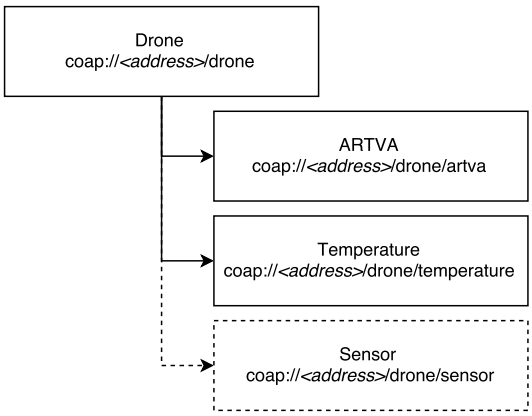


Figura 5.4: Esempi di sotto-risorse CoAP per la risorsa *drone*

Per poter lanciare una missione, l'utente dovrà effettuare una GET alla sotto-risorsa rappresentante il sensore da usare per il campionamento specificando come parametri i seguenti valori: *navigation_module, navigation_params, sampling_module, sampling_params, loop.* I valori che possono assumere questi parametri sono stati descritti in Tabella 5.3.

Dato che una missione generalmente richiede del tempo per essere eseguita, per questo tipo di risorsa abbiamo deciso di utilizzare la modalità *separate response* per rispondere al mittente; così facendo verrà inviato immediatamente un'*ack* al client per notificare di aver ricevuto il messaggio ed a missione terminata verrà inviato un messaggio contenente i risultati della missione.

Inoltre, abbiamo scelto che ogni sotto risorsa potesse gestire le richieste con l'opzione *observe* di CoAP (Sezione 2.1.1.2). Questa scelta è stata fatta perché come detto precedentemente (Sezione 5.2.3.2), il drone può eseguire solo una missione alla volta; pertanto, una volta che un client ha eseguito una GET gli altri client non possono far eseguire al drone nuove missione finché quella in esecuzione non termina.

Facendo una richiesta GET con l'opzione *observe*, senza specificare alcun parametro, eventuali client interessati all'esito della missione in esecuzione possono ricevere il messaggio con i risultati della missione.

Inoltre, l'utilizzo del opzione *observe* è indispensabile quando si vuole eseguire una missione ciclica per ricevere i risultati delle varie esecuzioni; senza questo tipo di opzione non sarebbe possibile inviare al client i risultati al temine di ogni singola esecuzione ma solamente al temine della missione ciclica. Ciò è un problema per le missioni che durano molto tempo in quanto l'utente non avrebbe nessun riscontro immediato sull'andamento della missione.

modules: questa risorsa è quella utilizzata per interfacciare i vari client con il *gestore dei moduli*; in particolare essa è composta da due sotto-risorse: *navigation* per gestire i moduli di navigazione e *sampling* per gestire i moduli di campionamento ed elaborazione dati.

Queste due sotto-risorse permettono all'utente di eseguire i metodi GET e POST: utilizzando il metodo GET la sotto-risorsa dovrà rispondere all'utente con un messaggio contenente tutti i moduli installati su IDrOS; utilizzando il metodo POST l'utente potrà caricare su IDrOS nuovi moduli - in questo caso il *payload* del messaggio dovrà contenere il file rappresentate il modulo che si intende caricare.

Una volta che un modulo viene caricato il sistema lo rappresenterà come una risorsa CoAP la quale mette a disposizione i metodi GET e DELETE: attraverso il metodo GET è possibile accedere alle informazioni riguardanti il funzionamento del modulo, mentre invocando il metodo DELETE si eliminerà il modulo e la risorsa che lo rappresenta.

sensors: tale risorsa è quella relativa al *gestore dei sensori*; in particolare essa è formata da tante sotto-risorse quanti sono i sensori installati su IDrOS. Ogni sotto risorsa identifica un sensore ed offre all'utente il metodo GET per leggere la grandezza fisica da esso misurata.

Nel file di configurazione di IDrOS, attraverso il parametro *coap_obs_sensors,* è possibile specificare una lista di sensori che possono essere monitorati attraverso il binding CoAP. Per tutti i sensori presenti in questa lista, la risorsa che li rappresenterà permetterà all'utente di utilizzare l'opzione *observe* nelle richieste così da poter notificare ai client tutti i cambiamenti di stato del sensore.

failsafe: questa risorsa è quella che offre all'utente le funzionalità messe a disposizione dal *gestore fail-safe* presente nel livello contenente la logica applicativa. Essa è composta da tre sotto-risorse: *land*, *rtl* e *reset*; ognuna di queste tre risorse permette di utilizzare solamente il metodo GET per accedere alle funzionalità messe a disposizione - tali funzionalità sono state descritte nel Sezione 5.2.3.4.

5.2.4.2 MQTT

Come descritto nel Capitolo 2 (Sezione 2.1.2) MQTT è un protocollo basato sul paradigma *publish/subscribe*; tale caratteristica rende necessaria la presenza di un broker per gestire lo scambio di messaggi tra i vari client.

Rispetto al caso CoAP l'implementazione di questo binding è stata più semplice perché abbiamo deciso di utilizzare un broker già esistente e di implementare solamente un client MQTT per comunicare con la logica applicativa.

La scelta del broker non influenza lo sviluppo del binding perché esso deve sempre rispettare le specifiche MQTT [19]; nel nostro caso abbiamo scelto di utilizzare come broker Mosquitto [29].

Il funzionamento del binding MQTT che abbiamo sviluppato si basa sul concetto di incapsulare i messaggi diretti ad uno dei quatto gestori che compongono il livello della logica applicativa all'interno del *payload* del pacchetto PUBLISH di MQTT. Il client MQTT, che caratterizza il binding che abbiamo sviluppato, all'avvio del sistema si sottoscrive alla ricezione dei messaggi aventi come *topic* la stringa *command*; l'utente per utilizzare le funzionalità messe a disposizione da IDrOS dovrà pubblicare un messaggio con questo *topic* includendo nel *payload* il messaggio diretto ad uno dei gestori presenti al livello della logica applicativa (Sezione 5.2.3). Il formato dei possibili messaggi è stato descritto al Sezione 5.2.3.

Una volta eseguite le operazioni richieste, la logica applicativa ritornerà al client MQTT i risultati della missione il quale dovrà pubblicare un messaggio contenente tali risultati. In Figura 5.5 è schematizzato il funzionamento del binding MQTT: i messaggi in rosso sono i messaggi destinati al livello contenente la logica applicativa, quelli in blu sono i risultati ottenuti dall'esecuzione del compito richiesto; come si può notare essi vengono incapsulati nei pacchetti MQTT dal client implementato su IDrOS.

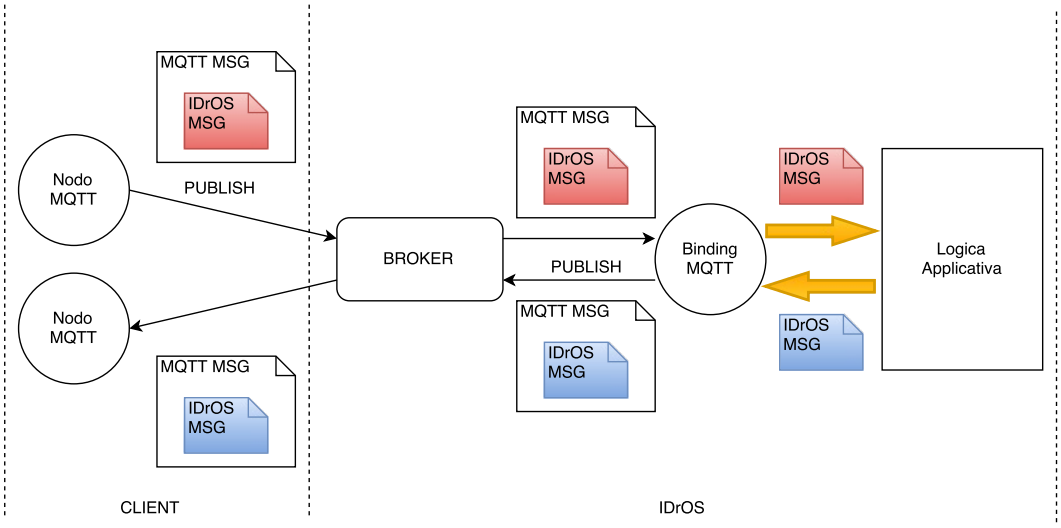


Figura 5.5: Principio di funzionamento del binding MQTT

5.3 Il Velivolo

In questa sezione viene illustrato il quadricottero (Figura 5.6) che abbiamo costruito per testare il funzionamento di IDrOS. Nella realizzazione del prototipo abbiamo dovuto tenere conto di molti aspetti pratici per far funzionare correttamente il nostro sistema nei casi reali; ad esempio, la connessione tra la scheda di supporto per eseguire IDrOS e la board Pixhawk.

La realizzazione di un prototipo funzionante è la dimostrazione che il sistema da noi realizzato è utilizzabile non solo in teoria ma anche nei casi reali: nel Capitolo 6 sono presentati alcuni esempi di applicazioni che abbiamo eseguito con il velivolo; in particolare, abbiamo testato il nostro sistema nello scenario della ricerca di dispersi sotto le valanghe e in uno scenario di videosorveglianza.

(a) Fase di costruzione del prototipo

(b) Dettaglio collegamenti

(c) Prototipo completato

(d) Prototipo durante un volo

Figura 5.6: Prototipo del velivolo realizzato per testare il funzionamento di IDrOS

Come citato nella Sezione 5.1, il componente principale per permettere il funzionamento degli APR è il controllore di volo: noi abbiamo scelto di utilizzare il firmware ArduCopter [5], pertanto il controllore di volo che abbiamo utilizzato è la board Pixhawk [34] - quella raccomandata dalla comunità che sviluppa ArduCopter.

Gli altri componenti necessari per la realizzazione del nostro velivolo sono:

- un frame: ovvero il telaio utilizzato per collocare i vari componenti elettronici e meccanici.
- quattro motori con relative eliche.
- una batteria.
- quattro controllori di velocità (ESC [1]).
- un radiocomando e un ricevitore radio[2].

[1] ESC: componenti che servono per controllare la velocità dei motori. Il controllore di volo (Pixhawk) infatti eroga un segnale PWM che indica la velocità dei motori; questo segnale è l'input che viene fornito ai controllori di velocità i quali, dopo averlo interpretato, regolano la corrente da fornire ai motori per farli ruotare alla velocità indicata.

[2] Nonostante IDrOS non necessiti di un radiocomando per pilotare il velivolo, per questioni di sicurezza abbiamo deciso di avere un radiocomando per poter controllare il velivolo in caso di emergenza.

Questi componenti sono quelli necessari per realizzare un normale quadricottero, tuttavia, essi non sono sufficienti per ottenere un velivolo in grado di volare autonomamente ed utilizzare IDrOS. Per questo motivo abbiamo dovuto equipaggiare il nostro velivolo con ulteriori dispositivi; di seguito illustreremo tali componenti e giustificheremo perché sono necessari.

5.3.1 Scheda Embedded

Come già spiegato in questo capitolo, abbiamo scelto di utilizzare la configurazione di *deployment* con IDrOS installato a bordo del velivolo (Figura 4.2) così da rendere l'APR un componente dell'IoT; pertanto è stato necessario utilizzare una scheda aggiuntiva per poter eseguire IDrOS.

Non è stato possibile installare IDrOS sulla scheda Pixhawk perché essa è dotata di un sistema operativo monoprocesso: dato che questa board deve già eseguire il software responsabile della stabilità del velivolo (ArduCopter), siamo stati costretti ad installare sul drone un'ulteriore scheda per gestire l'esecuzione di IDrOS.

La scelta della scheda embedded di supporto è stata fatta analizzando varie soluzioni che sono presenti sul mercato, in particolare i paramenti che hanno dettato tale scelta sono stati la potenza di calcolo ed il peso. Il consumo energetico delle soluzioni analizzate non ha inciso sulla scelta in quanto esso è sempre inferiore ai 5W: dato che il consumo di ogni singolo motore utilizzato è di 210W, possiamo considerare questo valore trascurabile.

Tabella 5.4: Schede embedded presenti sul mercato. Informazioni reperite da Adafruit [3]

Nome scheda	Perso (gr)	Consumo Medio (W)	Processore	Frequenza processore (MHz)	Ram (MB)
BeagleBone Black	40	2,3	ARM Cortex-A8	1000	512
Arduino Yún	32	1,5	MIPS 32 24K	400	64
Arduino Intel Galileo	50	2,8	Intel Quark X1000	400	256
Raspberry Pi b+	45	2,5	ARM v6	700	512

In Tabella 5.4 sono illustrati i vari dispositivi che sono state presi in considerazione per essere utilizzati. Come si può notare il dispositivo che pesa meno è Arduino Yún, tuttavia questa scheda ha una potenza di calcolo inferiore rispetto alle altre. Per questa ragione abbiamo deciso di non utilizzare questo dispositivo e di adottare come scheda di riferimento per i nostri test la board BeagleBone Black. Oltre ad avere un peso ed un consumo energetico medio inferiore rispetto alle rivali la scheda

BeagleBone Black [9] è stata scelta per gli sviluppi futuri del software ArduCopter: ciò costituirebbe un enorme vantaggio in futuro in quanto si potrebbe installare sia IDrOS che ArduCopter sullo stesso dispositivo - così facendo si otterrebbe una soluzione più compatta ed il velivolo sarebbe in grado di portare con sé un payload maggiore.

Come si vedrà nel Capitolo 6, per un'applicazione abbiamo deciso di utilizzare la scheda Raspberry Pi [37] in quanto fornisce la possibilità di utilizzare una camera digitale ad alta definizione poco ingombrante e molto leggera; tale camera è stata sviluppata appositamente per questo dispositivo e non è interfacciabile con le altre schede che abbiamo analizzato.

Sia la board BeagleBone Black che la board Raspberry Pi utilizzano come sistema operativo una distribuzione di Debian, sono dotate di connettore RJ45 per il collegamento ethernet, sono dotate di porte USB ed offrono la possibilità di collegare ad esse molti sensori grazie a numerosi pin GPIO e svariatati bus seriali (come I2C e CAN). Queste caratteristiche rendono entrambe le schede idonee per essere utilizzate come piattaforme su cui eseguire IDrOS.

Connessione con Pixhawk: La connessione tra la scheda per eseguire IDrOS (BeagleBone Black o Raspberry Pi) e la board Pixhawk è stata fatta realizzando un apposito cavo seriale il quale è stato collegato alla porta *Telem2* della board Pixhawk e ad una delle porte seriali della scheda di supporto per l'esecuzione di IDrOS. In Figura 5.7 è mostrato come è stato realizzato il collegamento tra BeagleBone Black e Pixhawk.

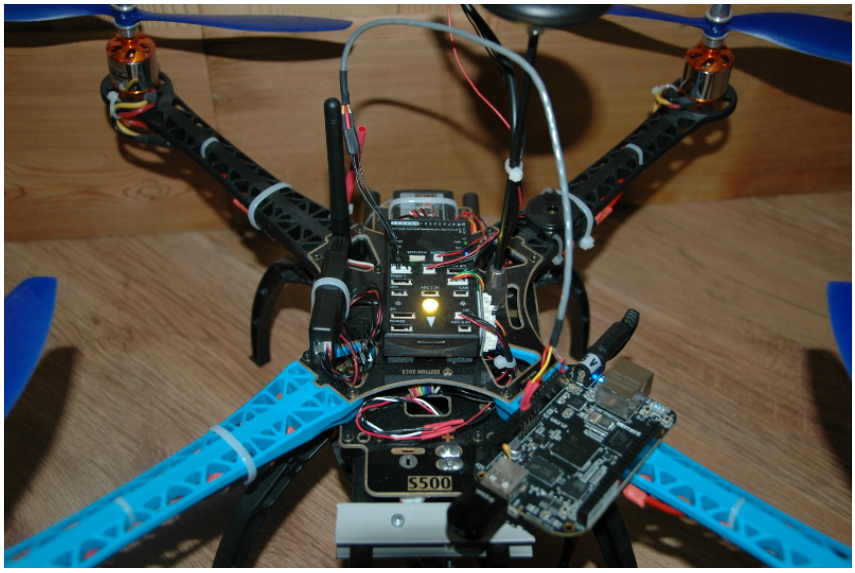


Figura 5.7: Collegamento tra BeagleBone Black e Pixhawk tramite il cavo seriale appositamente realizzato

5.3.2 GPS

Per rendere la navigazione del velivolo autonoma è stato necessario predisporre a bordo dell'aeromobile un sensore GPS: grazie a questo dispositivo IDrOS è in grado di localizzare l'APR e di conseguenza pilotare il velivolo per fargli raggiungere i punti di interesse per completare le varie missioni.

Il sensore GPS che abbiamo deciso di utilizzare è quello consigliato dalla comunità di ArduCopter (3DR u-blox GPS with Compass [1]); esso è stato collegato alla board Pixhawk tramite il bus I2C. IDrOS avrà accesso ai dati provenienti da questo sensore grazie al collegamento MAVLink tra le schede BeagleBone e Pixhawk.

5.3.3 Sonar

Il sonar non è un componente necessario per la navigazione ma rende il velivolo molto più stabile per quanto riguarda la quota di volo. Puntando il sonar verso il basso ArduCopter è in grado di misurare la distanza che c'è tra il velivolo ed il suolo così da regolare la potenza da erogare ai motori affinché questa rimanga costante.

In assenza del sonar ArduCopter usa l'altimetro integrato sulla scheda Pixhawk per calcolare la distanza dal suolo; i valori altimetrici sono però meno precisi rispetto a quelli provenienti dal sonar causando così alcuni inconvenienti durante la navigazione: il velivolo risulta meno stabile durante il volo e presenta problemi durante l'atterraggio perché la vicinanza delle eliche al suolo causa degli aumenti di pressione che falsano il valore letto dall'altimetro provocando un atterraggio brusco.

Il sonar che abbiamo utilizzato è il modello *MB1242 I2CXL-MaxSonar-EZ4* della Maxbotix [27]: esso si interfaccia con la scheda Pixhawk tramite il protocollo I2C ed è in grado di registrare altezze fino a 7 metri dal suolo.

Capitolo 6

Casi di Studio e Valutazione

In questo capitolo verranno presentati alcuni casi di studio che abbiamo effettuato per valutare le performance e le funzionalità che IDrOS mette a disposizione.

Inizialmente verranno illustrati gli strumenti che abbiamo utilizzato per eseguire gli esperimenti: in particolare verrà presentato il simulatore ed i vari client che abbiamo utilizzato per comunicare con il drone attraverso le interfacce Internet che IDrOS mette a disposizione.

Successivamente verranno presentate due applicazioni di esempio realizzate per valutare le prestazioni del nostro sistema durante l'esecuzione delle missioni; tali applicazioni sono state testate sul prototipo da noi costruito, e presentato nel Capitolo 5.

Infine, vengono presentati dei test effettuati al simulatore per valutare la scalabilità di IDrOS al variare del carico di lavoro richiesto: in particolare sono stati effettuati degli esperimenti per valutare come si comporta il sistema al variare della frequenza di campionamento dei sensori, del numero di sensori installati e del numero di utenti connessi ad IDrOS.

6.1 Strumenti Utilizzati

In questa sezione descriveremo gli strumenti utilizzati nelle applicazioni che abbiamo realizzato e nei vari test che abbiamo eseguito per analizzare le performance di IDrOS.

Per quanto riguarda gli esempi di applicazioni effettuati, abbiamo deciso di eseguire vari test sul prototipo da noi realizzato (Sezione 5.3) così da avere dei risultati più veritieri per quanto riguarda le performance e le funzionalità che IDrOS mette a disposizione. Tuttavia, alcuni test non potevano essere fatti utilizzando il velivolo reale perché miravano a cercare i limiti del sistema, pertanto ci è sembrato più opportuno utilizzare un simulatore così da non recare danni al velivolo qualora il

sistema non avesse supportato il carico di lavoro richiesto; inoltre, tale approccio è più pratico rispetto a quello di condurre dei voli reali.

Di seguito descriveremo in dettaglio gli strumenti utilizzati negli esperimenti, illustrati nella Sezione 6.2, che abbiamo effettuato per valutare le performance di IDrOS.

6.1.1 Simulatore

Durante l'esecuzione di alcuni test per misurare le performance di IDrOS abbiamo utilizzato SITL (Software In The Loop) ovvero il simulatore che la comunità di ArduCopter ha sviluppato per poter eseguire il software di controllo degli APR sui normali PC o laptop così da poter valutare il comportamento dei velivoli senza farli volare.

SITL, come mostrato in Figura 6.1, è composto da tre schermate principali:

- Command Prompt: grazie a questa *shell* è possibile inviare i comandi di volo all'APR.
- Console: tramite questa schermata è possibile visualizzare tutti i parametri riguardanti il volo. Ad esempio: l'altezza dal suolo, il livello della batteria, la modalità di volo utilizzata e tutti i dati riguardanti l'assetto del velivolo.
- Map: tramite questa schermata è possibile localizzare l'APR in un area geografica precisa.

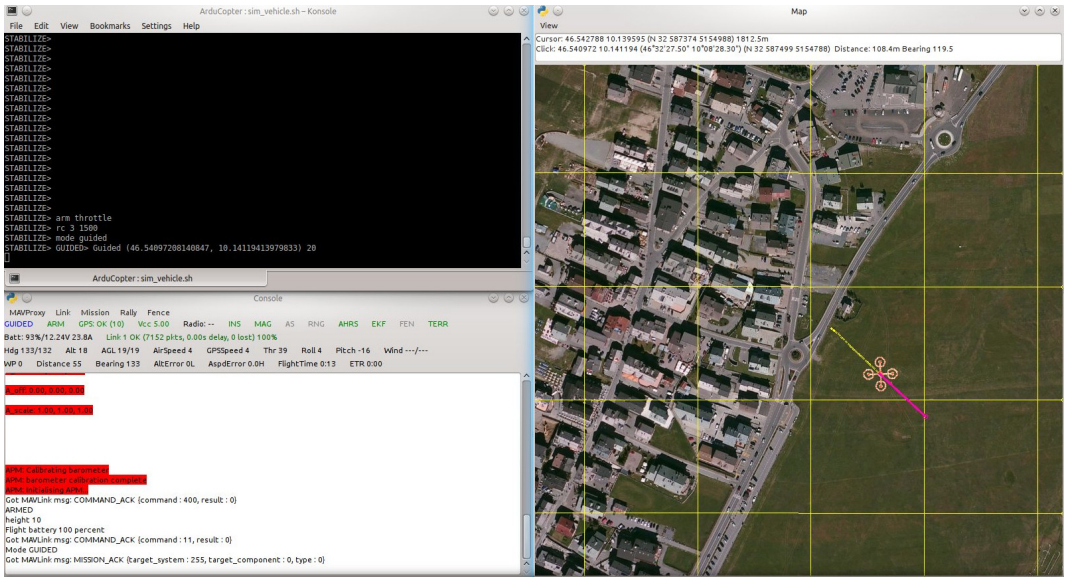


Figura 6.1: Schermate principali del simulatore SITL

Per poter controllare il velivolo simulato, oltre al Command Prompt, SITL mette a disposizione 3 socket per permettere l'interfacciamento con l'APR attraverso il protocollo MAVLink.

L'esecuzione dei test al simulatore è stata fatta nella configurazione *hardware in the loop*, ovvero collegando tramite un normale cavo di rete la scheda BeagleBone sulla quale era in esecuzione IDrOS al PC dove era presente il simulatore. Per far sì che IDrOS comunicasse correttamente con il simulatore abbiamo dovuto modificare il file di configurazione di IDrOS specificando che il collegamento al velivolo doveva essere fatto utilizzando il socket e non il normale link seriale utilizzato per comunicare con il velivolo reale. Nel Listato 6.1 è mostrato il file di configurazione con le impostazioni per comunicare con il simulatore; in questo caso abbiamo dovuto settare il parametro *mavlink_device* con l'indirizzo IP della macchina su cui era installato SITL ed indicare la porta utilizzata per la comunicazione.

Listato 6.1: File di configurazione di IDrOS per l'interfacciamento con SITL

```
# Mavlink Device
# Examples:
# /dev/ttyACM0,115200 -> usb cable to pixhawk
# /dev/ttyUSB0,57600  -> telemetry to pixhawk
# tcp:127.0.0.1:5763' -> sitl local
# tcp:172.16.0.1:5763 -> sitl remote
mavlink_device = tcp:172.16.0.1:5763
```

6.1.2 Client CoAP

Un altro strumento necessario durante l'esecuzione delle missioni di prova è stato Copper (Cu) [23] ovvero un client per poter comunicare con il velivolo utilizzando il protocollo CoAP.

La scelta di utilizzare Copper (Cu) come client deriva dalla sua semplicità di utilizzo; esso infatti è un plugin per il browser FireFox che offre un'interfaccia intuitiva per comunicare con un server CoAP. Una volta installato, basta digitare nella barra degli indirizzi di FireFox l'URL del server CoAP a cui ci si vuole collegare; a questo punto Copper fornisce una pagina per gestire la comunicazione con il server attraverso il protocollo CoAP. In particolare Copper permette di:

1. Visualizzare tutte le risorse CoAP che il server mette a disposizione.
2. Personalizzare le opzioni di ogni messaggio CoAP inviato.
3. Visualizzare l'*header* completo ed il *payload* di ogni messaggio ricevuto.

In Figura 6.2 è mostrata una schermata di Copper.

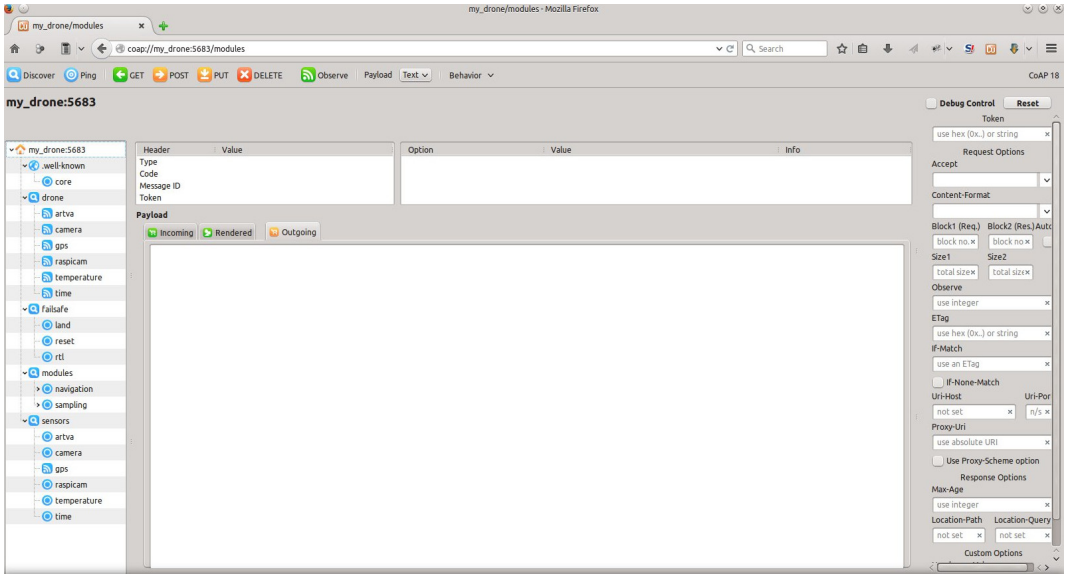


Figura 6.2: Schermata di Copper (Cu)

6.1.3 Client MQTT

Per la comunicazione con il drone attraverso l'utilizzo del protocollo MQTT abbiamo deciso di utilizzare Node-RED [32].

Node-RED è un editor *web-based* che permette di realizzare programmi secondo il paradigma *Flow-Based*: questo paradigma permette al programmatore di scrivere i propri programmi creando un grafo orientato; ogni nodo del grafo è un blocco funzionale che accetta dei dati in input e dopo averli elaborati produce un messaggio in output che viene dato in ingresso al nodo successivo. Secondo questo paradigma, un programma non è più una sequenza di istruzioni ma è caratterizzato da un insieme di flussi di dati che vengono scambiati tra vari nodi in maniera completamente asincrona.

La scelta di utilizzare Node-RED per interfacciare l'utente con il drone è stata dettata da due motivi principali: il primo perché Node-RED mette a disposizione i nodi per utilizzare il protocollo MQTT - in particolare è possibile utilizzare un nodo per pubblicare messaggi ed uno per sottoscriversi a vari *topic*; il secondo perché è possibile realizzare dei programmi dove il drone è visto come un componente di un'applicazione più complessa - ad esempio è possibile creare dei flussi dove i dati provenienti dal drone sono memorizzati su un database in una macchina remota oppure sono pubblicati sui social network.

In Figura 6.3 è mostrato un esempio di applicazione realizzata con Node-RED. Il nodo denominato *Start*, quando azionato dall'utente, genera un messaggio all'uscita privo di contenuto il cui scopo è solamente quello di attivare il nodo successivo. Il nodo *Mission Request*, una volta ricevuto un messaggio in input, produce un messaggio in uscita contenente i parametri necessari ad IDrOS per eseguire una missione;

in particolare, dovranno essere specificati specificati: il modulo di navigazione, il modulo di campionamento ed elaborazione ed il sensore da utilizzare. Il nodo *json* converte il contenuto del messaggio in ingresso in formato json. Infine, il nodo denominato *Mqtt* si occupa di effettuare l'operazione di PUBLISH comunicando al broker il messaggio che riceve in ingresso. Il nodo verde, denominato *msg.payload*, è un nodo di debug che permette all'utente di visualizzare il messaggio che riceve in ingresso.

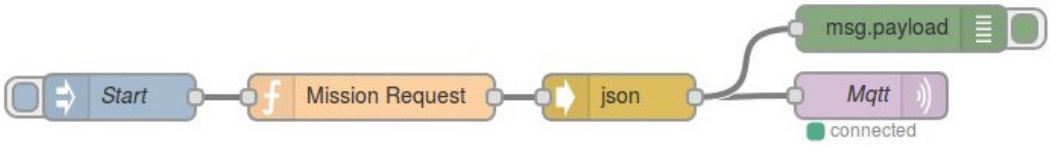


Figura 6.3: Esempio di applicazione realizzata con Node-RED

6.1.4 SysStat

Durante l'esecuzione degli esperimenti abbiamo valutato il consumo di risorse da parte di IDrOS: dato che eravamo interessati a valutare le sole performance del processo riguardante il nostro sistema e non il consumo globale di CPU e memoria da parte di tutti i processi in esecuzione, abbiamo dovuto utilizzare un'applicazione che ci permettesse di monitorare un singolo processo. Per fare ciò abbiamo deciso di utilizzare il *tool* SysStat [42]. Questo strumento permette di monitorare l'utilizzo della CPU e l'occupazione di memoria da parte di uno o più processi in esecuzione. SysStat permette sia di visualizzare in tempo reale questi dati, sia di salvare su un file i campioni raccolti durante la misura. La frequenza alla quale monitorare un processo è un parametro che l'utente può selezionare al momento dell'avvio di SysStat.

Durante l'esecuzione delle applicazioni di esempio abbiamo lanciato SysStat in modo che registrasse su un file i campioni relativi all'utilizzo della CPU e della memoria per tutta la durata dell'esperimento.

Invece, per quanto riguarda i test per valutare la scalabilità del sistema, man mano che aumentavamo il carico di lavoro, abbiamo monitorato per un minuto l'utilizzo della CPU e l'occupazione della memoria salvando su un file i campioni raccolti.

6.2 Esperimenti

In questa sezione verranno prima illustrate due applicazioni di esempio che abbiamo realizzato per valutare le funzionalità che IDrOS mette a disposizione ed il carico di lavoro richiesto all'hardware per poterle eseguire. Queste due applicazioni sono state eseguite effettuando dei voli reali sul prototipo da noi realizzato e presentato alla Sezione 5.3. In Figura 6.4 sono mostrate delle immagini scattate durante l'esecuzione dei voli.

Successivamente verranno presentati alcuni test che abbiamo eseguito per valutare la scalabilità del sistema. In questo caso le analisi sono state effettuate collegando IDrOS al simulatore; infatti, l'esecuzione di questi test non sarebbe stata sicura sull'APR perché in caso di errore non sarebbe stato possibile prevedere il comportamento del velivolo.

Per ogni esperimento che abbiamo effettuato siamo andati a misurare l'utilizzo della CPU e della memoria da parte di IDrOS ed abbiamo analizzato i dati raccolti.

Nelle applicazioni di esempio abbiamo confrontato le risorse hardware richieste dal nostro sistema con quelle richieste da MAVProxy (Sezione 5.2.1) ottenendo dei risultati migliori.

La scelta di utilizzare MAVProxy per confrontare i dati ottenuti durante l'esecuzione dei voli deriva dal fatto che essa è un applicazione che, in combinazione con le API DroneKit, offre la possibilità di creare dei programmi per pilotare il velivolo; tuttavia, MAVProxy e DroneKit offrono meno funzionalità rispetto ad IDrOS in quanto non permettono l'interfacciamento tramite Internet e non semplificano la gestione dei sensori. Inoltre, MAVProxy - oltre ad IDrOS - è l'unica applicazione compatibile con ArduCopter che è possibile installare a bordo degli APR per gestire il volo autonomo; anche per questo motivo è scelta come metro di paragone per valutare i nostri risultati.

(a) Prototipo da noi realizzato durante l'esecuzione di una missione

(b) Immagine acquisita dal nostro prototipo durante una missione

Figura 6.4: Immagini dei voli reali effettuati per valutare il funzionamento e le performance di IDrOS

6.2.1 Applicazione di Ricerca

Questa prima applicazione è un esempio concreto di una missione di ricerca, in particolare abbiamo voluto testare il comportamento di IDrOS nello scenario di ricerca dei dispersi sotto le valanghe.

Come già detto nel capitolo riguardante lo stato dell'arte, in particolare nella Sezione 2.2.2.1, per la ricerca di persone travolte da valanghe si utilizza un apparecchio detto ARTVA; il funzionamento di questo apparecchio è molto semplice: esso

è un ricetrasmittente radio che per localizzare la presenza di una vittima valuta la potenza di un segnale emesso da un dispositivo analogo che gli sciatori portano con sè. Localizzare una vittima corrisponde a trovare la posizione dove il segnale emesso dal dispositivo che lo sciatore porta con sè è massimo.

Per effettuare il nostro esperimento, dato che non avevamo a disposizione un ricevitore ARTVA da installare a bordo dell'APR, abbiamo deciso di simulare il comportamento di questo dispositivo creando un driver da installare su IDrOS che emulasse la presenza del ricevitore ARTVA.

Il funzionamento del driver per la simulazione del sensore ARTVA utilizza il sensore GPS per rilevare la posizione attuale del velivolo e calcola la distanza da un punto fisso che rappresenta la posizione della vittima. Una volta trovata la distanza tra APR e vittima il driver restituisce un valore che rappresenta il segnale ARTVA; tale valore assume valori decrescenti all'aumentare dalla distanza dal disperso.

Il modulo di navigazione utilizzato per svolgere questa applicazione riceve in ingresso i parametri rappresentanti l'area da perlustrare, successivamente divide tale area in una griglia e pilota l'APR in modo da farli raggiungere ogni cella della griglia. Per ottimizzare il processo di ricerca tale modulo si interfaccia con il sensore ARTVA per valutare se la vittima è stata trovata o meno: nel caso in cui la vittima viene localizzata il modulo di navigazione interrompe la perlustrazione e pilota il velivolo facendolo atterrare nel punto in cui è decollato. Se la vittima non viene localizzata e tutta l'area è stata perlustrata, il velivolo viene fatto atterrare.

Il modulo di campionamento ed elaborazione per valutare il punto in cui la vittima è situata è cosi costituito: la condizione da valutare per acquisire i dati dal sensore ARTVA ha sempre esito positivo in modo da acquisire costantemente i valori del sensore, i campioni che vengono acquisiti contengono il valore del segnale ARTVA e la posizione dove il dato è stato rilevato; la procedura di elaborazione una volta che la navigazione è conclusa deve valutare tutti i dati raccolti, fino al punto in cui la vittima è stata localizzata, per trovare il punto dove il segnale è massimo e restituire tale valore all'utente.

Questa applicazione è stata testata utilizzando il binding CoAP: per eseguire la missione è stata fatta una richiesta GET alla risorsa per la gestione delle missioni con il sensore ARTVA specificando come parametri il modulo di navigazione ed il modulo di campionamento creati per l'esecuzione di questa missione.

La missione è stata ripetuta più volte e durante i vari voli abbiamo registrato i dati dell'utilizzo della CPU e di occupazione della memoria; di seguito vengono illustrati ed analizzati i dati che abbiamo raccolto.

Utilizzo CPU: Durante l'esecuzione delle varie missioni abbiamo monitorato costantemente l'utilizzo della CPU dei processi riguardanti il binding CoAP ed IDrOS-

CORE (Figura 4.1). Le misure sono state effettuate utilizzando il tool SysStat: dopo aver mandato in esecuzione IDrOS-CORE ed il binding CoAP abbiamo lanciato SysStat facendogli salvare su un file i dati riguardanti l'utilizzo della CPU dei due processi sotto analisi.

Per quanto riguarda il binding CoAP quello che abbiamo ottenuto è un utilizzo medio del 0,89% del processore; tale valore rimane costante sia durante l'esecuzione della missione sia durante le fasi in cui il sistema è in attesa di istruzioni ed il velivolo è a terra pronto a decollare. Il fatto che questo valore rimanga costante, ovvero che non ci siano delle sostanziali differenze durante l'esecuzione delle missioni, è del tutto normale perché il binding CoAP deve solamente processare la richiesta contenente i parametri per lanciare la missione e ritornare i risultati a fine missione: tali operazioni non impattano sulle prestazioni del sistema perché avvengono solo all'inizio e alla fine missione.

Diversi sono invece i dati relativi all'utilizzo del processore da parte del processo IDrOS-CORE; i dati raccolti durante l'esecuzione dei vari esperimenti sono mostrati in Figura 6.5 (area blu): come si può osservare dal grafico sono presenti tre intervalli temporali in cui l'utilizzo della CPU è di circa il 5% superiore rispetto al normale. Questi intervalli sono i momenti in cui il drone era in volo per eseguire la missione. Durante l'esecuzione della missione il carico di lavoro richiesto è superiore per due motivi: il primo è perché IDrOS deve inviare i comandi di volo all'APR; il secondo è perché il sistema deve acquisire continuamente i dati dal sensore ARTVA utilizzato durante la missione. Inoltre, dal grafico si può notare che al termine di ogni missione è presente un picco di utilizzo della CPU attorno al 30%, tale valore è giustificato dal fatto che una volta terminata la missione il sistema deve processare i dati raccolti per trovare il punto in cui è stata rilevata la vittima.

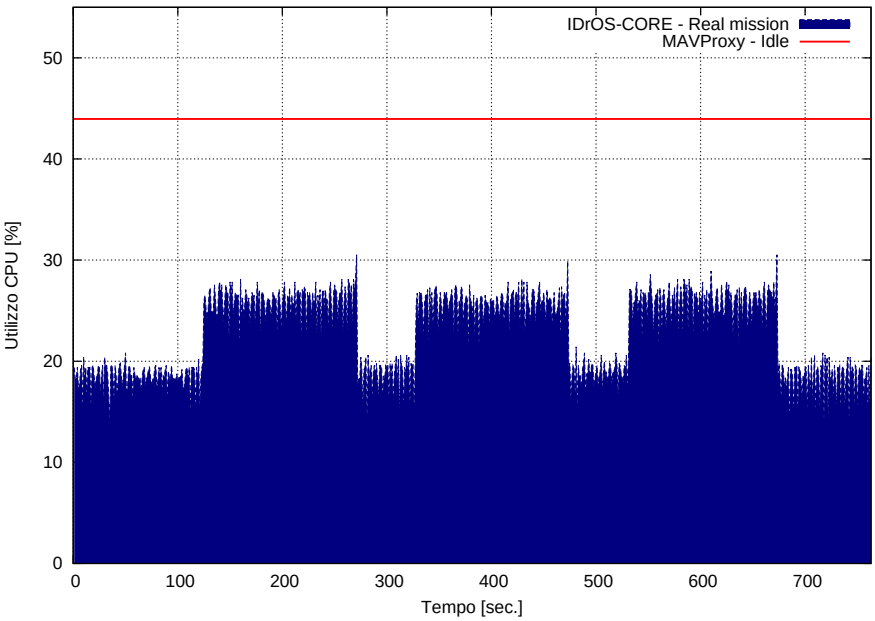


Figura 6.5: Utilizzo del processore nelle missioni di ricerca dispersi sotto le valanghe dal processo IDrOS-CORE

Nella Tabella 6.1 sono riportati i valori medi dell'utilizzo del processore durante l'esecuzione delle tre missioni da parte del processo IDrOS-CORE, essi sono in relazione con il grafico in Figura 6.5 in quanto rappresentano i valori medi dei tre intervalli temporali dove l'utilizzo della CPU è più alto. L'utilizzo della CPU medio, sempre da parte del processo IDrOS-CORE, mentre il velivolo non è in volo è del 18,58%. Come detto precedentemente l'utilizzo del processore da parte del binding CoAP è rimasto costante durante l'esecuzione di tutto l'esperimento ed il valore medio registrato è stato dello 0,89%.

Tabella 6.1: Utilizzo CPU durante le missioni di ricerca dispersi sotto le valanghe

Missione numero	IDrOS-CORE - Utilizzo medio CPU
1	25,85
2	25,69
3	25,97
Media	25,84

Dato che non esistono delle applicazioni simili ad IDrOS non è stato possibile fare dei confronti con altri sistemi che offrono le stesse funzionalità; tuttavia abbiamo provato ad installare MAVProxy sulla scheda BeagleBone in modo da utilizzare le

API DroneKit per pilotare il velivolo (Sezione 5.2.1), ottenendo un utilizzo medio della CPU del 43,96% mentre il drone era a terra in attesa di volare.

Questo valore è di gran lunga superiore alla somma delle misure registrate da IDrOS-CORE (25,84%) e dal binding CoAP (0,89%) durante l'esecuzione della missione ed è la conferma che la scelta, descritta nel Capitolo 5, di realizzare da zero delle API da utilizzare nel livello di astrazione hardware per pilotare il velivolo è stata corretta. In Figura 6.5 sono messi a confronto i dati relativi al consumo di CPU da parte di IDrOS-CORE e di MAVProxy.

Bisogna osservare che IDrOS è un sistema che offre molte più funzionalità rispetto a MAVProxy e alle API DroneKit, queste infatti possono essere comparate al solo livello di astrazione hardware che compone IDrOS. La nostra piattaforma rispetto a MAVProxy semplifica la gestione dei sensori, permette all'utente di caricare - in qualsiasi momento - i moduli per la gestione della missione ed offre delle interfacce Internet per controllare il velivolo. Nonostante ciò, il consumo di risorse da parte di IDrOS è minore rispetto a MAVProxy, il quale, attraverso le API DroneKit fornisce solamente delle funzioni ad alto livello per creare dei programmi in grado di pilotare il velivolo.

Memoria: Anche per effettuare l'analisi dell'occupazione di memoria da parte dei due processi (binding CoAP e IDrOS-CORE), abbiamo utilizzato il tool SysStat. Come nel caso precedente i dati relativi all'utilizzo di memoria sono stati registrati sia negli istanti temporali in cui il drone era fermo ed aspettava le istruzioni da eseguire, sia durante il volo ovvero durante l'esecuzione della missione.

L'utilizzo della memoria è stato costante durante l'esecuzione degli esperimenti, non sono state registrate sostanziali differenze tra gli istanti in cui il drone era a terra in attesa di ricevere istruzioni ed il periodo in cui il velivolo era in volo per eseguire la missione di ricerca.

I dati raccolti durante la missione di ricerca di dispersi sotto le valanghe sono riportati nella Tabella 6.2; mentre l'occupazione massima di memoria registrata durante i periodi in cui il sistema è in attesa di istruzioni è stata di 16,680MB per il binding CoAP e di 15,356MB per IDrOS.

Tabella 6.2: Occupazione memoria durante le missioni di ricerca dispersi sotto le valanghe

Missione numero	Binding CoAP - Utilizzo max memoria [MB]	IDrOS-CORE - Utilizzo max memoria [MB]
1	16,716	16,268
2	16,732	16,420
3	16,728	16,416
Media	16,725	16,368

Come per l'utilizzo della CPU, abbiamo confrontato IDrOS con la soluzione composta da MAVProxy ed API DroneKit ottenendo anche in questo caso dei risultati positivi: l'utilizzo di memoria da parte di MAVProxy è rimasto constante per tutto il tempo della misura al valore di 34,980MB. Tale valore è superiore alla somma delle medie dei valori massimi registrati da IDrOS-CORE (16,725MB) e dal binding CoAP (16,368MB) durante l'esecuzione della missione.

6.2.2 Applicazione di Campionamento

La seconda applicazione che abbiamo realizzato per l'analisi delle prestazioni di IDrOS è stata un'applicazione di videosorveglianza. Lo scopo della missione è quello di far volare il drone lungo un perimetro ed al raggiungimento di alcuni punti specificati scattare delle fotografie ed inviarle all'utente.

Come anticipato nella Sezione 5.3.1, durante l'esecuzione di questa missione non abbiamo utilizzato la board BeagleBone Black, ma abbiamo installato a bordo dell'APR la scheda Raspberry Pi b+ con il modulo per l'acquisizione di immagini. Questa scelta è stata fatta per la facilità con cui è possibile acquisire le immagini utilizzando il modulo telecamera sviluppato per il Raspberry Pi e per le ridotte dimensioni di questa camera digitale.

Per l'esecuzione della missione abbiamo dovuto caricare a bordo del sistema il driver per permettere ad IDrOS di utilizzare il modulo per l'acquisizione delle immagini (Listato 5.1), inoltre abbiamo sviluppato un modulo di navigazione per permettere la navigazione tramite *waypoints*. Per questa missione l'utente deve specificare i *waypoints* da fare raggiungere all'APR per seguire il percorso desiderato.

Per quanto riguarda il modulo di campionamento ed elaborazione, esso è così composto: la condizione da valutare per l'acquisizione di un'immagine deve valutare se l'APR è in una delle posizioni indicate dall'utente per scattare una fotografia; la procedura di elaborazione non deve svolgere nessuna operazione sui dati raccolti ma deve semplicemente restituire tutte le fotografie scattate.

Questa applicazione è stata testata utilizzando il binding MQTT. Per lanciare le missioni ed analizzare i risultati è stato utilizzato Node-RED; in particolare, sono stati realizzati i flussi mostrati in Figura 6.6. Il primo flusso è quello per inviare i comandi al drone per lanciare la missione, esso è identico a quello descritto nella Sezione 6.1.3. Il secondo flusso invece è così costituito: il nodo *Mqtt Subscribe* serve per informare il broker MQTT che è interessato a ricevere i messaggi restituiti dalla missione; il nodo *switch* serve per valutare se la risposta contiene un errore o meno, in caso affermativo viene stampato a video l'errore, in caso negativo il flusso prosegue; il nodo *Get Samples* da un messaggio contenente la lista di foto acquisite produce tanti messaggi contenenti una singola foto; il nodo *Create File* riceve le foto e genera

le informazioni necessarie per il salvataggio delle immagini (*path* e *filename*); infine il nodo *Save File* salva le foto sul file system.

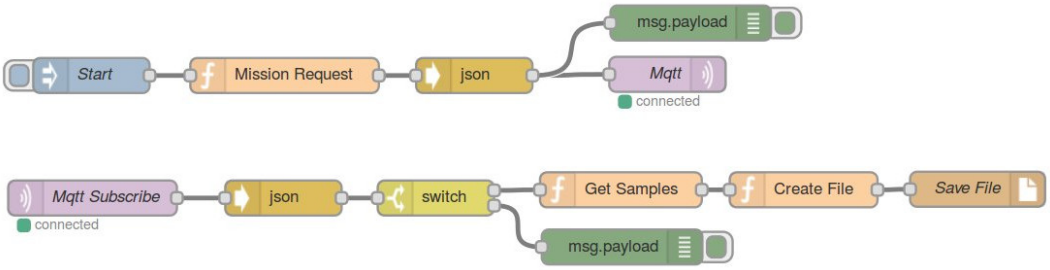


Figura 6.6: Applicazione realizzata con Node-RED per interagire con l'APR utilizzando il binding MQTT

Come per l'applicazione di ricerca dei dispersi sotto le vanghe, abbiamo mandato in esecuzione questa missione più volte ed abbiamo monitorato per tutto il tempo dell'esperimento i valori di utilizzo della CPU e dell'occupazione di memoria. Dato che per questa missione abbiamo utilizzato una piattaforma hardware diversa da quella utilizzata nell'applicazione mostrata precedentemente, i dati raccolti dai due esperimenti non sono comparabili tra di loro. Di seguito sono riportati i dati che abbiamo raccolto.

Utilizzo CPU: Come per il caso precedente, il processo che utilizza la maggior parte delle risorse della CPU è IDrOS-CORE; in Figura 6.7 (area blu) è mostrato il grafico dell'utilizzo del processore da parte di IDrOS-CORE per tutto il tempo dell'esecuzione dell'esperimento. Come si può osservare, analogamente al caso precedente, sono presenti tre intervalli temporali in cui l'utilizzo del processore è di qualche punto percentuale superiore rispetto alla norma; questi intervalli sono i periodi di tempo in cui la missione era in esecuzione ed il drone era in volo. Però, rispetto all'applicazione di ricerca, durante l'esecuzione della missione la differenza di utilizzo della CPU non è così marcata perché, in questo caso, la missione non prevedeva di acquisire continuamente i dati dalla camera digitale ma l'acquisizione è stata fatta solo quando l'aeromobile raggiungeva dei punti precisi. Al termine della missione c'è un picco in cui l'utilizzo della CPU tocca il 25%: esso è dovuto all'invio delle immagini raccolte.

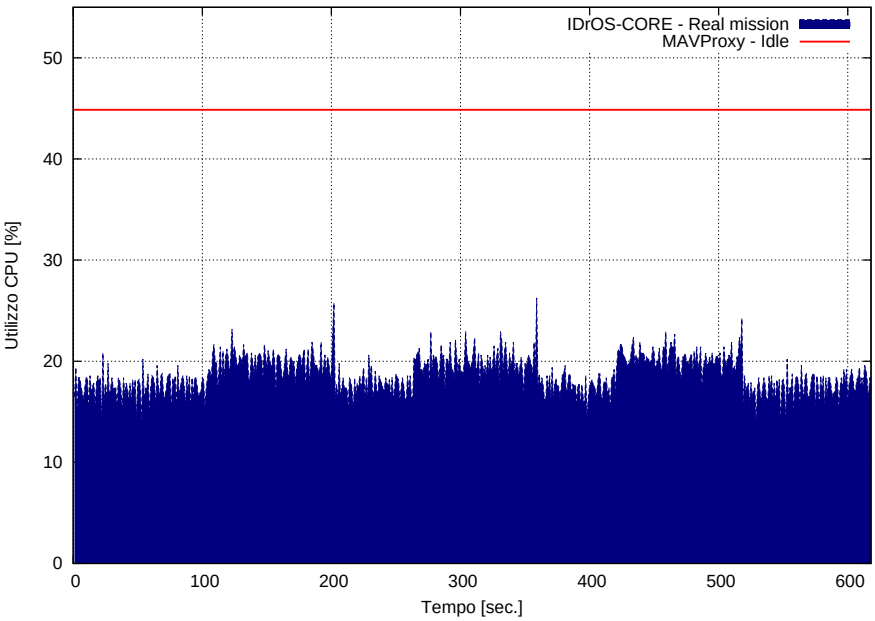


Figura 6.7: Utilizzo del processore nelle missioni di videosorveglianza dal processo IDrOS-CORE

In Tabella 6.3 è illustrato l'utilizzo della CPU durante l'esecuzione delle tre missioni da parte del processo IDrOS-CORE, i dati riportati sono in relazione con il grafico in Figura 6.7 in quanto rappresentano i valori medi dei tre instanti dove l'utilizzo della CPU è più alto. Per quanto riguarda l'utilizzo della CPU da parte del binding MQTT esso assume un valore costante durante tutta la durata dell'esperimento: il valore medio misurato indica un utilizzo del 0,46%.

Tabella 6.3: Utilizzo CPU durante le missioni di videosorveglianza

Missione numero	IDrOS-CORE - Utilizzo medio CPU
1	20,01
2	19,99
3	20,34
Media	20,11

Anche per questo esperimento abbiamo provato ad installare MAVProxy e le API DroneKit sul Raspberry Pi per valutare il carico di lavoro richiesto e confrontarlo con quello necessario per eseguire IDrOS. I dati che abbiamo ottenuto sono ancora una volta positivi: MAVProxy utilizza mediamente il 44,86% della CPU, valore nettamente superiore rispetto alla somma di quelli richiesti da IDrOS-CORE (20,11%) e dal binding MQTT (0,46%).

Come nel caso precedente, la misura registrata utilizzando MAVProxy è stata effettuata quando il velivolo non era in volo, mentre la misura effettuata con IDrOS è stata fatta quando il drone era in volo. Ciò risalta maggiormente i risultati ottenuti nell'applicazione precedente ovvero che IDrOS, offrendo più funzionalità rispetto a MAVProxy, riesce comunque a consumare meno risorse hardware. In Figura 6.7 sono confrontati i dati registrati da IDrOS-CORE durante l'esecuzione della missione e quelli medi registrati mentre MAVProxy era in esecuzione in stato *idle*.

Inoltre, dato che abbiamo ottenuto risultati positivi utilizzando due schede diverse, possiamo affermare che non è casuale il fatto che IDrOS richieda meno risorse hardware rispetto a MAVProxy per poter operare.

Memoria: L'occupazione di memoria, come per il caso precedente, è rimasta costante per tutta l'esecuzione degli esperimenti: non si sono riscontrate differenze tra gli istanti in cui il drone era in volo per eseguire la missione e gli istanti in cui il velivolo era a terra in attesta di eseguire nuove missioni. In Tabella 6.4 sono riportati i dati relativi all'utilizzo massimo di memoria durante l'esecuzione delle tre missioni di videosorveglianza.

Tabella 6.4: Occupazione memoria durante le missioni di videosorveglianza

Missione numero	Binding MQTT - Utilizzo max memoria [MB]	IDrOS-CORE - Utilizzo max memoria [MB]
1	9,344	20,928
2	9,336	21,256
3	9,340	21,208
Media	9,340	21,131

L'utilizzo di memoria massimo mentre il sistema non stava eseguendo alcuna missione è di 9,316MB per il binding MQTT e di 20,120MB per IDrOS-CORE.

Anche sul Raspberry Pi abbiamo valutato l'occupazione di memoria nel caso in cui si utilizzino MAVProxy e le API DroneKit per interfacciarsi con il drone ottenendo un consumo di memoria medio di 33,65MB. Tale dato conferma ancora una volta che IDrOS richiede meno risorse hardware per poter operare nonostante fornisca all'utente più funzionalità.

6.2.3 Scalabilità

Presentiamo ora alcuni test che abbiamo effettuato per valutare come scala il sistema al variare del carico di lavoro richiesto; in particolare, sono stati fatti dei test per valutare come varia il consumo di risorse hardware richieste da IDrOS all'aumentare dei seguenti parametri: numero di sensori installati a bordo del sistema, frequenza di campionamento dei sensori, numero di utenti che accedono al velivolo.

Questi test sono stati fatti utilizzando la scheda BeagleBone Black come piattaforma hardware per eseguire IDrOS ed abbiamo utilizzato il simulatore SITL per emulare la presenza del velivolo.

Di seguito sono illustrati i vari esperimenti che abbiamo eseguito ed i relativi risultati.

6.2.3.1 Frequenza di Campionamento dei Sensori

Lo scopo di questo test è quello di valutare come varia l'utilizzo della CPU e della memoria al variare della frequenza di campionamento di un sensore. Per eseguire questa analisi abbiamo scelto di utilizzare il sensore GPS: in questo caso il sistema non dovrà andare a leggere fisicamente il sensore perché esso non è installato sul BeagleBone ma dovrà andare a leggere l'ultimo dato inviato dal velivolo, relativo a questo sensore, attraverso il collegamento alla board di controllo tramite il protocollo MAVLink.

Le varie misure sono state eseguite variando la frequenza di campionamento del sensore utilizzato durante queste analisi modificando il file di configurazione di IDrOS. Per ogni valore di frequenza utilizzato abbiamo monitorato per un minuto il consumo di memoria e l'utilizzo di CPU attraverso il tool SysStat; questa analisi è stata fatta due volte: la prima utilizzando il binding CoAP, la seconda utilizzando il binding MQTT.

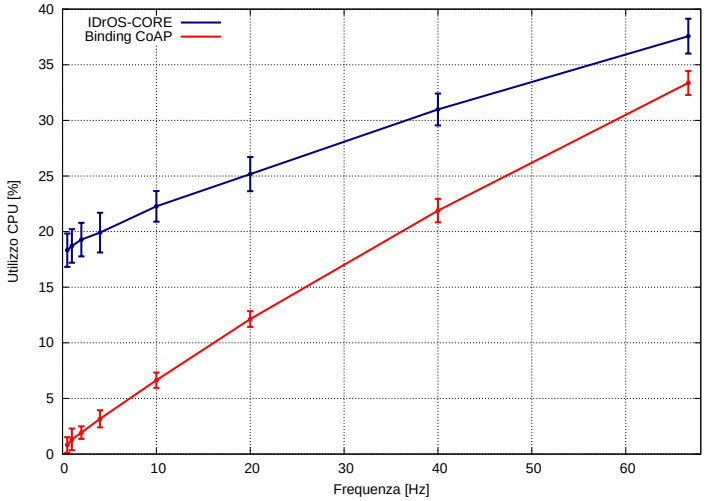
Al termine dell'esperimento, per ogni valore di frequenza analizzato abbiamo calcolato la media e la deviazione standard dei campioni raccolti. In Figura 6.8a sono illustrati i dati che abbiamo raccolto relativi all'utilizzo della CPU nella configurazione di IDrOS con il binding CoAP, in Figura 6.8b sono rappresentati gli stessi dati nella configurazione in cui IDrOS è stato utilizzato con il binding MQTT.

Come si può notare l'utilizzo della CPU segue un trend lineare all'aumentare della frequenza di campionamento. Il valore massimo con cui il sistema può campionare il sensore da noi testato, prima che IDrOS-CORE, il binding ed il sistema operativo utilizzino il 100% del processore, è di 66Hz se si utilizza il binding CoAP e di 40Hz se si utilizza MQTT. I valori massimi registrati sono differenti tra i due binding perché nel caso in cui si utilizza il binding MQTT è necessario che sul sistema sia operante un broker per gestire i messaggi scambiati dai vari nodi MQTT.

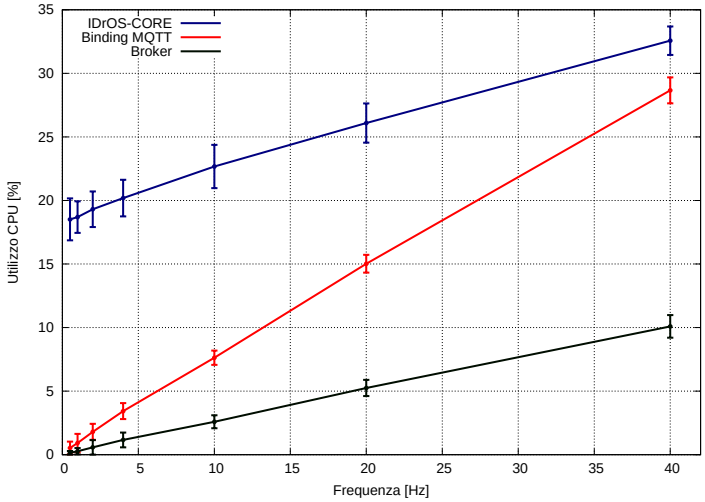
Bisogna osservare che tali frequenze sono più che sufficienti per campionare la maggioranza di sensori che si possono utilizzare a bordo degli APR per svolgere le missioni, pertanto tali valori costituiscono un buon risultato.

Dai grafici in Figura 6.8 possiamo notare che il processo IDrOS-CORE in entrambe le configurazioni (CoAP e MQTT) fa registrare gli stessi dati per gli stessi valori di frequenza , questo aspetto è del tutto normale in quanto il suo comportamento non dipende dal binding utilizzato. Quello che emerge dai grafici è che il coefficiente di crescita dei due binding è maggiore rispetto a quello di IDrOS-CORE; questo indica

che la parte del sistema che incide maggiormente sulla CPU è l'invio dei messaggi attraverso l'interfaccia di rete: il binding CoAP deve interfacciarsi con IDrOS-CORE attraverso l'interfaccia di rete in *loopback* e deve comunicare con l'utente mediante il protocollo CoAP attraverso l'interfaccia Wifi; il binding MQTT deve utilizzare l'interfaccia in loopback per collegarsi con IDrOS e inviare tali messaggi al broker utilizzando un'altra interfaccia di rete sempre in *loopback*.



(a) Utilizzo medio della CPU nella configurazione con il binding CoAP



(b) Utilizzo medio della CPU nella configurazione con il binding MQTT

Figura 6.8: Utilizzo medio della CPU al variare della frequenza di campionamento dei sensori

Per quanto riguarda l'occupazione della memoria, il valore è rimasto costante al variare della frequenza di campionamento; i valori massimi registrati durante l'esecuzione dei test sono mostrati in Tabella 6.5. Dato che il valore di utilizzo di memoria rimane costante, possiamo concludere dicendo che la frequenza di campionamento dei sensori non incide sull'occupazione di memoria ma solamente sull'utilizzo della CPU.

Tabella 6.5: Occupazione massima di memoria registrata durante gli esperimenti in cui è stata fatta variare la frequenza di campionamento dei sensori

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,58
Binding CoAP	16,42

(a) Caso CoAP

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,49
Binding MQTT	5,66
Broker	0,77

(b) Caso MQTT

6.2.3.2 Numero di Sensori

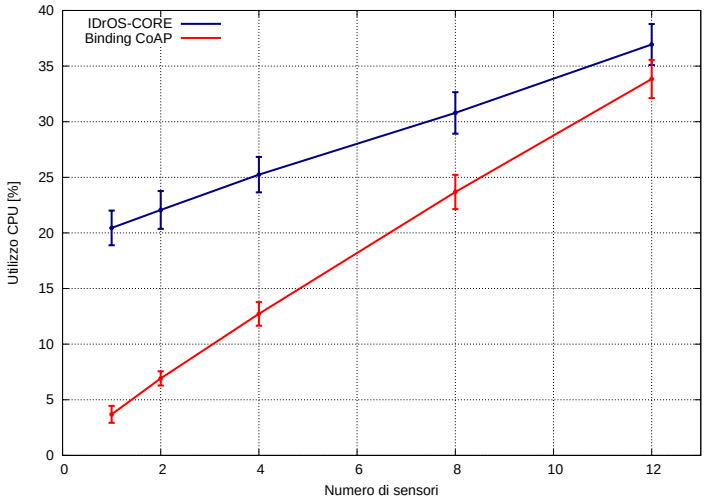
Lo scopo di questa analisi è quello di valutare come scala il sistema al variare del numero di sensori installati a bordo dell'APR, in particolare abbiamo testato le performance di IDrOS variando il numero di sensori GPS. Questa analisi è stata fatta perché il processing per campionare continuamente più sensori è diverso rispetto a quello per campionare un singolo sensore con frequenze diverse in quanto sono attivi più *thread*, uno per monitorare ogni sensore.

Dato che il drone dispone di un solo sensore GPS abbiamo creato tanti driver che leggessero i valori provenienti dal GPS così da emulare la presenza di tanti sensori dello stesso tipo. Come per il caso precedente i driver utilizzati per leggere il GPS devono accedere all'ultimo dato - relativo al sensore GPS - inviato dalla board di controllo tramite il protocollo MAVLink ad IDrOS.

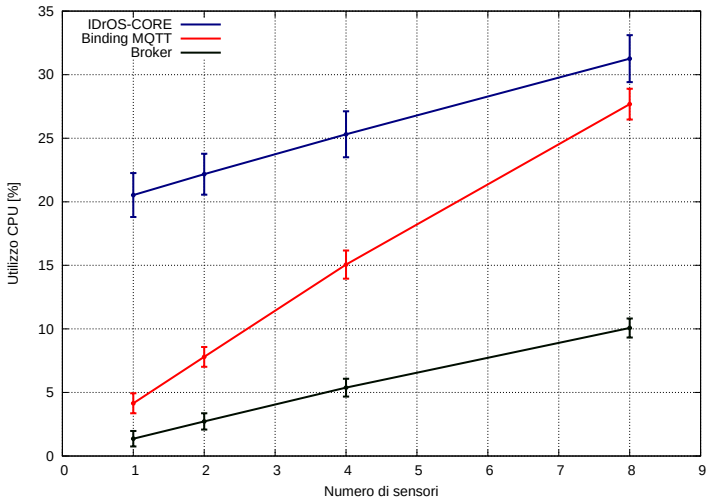
A differenza del test in cui abbiamo variato la frequenza di campionamento, in questo caso essa non deve variare tra una misura e l'altra perché quello che varia è il numero di sensori campionati. Per le nostre misure abbiamo deciso di fissare la frequenza di campionamento dei sensori alla frequenza di default di 5Hz; in un'applicazione reale non sarebbe necessario campionare il sensore GPS con una frequenza così elevata tuttavia, dato che il nostro scopo è quello di valutare i limiti del sistema, tale valore è significativo perché ci fornirà una stima pessimistica delle capacità reali del sistema.

Per variare il numero di sensori da campionare tra una misura e l'altra abbiamo dovuto modificare il file di configurazione di IDrOS specificando quali sensori andavano utilizzati.

I dati raccolti relativi all'utilizzo della CPU sono mostrati nella Figura 6.9: come nel caso precedente si può osservare che il consumo di CPU varia linearmente all'aumentare del numero di sensori campionati e che i processi riguardanti i due binding hanno un coefficiente di crescita più elevato rispetto al processo IDrOS-CORE. Inoltre, utilizzando MQTT come binding protocollare il limite massimo di sensori che si possono campionare contemporaneamente, prima di saturare il processore, è inferiore rispetto al numero di quelli che si possono monitorare utilizzando CoAP. Il limite massimo di 12 sensori per CoAP e 8 per MQTT è comunque un buon risultato perché nei casi reali è difficile trovare esempi di applicazioni che richiedono di utilizzare più di 5/10 sensori per svolgere una missione. Inoltre, come visto nei test precedenti, la frequenza di campionamento incide sull'utilizzo della CPU: quindi, diminuendo la frequenza di campionamento è possibile utilizzare contemporaneamente più sensori rispetto ai limiti trovati da questo esperimento.



(a) Utilizzo medio della CPU nella configurazione con il binding CoAP



(b) Utilizzo medio della CPU nella configurazione con il binding MQTT

Figura 6.9: Utilizzo medio della CPU al variare del numero di sensori installati

Anche in questo caso il consumo di memoria è rimasto costante al variare del numero di sensori installati a bordo: i valori massimi registrati durante il nostro esperimento sono mostrati nella Tabella 6.6.

Tabella 6.6: Occupazione massima di memoria registrata durante gli esperimenti in cui è variato il numero di sensori installati

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,8
Binding CoAP	17,1

(a) Caso CoAP

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,54
Binding MQTT	5,67
Broker	0,78

(b) Caso MQTT

6.2.3.3 Numero di Client

L'ultima analisi che abbiamo eseguito ha lo scopo di valutare quanti utenti può gestire contemporaneamente il sistema che abbiamo sviluppato.

Per effettuare quest'analisi abbiamo mandato in esecuzioni su un laptop un numero variabile di client, i quali ogni 2 secondi effettuavano una richiesta verso un sensore installato a bordo dell'APR. La richiesta non è stata fatta al componente software *gestore della missione* (Figura 4.1) perché, come illustrato al Capitolo 5, IDrOS permette l'esecuzione di una sola missione alla volta, quindi, non sarebbe stato possibile valutare le performance del sistema con più utenti collegati.

Quest'analisi è stata effettuata facendo in modo che le richieste provenienti dai client fossero distribuite uniformemente nel tempo così da non avere degli istanti in cui il sistema dovesse gestire tutte le richieste e degli istanti in cui il sistema non dovesse gestire alcuna richiesta.

L'intervallo di 2 secondi tra una richiesta e l'altra è un valore abbastanza basso se consideriamo l'utilizzo del sistema in uno scenario reale perché, generalmente, una missione dura qualche minuto; la scelta del valore di 2 secondi è stata dettata dal fatto che questo è un valore limite il quale ci ha permesso di avere una valutazione pessimistica delle capacità del sistema.

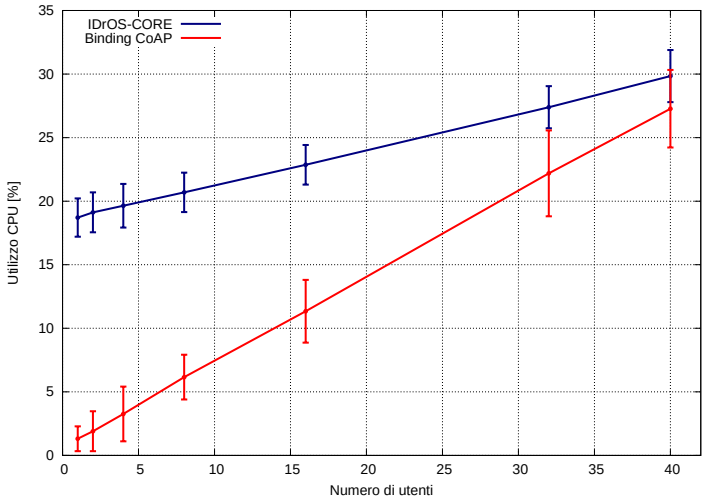
Come per gli esempi illustrati precedentemente, ogni volta che abbiamo variato il numero di utenti abbiamo monitorato per un minuto l'utilizzo di CPU e memoria. Al termine degli esperimenti è stata calcolata la media e la deviazione standard dei campioni raccolti.

I dati raccolti durante questa missione sono riportati nel grafico in Figura 6.10 e nella Tabella 6.7.

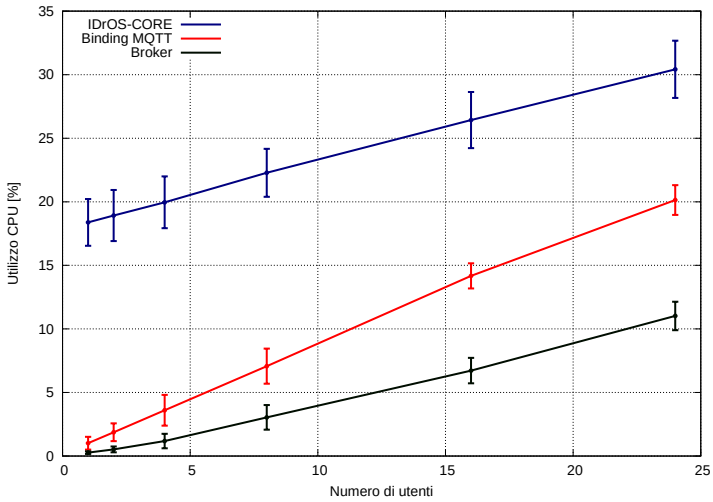
Il numero massimo di utenti che IDrOS può gestire, prima che la CPU sia satura, è di 40 se si utilizza il binding CoAP e di 24 se si utilizza il binding MQTT.

Tali valori possono essere considerati un risultato positivo perché è difficile trovare un'applicazione reale per la quale ci sono così tanti utenti collegati al sistema contemporaneamente.

Inoltre, anche in questo caso, l'utilizzo della CPU è aumentato linearmente al variare degli utilizzatori mentre l'occupazione di memoria da parte del sistema è rimasta costante al variare del numero di client connessi.



(a) Utilizzo medio della CPU nella configurazione con il binding CoAP



(b) Utilizzo medio della CPU nella configurazione con il binding MQTT

Figura 6.10: Utilizzo medio della CPU al variare del numero di client connessi ad IDrOS

Tabella 6.7: Occupazione massima di memoria registrata durante gli esperimenti per valutare il numero massimo di client supportati da IDrOS

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,54
Binding CoAP	17,32

(a) Caso CoAP

Processo	Utilizzo massimo memoria [MB]
IDrOS-CORE	15,61
Binding MQTT	5,77
Broker	0,78

(b) Caso MQTT

Conclusioni

In questa tesi abbiamo presentato IDrOS, la piattaforma da noi sviluppata per permettere agli APR di superare i limiti emersi dallo studio dello stato dell'arte, ovvero la possibilità di programmare i droni specificando solamente un lista di *waypoints* e la necessità di utilizzare la stazione di controllo per interfacciarsi con essi. Per fare ciò, abbiamo proposto un nuovo modello di programmazione per la gestione degli APR ed abbiamo sviluppato un'architettura software in grado di supportare tale modello e rendere possibile il controllo del velivolo tramite Internet.

Il modello di programmazione con cui è stato sviluppato IDrOS è pensato per permettere agli APR di utilizzare tecniche di *Active Sensing* per gestire la navigazione: ciò significa rendere i droni intelligenti ed autonomi nell'eseguire le missioni richieste permettendo ad essi di modificare il proprio piano di volo sulla base dei valori letti dai sensori installati a bordo. Inoltre, il nostro modello di programmazione è stato definito in maniera tale da essere versatile e al contempo semplice: infatti, con tale modello gli utenti possono programmare facilmente gli APR per svolgere missioni in scenari molto diversi tra loro.

L'architettura, oltre a supportare il modello di programmazione da noi definito, offre la possibilità di interfacciarsi con gli APR tramite Internet attraverso una serie di binding protocollari: in questo modo, i droni possono affacciarsi nel contesto dell'*Internet of Things*.

Grazie allo sviluppo di questi due contributi è possibile avere un velivolo autonomo, intelligente ed in grado di interfacciarsi con molti dispositivi e sistemi informatici aprendo così nuovi scenari applicativi per quanto riguarda il campionamento di dati da remoto.

Il nostro lavoro di tesi ha permesso anche di validare IDrOS sul campo: infatti, non ci si è limitati solamente alla progettazione ed allo sviluppo della piattaforma, ma si è anche realizzato un prototipo con il quale sono stati effettuati voli di test e simulazioni. La realizzazione di un prototipo funzionante è la dimostrazione che il sistema da noi realizzato è utilizzabile non solo in teoria ma anche in applicazioni reali. Inoltre, i risultati ottenuti dalle misure delle prestazioni sono stati largamente positivi poiché IDrOS, in esecuzione su un dispositivo embedded installato a bordo del velivolo, ha occupato meno del 30% delle risorse hardware - in termini di CPU

- ed ha scalato linearmente al variare del carico di lavoro richiesto, lasciando quindi ampio margine per la realizzazione di applicazioni più complesse rispetto a quelle da noi testate.

Nonostante IDrOS sia una piattaforma realmente utilizzabile sono possibili diversi sviluppi futuri.

In primo luogo, sarebbe interessante rendere IDrOS utilizzabile per pilotare tutti gli APR che si trovano sul mercato: infatti, un limite dell'implementazione attuale è quello di poter impiegare il nostro sistema solamente sugli aeromobili che usano come controllore di volo il software ArduCopter che, pur essendo compatibile con molti velivoli, non copre la totalità degli APR disponibili in commercio.

In secondo luogo, si potrebbe estendere il modello di programmazione da noi proposto per rendere possibile l'utilizzo di attuatori durante la missione. In tal modo il drone sarebbe capace di svolgere quelle applicazioni che richiedono di poter afferrare degli oggetti durante la missione: un esempio di quest'ultimo caso è l'utilizzo dei droni come corrieri [4]. Nel nostro modello non è stato tenuto in considerazione questo aspetto perché ci siamo concentrati prevalentemente sull'utilizzo dei droni per svolgere applicazioni basate sull'acquisizione di dati.

L'ultimo sviluppo che potrebbe riguardare il nostro sistema in futuro è quello di permettere il coordinamento tra più dispositivi che operano utilizzando IDrOS. Ciò è già possibile attraverso le interfacce basate sui protocolli Internet messe a disposizione; tuttavia, questo approccio è pensato per connettere dispositivi eterogenei quindi non avrebbe senso utilizzarlo per far comunicare più droni controllati da IDrOS perché causerebbe un overhead non necessario. Per risolvere questo inconveniente si potrebbe pensare ad una soluzione più efficiente per mettere in comunicazione, senza l'utilizzo delle interfacce Internet, più dispositivi gestiti tramite IDrOS.

Bibliografia

[1] 3DRobotics. 3dr u-blox gps with compass. https://store.3drobotics.com/products/3dr-gps-ublox-with-compass.

[2] AA.VV. *UAS: The Global Perspective*. Blyenburgh & Co, 2011.

[3] Adafruit. https://learn.adafruit.com/embedded-linux-board-comparison.

[4] Amazon Prime Air. http://www.amazon.com/b?node=8037720011.

[5] ArduCopter. http://copter.ardupilot.com/ - https://github.com/diydrones/ardupilot/tree/master/ArduCopter.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 2010.

[7] Reg Austin. *Unmanned Aircraft Systems*, pages 1–15. John Wiley & Sons, Ltd, 2010.

[8] Barnard Microsystems. Oil and gas pipeline monitoring. http://www.barnardmicrosystems.com/UAV/pipeline_monitoring/pipeline_monitoring.html.

[9] Beagle Board. BeagleBone Black. http://beagleboard.org/black.

[10] H. Bendea, Piero Boccardo, S. Dequal, Fabio Giulio Tonolo, D. Marenchino, and M. Piras. Low cost uav for post-disaster assessment. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2008.

[11] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 2013.

[12] Marco Buschmann, Jens Bange, and Peter Vörsmann. Mmav a miniature unmanned aerial vehicle (mini-uav) for meteorological purposes. *Technische Universitat Braunschweig*, 2004.

[13] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College, Computer Science, 2000.

[14] Cranes and Drones: Strange Airfellows? https://www.fort.usgs.gov/science-feature/127.

[15] DroneKit. http://python.dronekit.io/ - https://github.com/dronekit/dronekit-python.

[16] DS18B20. https://www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS18B20.html.

[17] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[18] Gartner Inc. Gartner's 2015 hype cycle for emerging technologies identifies the computing innovations that organizations should monitor. http://www.gartner.com/newsroom/id/3114217, 2015.

[19] Rahul Gupta and Andrew Banks. *MQTT Version 3.1.1.* OASIS Standard, 2014.

[20] Patrick Y. Haas, Christophe Balistreri, Piero Pontelandolfo, Gilles Triscone, Hasret Pekoz, and Antonio Pignatiello. Development of an unmanned aerial vehicle uav for air quality measurement in urban areas. *American Institute of Aeronautics and Astronautics*, 2014.

[21] K. Hartke. Observing resources in coap. 2014.

[22] Matthias Kovatsch. Californium (cf). http://www.eclipse.org/californium/ - https://github.com/eclipse/californium.

[23] Matthias Kovatsch. Cooper (cu). https://github.com/mkovatsc/Copper.

[24] Matthias Kovatsch, Simon Mayer, and Benedikt Ostermaier. Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012.

[25] LM35. http://www.ti.com/product/lm35.

[26] MAVLink. http://qgroundcontrol.org/mavlink/start - https://github.com/mavlink/mavlink.

[27] Maxbotix. http://www.maxbotix.com/.

[28] Hugh McDaid, David Oliver, Admiral Barton Strong, and General Ken Israel. Remote piloted aerial vehicles : An anthology. 2003.

[29] Mosquitto. http://mosquitto.org/.

[30] S. Nebikera, A. Annena, M. Scherrerb, and D. Oeschc. A light-weight multi-spectral sensor for micro uav - opportunities for very high resolution airborne remote sensing. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2008.

[31] Francesco Nex and Fabio Remondino. Uav for 3d mapping applications: a review. *Applied Geomatics*, 2014.

[32] Node-RED. `http://nodered.org/`.

[33] Lian Pin Koh and Serge A Wich. Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation. *Tropical Conservation Science*, 2012.

[34] Pixhawk. `https://pixhawk.org`.

[35] pymavlink. `https://github.com/mavlink/mavlink/tree/master/pymavlink`.

[36] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[37] Raspberry Pi. Raspberry Pi 1 Model B+. `https://www.raspberrypi.org/products/model-b-plus/`.

[38] Fulvio Rinaudo, Filiberto Chiabrando, Andrea Maria Lingua, and A.T. Spanò. Archaeological site monitoring: Uav photogrammetry can be an answer. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2012.

[39] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). *RFC 7252*, 2014.

[40] Sherpa. `http://www.sherpa-project.eu/sherpa/`.

[41] T. Spiess, J. Bange, M. Buschmann, and P. Vörsmann. First application of the meteorological Mini-UAV 'M^{2}AV'. *Meteorologische Zeitschrift*, 2007.

[42] SysStat. `http://sebastien.godard.pagesperso-orange.fr/`.

[43] Andrew Tridgell. MAVProxy. `http://dronecode.github.io/MAVProxy/html/index.html`.