Dipartimento di Ingegneria dell'Informazione
Corso di laurea in
Ingegneria Elettronica

Relatore:
Bevilacqua Andrea

Laureanda:
Girardello Sofia

Anno Accademico 2022/2023
Data di laurea 17/07/2023

agri drone

Implementazione di un sistema di monitoraggio tramite drone per la gestione dell'agricoltura di precisione

agrid

lrone

Abstract

Il progetto di tesi propone lo sviluppo di un sistema di monitoraggio tramite drone per l'agricoltura di precisione, in risposta alla crisi demografica ed alimentare prevista nel settore nei prossimi anni. L'obiettivo principale del sistema è garantire la sicurezza e l'efficienza delle operazioni agricole attraverso l'utilizzo di tecnologie avanzate.

L'implementazione del sistema, avvenuta in due fasi distinte, prevede una prima progettazione di una struttura in grado di lavorare solo a livello locale ed in un secondo momento la distribuzione di questo sistema su di un'architettura distribuita. Qui, i dati raccolti dal drone vengono trasmessi ad un server locale e successivamente distribuiti nel Cloud in modo tale da renderli accessibili anche al di fuori dell'Intranet. Gli agricoltori possono interagire con il sistema attraverso un'interfaccia web o un'applicazione Android, consentendo loro di scegliere le porzioni di terreno da monitorare. Quando viene selezionata un'area, il drone si dirige verso la posizione scelta ed acquisisce una fotografia della destinazione desiderata. Questa immagine viene quindi visualizzata dall'utente finale attraverso le interfacce web o mobile. La disponibilità delle immagini consente agli agricoltori di monitorare l'andamento dei propri campi in tempo reale, identificando tempestivamente eventuali problemi o anomalie.

Il sistema offre prospettive di espansione future attraverso l'applicazione di algoritmi di visione artificiale o apprendimento automatico su di esso. Questo consentirà l'automazione di ulteriori processi nell'agricoltura di precisione, come l'identificazione di malattie delle piante, l'ottimizzazione dell'uso delle risorse e la previsione delle rese.

In conclusione, il progetto di tesi propone un sistema innovativo di monitoraggio tramite drone per l'agricoltura di precisione, che sfrutta le tecnologie emergenti per migliorare la sicurezza e l'efficienza delle operazioni agricole. La sua natura distribuita e la possibilità di integrazione con algoritmi di intelligenza artificiale lo rendono un'opzione promettente per l'industria agricola del futuro.

Indice

Elenco delle figure

Elenco dei codici

Introduzione

1.1 Concetti di agricoltura di precisione e droni

Nei prossimi anni l'agricoltura sarà chiamata a soddisfare un'enorme richiesta di alimenti. La popolazione mondiale raggiungerà infatti la quota di 9.7 miliardi di persone nel 2050 e, per questo motivo, la produzione di alimenti dovrà potenzialmente raddoppiare entro tale data.[1]

L'urgente bisogno di raddoppiare la produzione agricola nei prossimi 25 anni su di un terreno ridotto e con minor disponibilità di acqua comporterebbe inevitabilmente significativi costi sociali, economici ed ambientali. L'identificazione di strumenti in grado di minimizzare tali costi attraverso un aumento della produttività e dei profitti economici, contemporaneamente alla conservazione dell'ambiente, risulta pertanto cruciale. L'agricoltura di precisione, che ha attirato l'attenzione a livello mondiale fin dagli anni '90, è uno di questi strumenti.

L'agricoltura di precisione può essere definita come una strategia olistica e rispettosa dell'ambiente che consiste nell' applicazione di tecnologie, principi e strategie per una gestione intelligente della produzione agricola, in relazione alle reali necessità dell'appezzamento ed alla loro variabilità spaziale e temporale.[2] L'agricoltura di precisione può quindi essere intesa come una forma di agricoltura progredita, volta all'impiego di tecniche e tecnologie mirate all'applicazione variabile degli input colturali all'interno degli appezzamenti sulla base dell'effettiva esigenza della coltura e delle proprietà chimico-fisiche e biologiche del suolo, al fine di perseguire dei vantaggi di ordine agronomico, mediante l'accrescimento della performance della coltura; economico, attraverso la razionalizzazione degli ingressi[3] e la riduzione dei costi colturali; ambientale, attraverso l'uso ragionato degli input chimici e meccanici.[4]

Tra le tecnologie inserite nell'ambito dell'agricoltura di precisione, i droni hanno avuto un larghissimo impiego per gli straordinari risultati che apportano in termini di raccolta di immagini, informazioni e dati. Questa tipologia di velivoli, date le loro dimensioni ridotte, può raggiungere facilmente ogni porzione di territorio restituendo dati precisi, accurati ed affidabili. Pertanto, essi vengono utilizzati in diverse applicazioni legate all'agricoltura: grazie a rilevatori avanzati quali sensori

multispettrali o iperspettrali ed a telecamere RGB o termiche, i droni sono in grado di raccogliere informazioni dettagliate sullo stato delle coltivazioni, come la salute delle piante, la presenza di malattie o la carenza di nutrienti, e la distribuzione spaziale delle colture stesse. Inoltre, la possibilità di applicare serbatoi agli stessi droni consente agli agricoltori di utilizzarli per l'irrigazione o per la distribuzione di prodotti chimici, quali fertilizzanti, erbicidi o pesticidi, per il trattamento delle coltivazioni. Questa modalità di utilizzo, denominata "spraying" (vedi Figura 1.1), permette di risparmiare una significativa quantità di prodotto ed allo stesso tempo di raccogliere informazioni utili riguardanti il processo di distribuzione e lo stato delle piante.[5] Infine, un'altra importante applicazione di questi sistemi aerei nell'ambito dell'agricoltura è quella della videosorveglianza delle aree coltivate, che consente agli agricoltori di monitorare i propri possedimenti e di garantire la sicurezza e l'efficienza delle operazioni agricole.[6]

Figura 1.1: Applicazione dei droni nell'agricoltura di precisione, droni per l'irrigazione
In questa figura si può osservare un drone DJI modello MG-1P, dotato di serbatoio, utilizzato per l'irrigazione dei campi attraverso avanzate tecniche di "spraying".[8]

1.2 Contesto e motivazioni del progetto

Come si è dunque potuto osservare, il contesto attuale dell'agricoltura si trova di fronte a sfide estremamente complesse; ed è proprio in questo contesto che i droni emergono come una soluzione promettente in grado di rispondere a molti di questi interrogativi.

L'obiettivo principale del progetto è quello di esplorare le potenzialità ed i benefici dell'utilizzo dei droni nell'agricoltura di precisione, in particolare nell'ambito dell'ultima delle applicazioni elencate nel paragrafo precedente: il monitoraggio e la sorveglianza dei campi.

L'intero sistema mira a sviluppare un approccio innovativo per affrontare le sfide dell'agricoltura moderna, promuovendo una gestione intelligente delle risorse, l'aumento della produttività e la riduzione dell'impatto ambientale.

1.3 Obiettivi e metodologia della tesi

Il progetto prevede lo sviluppo di un sistema di monitoraggio distribuito che utilizzi un drone come vettore di vigilanza. L'agricoltore, in veste di utente finale, deve essere in grado di selezionare una posizione prestabilita a cui inviare il drone e, conseguentemente, ricevere un'immagine in tempo reale che gli consenta di visualizzare lo stato attuale dell'area che ha scelto di monitorare.

Per fare ciò, il sistema di sorveglianza è stato per prima cosa progettato ed implementato a livello locale. L'obiettivo successivo è stato quello di ottimizzare le funzionalità e le prestazioni di questa infrastruttura locale, distribuendola su di una piattaforma Cloud per garantirne la scalabilità, la flessibilità e l'accessibilità.

Background teorico

2.1 Principi e funzionamento del drone

Il drone utilizzato per lo sviluppo di questo progetto è il ***Parrot AR Drone 2.0*** (vedi Figura 2.1), un quadricottero prodotto dall'azienda francese Parrot SA che si basa su di un sistema operativo Linux con kernel versione 2.6.32 (BusyBox). Questo drone dispone di un processore ARM Cortex A8 da 1 GHz a 32 bit con 1 Gbit di RAM DDR2 a 200 MHz. Inoltre, è dotato di un modulo GPS in grado di geolocalizzare e tenere traccia della posizione del drone, di una telecamera frontale ad alta definizione e di una telecamera verticale per offrire una copertura completa dell'area. In particolare la telecamera frontale, utilizzata in questo progetto, presenta un'alta definizione a 720p. La connessione al drone dal dispositivo in cui viene lanciato il core ROS avviene grazie ad un hotspot WiFi. Per mezzo di questa interconnessione, i nodi ROS sono abilitati ad impartire al drone due comandi fondamentali: il movimento tra la base e la posizione desiderata e l'acquisizione dell'immagine. [7] [9]

Per la prima di queste attività, vale a dire lo spostamento del drone, è stata utilizzata ***pyardrone***, una libreria cliente Python. Attraverso le funzioni definite all'interno di questa libreria è possibile far decollare il drone, muoverlo avanti ed indietro, farlo ruotare in senso orario o antiorario e farlo atterrare.

Per catturare le immagini tramite la fotocamera frontale è stata invece utilizzata la libreria Python ***cv2,*** più comunemente conosciuta con il nome di ***OpenCV.*** Essa è una libreria software multipiattaforma dedicata alla visione artificiale che in questo caso permette di accedere alla fotocamera del drone in tempo reale, di immortalare l'istante desiderato e di salvare il frame localmente.

Figura 2.1: Parrot AR Drone 2.0

2.2 Framework ROS per lo sviluppo di robotica

Come già accennato in precedenza, il drone comunica con il resto del sistema attraverso il framework ROS ed i relativi nodi creati in esso.

Il ***Robot Operating System*** (ROS) è un framework di sviluppo open-source, ampliamente utilizzato nella robotica, che fornisce una serie di strumenti per semplificare lo sviluppo, il controllo e l'interazione tra i componenti di un sistema robotico. L'architettura di questo middleware è fortemente ispirata a quella delle reti di telecomunicazioni TCP/IP. Infatti, la struttura generale di un sistema robotico che utilizza ROS si può descrivere come una rete di nodi interconnessi tra loro, che si scambiano messaggi accedendo a servizi messi a disposizione da altri nodi. I nodi, che sono solitamente sviluppati attraverso l'uso di librerie come *roscpp*, *rospy* e *roslisp* per la programmazione rispettivamente in C++, Python e Lisp, rappresentano i singoli processi computazionali che possono comunicare tra di loro tramite la pubblicazione e la sottoscrizione di messaggi sui topic. I topic fungono quindi da canali di comunicazione. Fondamentale all'interno di questa architettura è il nodo Master, il quale si occupa di registrare nuovi nodi all'interno della rete, di gestire le connessioni e di instradare i messaggi, permettendo l'accesso ai servizi ad un nodo piuttosto che ad un altro. (vedi Figura 2.2)

ROS è pensato per operare con sistemi operativi Linux; in particolare i principali OS in cui avviene lo sviluppo di questo tipo di applicazioni sono Ubuntu e Debian, anche se la compatibilità di ROS è in continua espansione grazie ai numerosi pacchetti che ne ampliano la portabilità.[10]

Nel contesto di questo progetto, oltre all'utilizzo del framework ROS, è stato necessario adottare la tecnologia ***ROSBridge*** per far comunicare la rete di nodi con l' interfaccia utente locale. Questa tecnologia espone le funzionalità dei sistemi ROS ai sistemi non-ROS: ***ROSBridge*** consente infatti la comunicazione tramite websockets HTML o standard TCP/IP sockets. Ciò fornisce ai sistemi non-ROS meccanismi di comunicazione similari a ROS, inclusa la pubblicazione/sottoscrizione di topic; questa progettazione architetturale espande quindi l'ambiente per l'esecuzione di applicazioni robotiche a qualsiasi linguaggio di programmazione che supporti socket IP.[11] (vedi Figura 2.3)

Riassumendo, ROS rappresenta uno strumento chiave per lo sviluppo di applicazioni robotiche poiché offre un meccanismo di comunicazione facile ed agevole tra gli elementi del sistema.

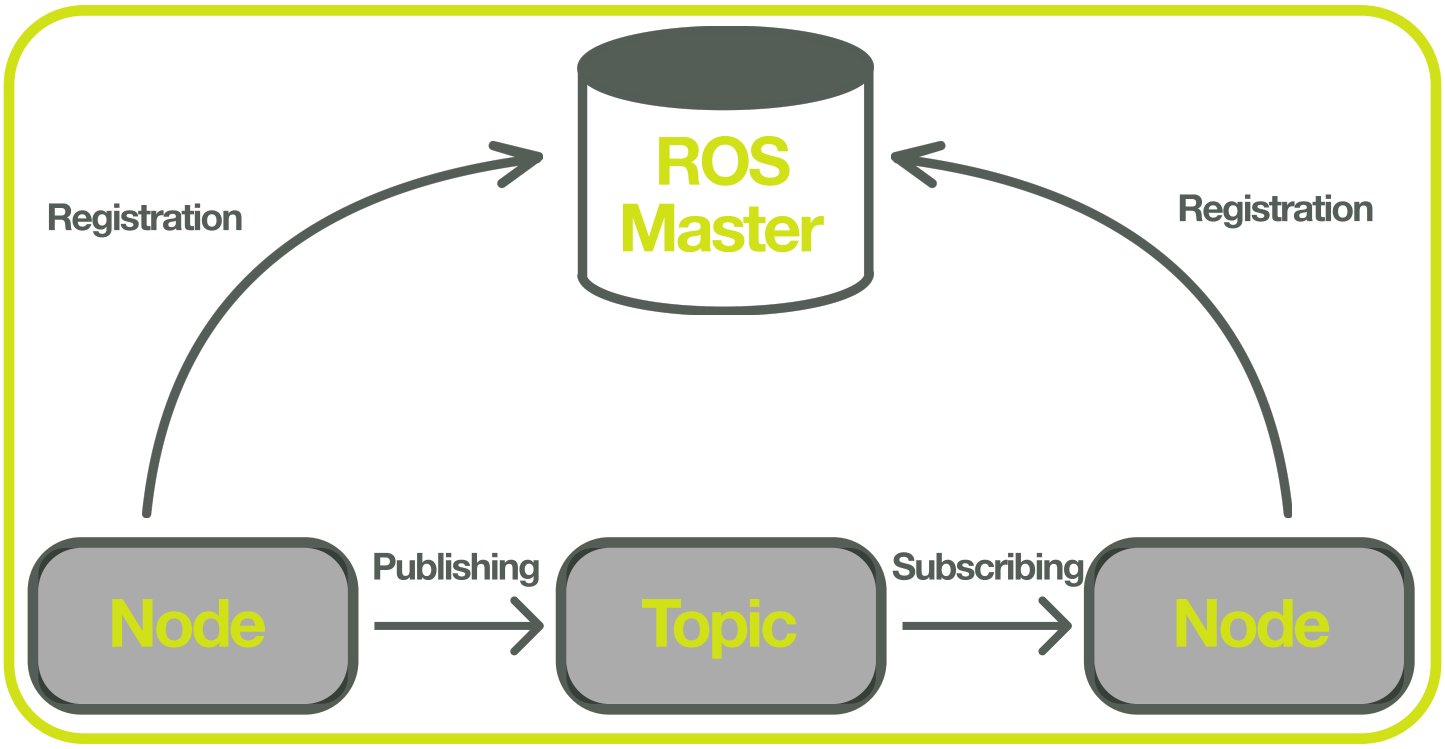

Figura 2.2: Architettura di una rete di nodi ROS
Nella figura si possono notare gli elementi di una rete di nodi ROS e le relative connessioni.

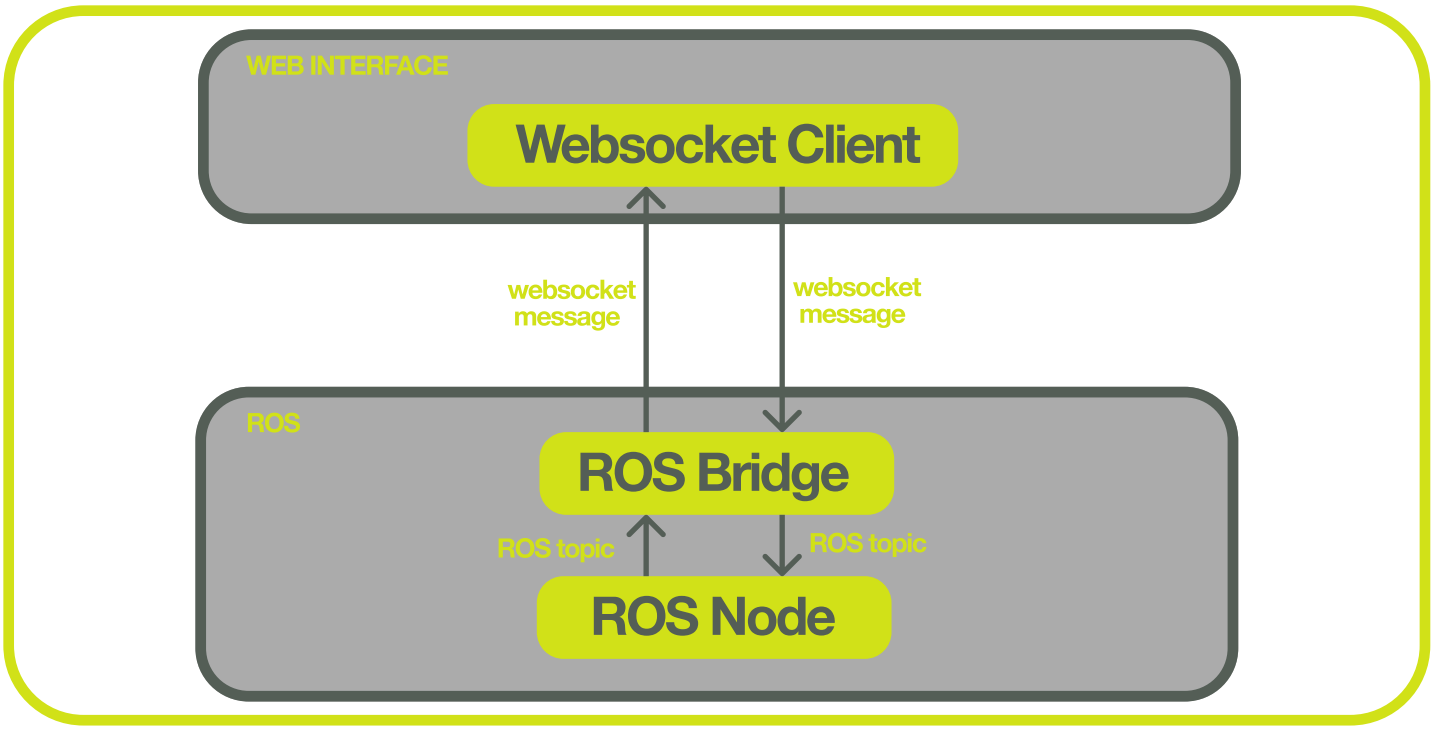

Figura 2.3: Rappresentazione del flusso di comunicazione usando ROSBridge
Nella figura si può osservare la rappresentazione grafica di una comunicazione tra l'intorno di ROS ed una interfaccia web, collegati mediante l'uso di ROSBridge.

2.3 Architetture software distribuite in cloud computing

Il progetto, che prevede lo sviluppo di questo sistema di monitoraggio distribuito, parte dunque da un prototipo a livello locale e successivamente lo ottimizza in modo tale da distribuirlo sul Cloud e renderlo così scalabile ed accessibile a livello Internet; per fare ciò si fa uso del *cloud computing*.

Il termine *cloud computing* descrive sia una piattaforma che un tipo di applicazione. Una piattaforma di *cloud computing* provvede in modo dinamico alla fornitura, alla configurazione ed all'amministrazione di server secondo le necessità. Le applicazioni cloud sono applicazioni che vengono estese per essere accessibili tramite Internet; queste applicazioni utilizzano grandi data center e robusti server in grado di ospitare applicazioni e servizi web.[12]

Le architetture software distribuite nel *cloud computing* sono fondamentali nell'ambito dello sviluppo informatico: esse consentono di creare e gestire applicazioni complesse distribuendo i carichi di lavoro su risorse cloud remote invece di eseguirli su di un singolo server locale. Le architetture cloud risolvono dunque molte delle difficoltà legate all'elaborazione di dati su larga scala quali: ottenere il numero di macchine necessario per un'applicazione, distribuire e coordinare il lavoro su diversi dispositivi, eseguire i processi su di queste macchine e scalarli automaticamente in base al carico dinamico ed infine eliminare le istanze quando il processo è terminato.
Le applicazioni basate sulle architetture cloud vengono eseguite nel Cloud, dove la posizione fisica dell'infrastruttura è determinata dal provider. Esse approfittano delle semplici API dei servizi accessibili via Internet che si adattano su richiesta e che hanno una robustezza industriale, dove la complessa logica di affidabilità e scalabilità dei servizi sottostanti rimane implementata e nascosta all'interno del Cloud.[13]

2.4 Google Cloud Platform come piattaforma di cloud computing

Nel contesto del progetto, l'architettura cloud è stata implementata utilizzando i servizi di ***Google Cloud Platform*** per sfruttarne l'elasticità, la scalabilità e la disponibilità delle risorse. In particolare, sono stati impiegati i servizi ***Google App Engine*** per l'esecuzione delle applicazioni nel Cloud e ***Google Cloud SQL*** in ***MySQL*** per la gestione dei database e la memorizzazione dei dati delle applicazioni.

Google App Engine è un sistema che espone varie parti dell'infrastruttura scalabile di Google in modo che si possano scrivere applicazioni lato server sopra di esse. In sostanza, si tratta di una piattaforma che consente agli utenti di eseguire e ospitare le proprie applicazioni web sull'infrastruttura di Google. Queste applicazioni sono facili da costruire, da mantenere e da scalare in termini di traffico e archiviazione dei dati. Utilizzando ***Google App Engine***, non ci sono server da gestire e non sono necessari amministratori. L'idea è che il fruitore del servizio carichi semplicemente la propria applicazione ed essa sia pronta ad essere utilizzata dai propri clienti. L'utente ha la possibilità di scegliere se far servire il proprio prodotto dal dominio gratuito appspot.com o consentire a *Google Apps* di servirlo dal dominio scelto dal cliente.[14]

Google Cloud SQL è un servizio di database relazionale per ***MySQL*** completamente gestito. Ogni istanza creata con questo servizio è basata su di una macchina virtuale in esecuzione su di un server Google Cloud host e gestisce il programma di database. La base di dati creata presso l'istanza, che viene poi memorizzata su di un dispositivo di archiviazione di rete scalabile e durevole, può permettere l'accesso ai propri dati tramite un indirizzo IP pubblico o privato, a seconda della configurazione.[15]

Progettazione del sistema

3.1 Analisi dei requisiti funzionali e non funzionali

Per prima cosa verrà analizzata la struttura del sistema di monitoraggio; per fare ciò, è importante definire i requisiti del progetto.

Questo sistema di monitoraggio deve consentire agli utenti di accedere all'interfaccia dell'applicazione e di scegliere tra tre diverse posizioni a cui indirizzare il drone, in modo tale da poter monitorare una specifica porzione d'area. Le interfacce devono dimostrarsi user-friendly, in modo tale da permettere un accesso intuitivo a questo servizio. Inoltre, il servizio distribuito deve garantire l'accessibilità indipendentemente dalla posizione geografica dell'utente, in modo tale da non vincolarlo alla rete locale a cui è collegato il drone. Una volta fatto ciò, il sistema deve restituire all'utente un'immagine comprensibile ed in un tempo ragionevole. L'utente deve inoltre avere la possibilità di eseguire tali operazioni in modo iterativo, al fine di monitorare le diverse sezioni delle proprietà in momenti distinti. É importante quindi garantire la scalabilità del sistema, ovvero una capacità di gestione dell'aumento di carico utente e di carico dati, senza prevedere una perdita in senso di prestazioni.

3.2 Descrizione del sistema a livello locale

La fase intermedia della progettazione del sistema di monitoraggio distribuito, indirizzata allo sviluppo di un sistema funzionante soltanto a livello locale, prevede due principali protagonisti che comunicano tra loro all'interno dell'Intranet: l'utente ed il drone.

Osservando l'immagine (vedi Figura 3.1) si possono analizzare i singoli elementi che vanno a costituire il sistema locale.

L'interfaccia web per l'utente finale viene progettata per consentire all'utente di interagire con il sistema. Il Frontend viene sviluppato utilizzando tecnologie web standard HTML e presenta un'intuitiva schermata che permette all'utente di scegliere con facilità la posizione a cui inviare il drone. L'archivio JavaScript funge da supporto per il Frontend dell'interfaccia web e contiene librerie, framework e script personalizzati in grado di gestire la logica dell'interfaccia utente e le relative comunicazioni con i nodi ROS. In particolare, il codice JavaScript definisce un Publisher in grado di passare ai nodi ROS la posizione scelta dall'utente e, successivamente, un Subscriber che si sottoscrive ad uno dei topic in cui pubblicano i nodi per ricevere le informazioni riguardo la posizione locale dell'immagine scattata. Con quest'ultima informazione, l'archivio JavaScript permette la visualizzazione dell'immagine nella schermata principale. I nodi ROS, responsabili del controllo del drone, gestiscono il movimento di esso in funzione della posizione letta e restituiscono i dati riguardanti l'immagine salvata localmente nel dispositivo.

Questo primo prototipo di sistema, seppur funzionante, non garantisce l'accessibilità completa del servizio. Per questo verrà nel prossimo paragrafo descritta un'implementazione distribuita del progetto.

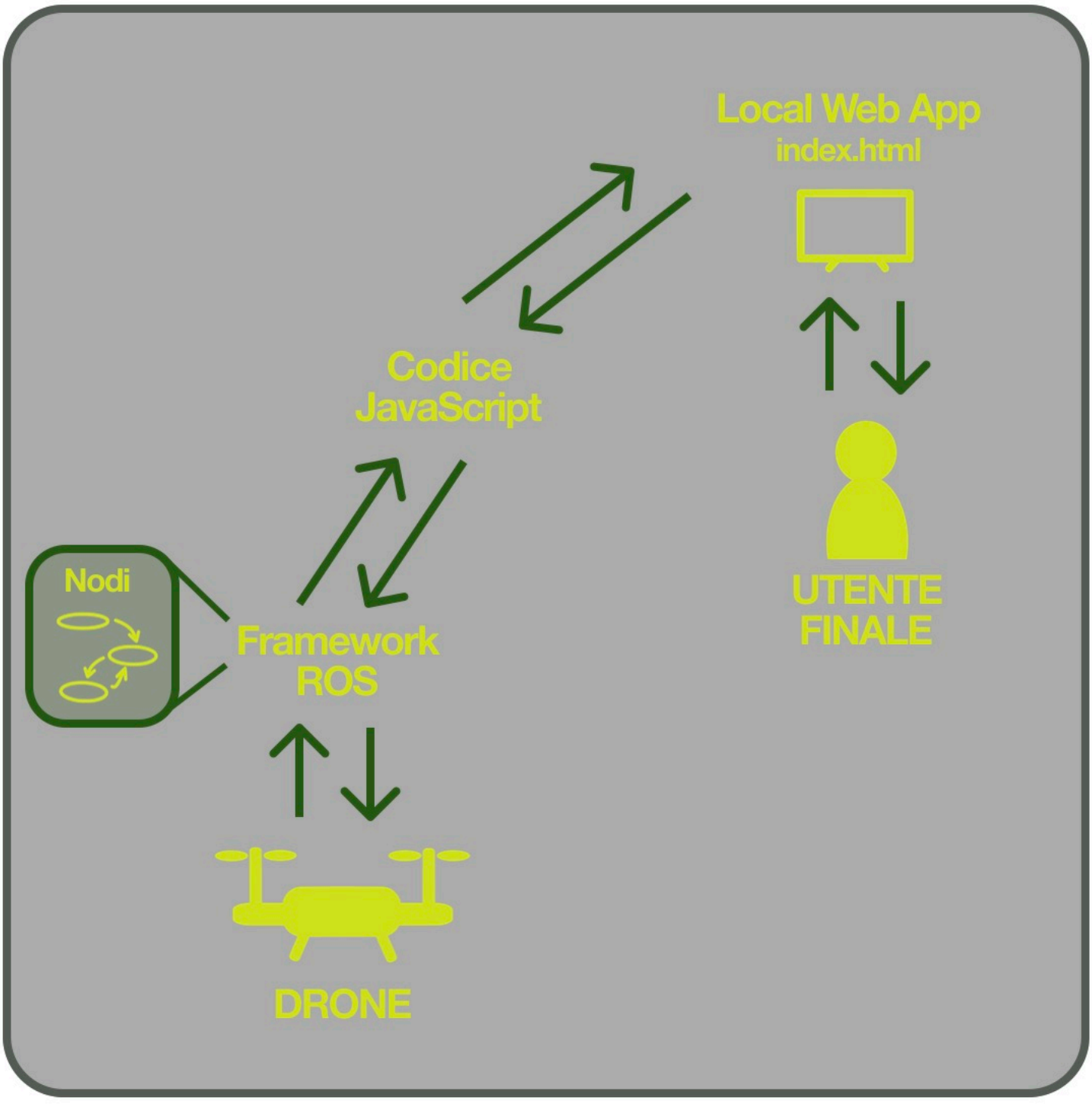

Figura 3.1: Rappresentazione del sistema locale

Lo schema in figura rappresenta il sistema implementato localmente, mettendo in evidenza tutti gli elementi che lo compongono e le rispettive relazioni.

3.3 Descrizione del sistema distribuito nel Cloud

Il nuovo sistema, che non lavora più soltanto a livello di Intranet ma che è stato espanso a livello Internet, presenta ora un nuovo elemento prima assente: il Cloud. Inoltre, è stata aggiunta in questa successiva implementazione la possibilità di utilizzare il servizio tramite un'applicazione Android.

Come si può osservare (vedi Figura 3.2), questo sistema è formato da diversi elementi e diversi livelli di astrazione.

Come prima, a livello di Intranet compare il sottosistema del robot aereo, il drone, il cui controllo si basa sul framework ROS. All'interno di questo framework vengono sviluppati una molteplicità di nodi in grado di comunicare tanto con il drone per fornirgli le indicazioni sul movimento e le operazioni da compiere, quanto con il Cloud per la consulta o la memorizzazione dei dati. La rete di nodi preleva quindi informazioni riguardo alla prossima posizione a cui indirizzare il drone dal Cloud, invia un comando al drone per far sì che questo si posizioni nel luogo prestabilito e scatti una foto che verrà prontamente immagazzinata nel database implementato nel Cloud, ed infine riporta il drone alla base di partenza. La piattaforma Cloud, come si è potuto comprendere, contiene un database di informazioni che conserva i dati riguardanti le posizioni richieste dall'utente e le relative immagini scattate dal drone: per fare tutto ciò utilizza lo strumento di raccolta ***MySQL*** di ***Google Cloud Platform***. Dall'altro lato, ad un livello di astrazione più elevato, si posiziona l'utente finale che è in grado di inviare comandi al robot tramite due interfacce: una piattaforma web ed un'applicazione Android. Il Backend di queste due interfacce viene distribuito nel Cloud utilizzando lo strumento ***Google App Engine***, in modo tale da rendere le applicazioni accessibili anche al di fuori della rete locale. I codici Backend comunicano con i database del Cloud inviando i comandi di posizione e recuperando i dati relativi alle immagini, le quali verranno poi mostrate all'utente.

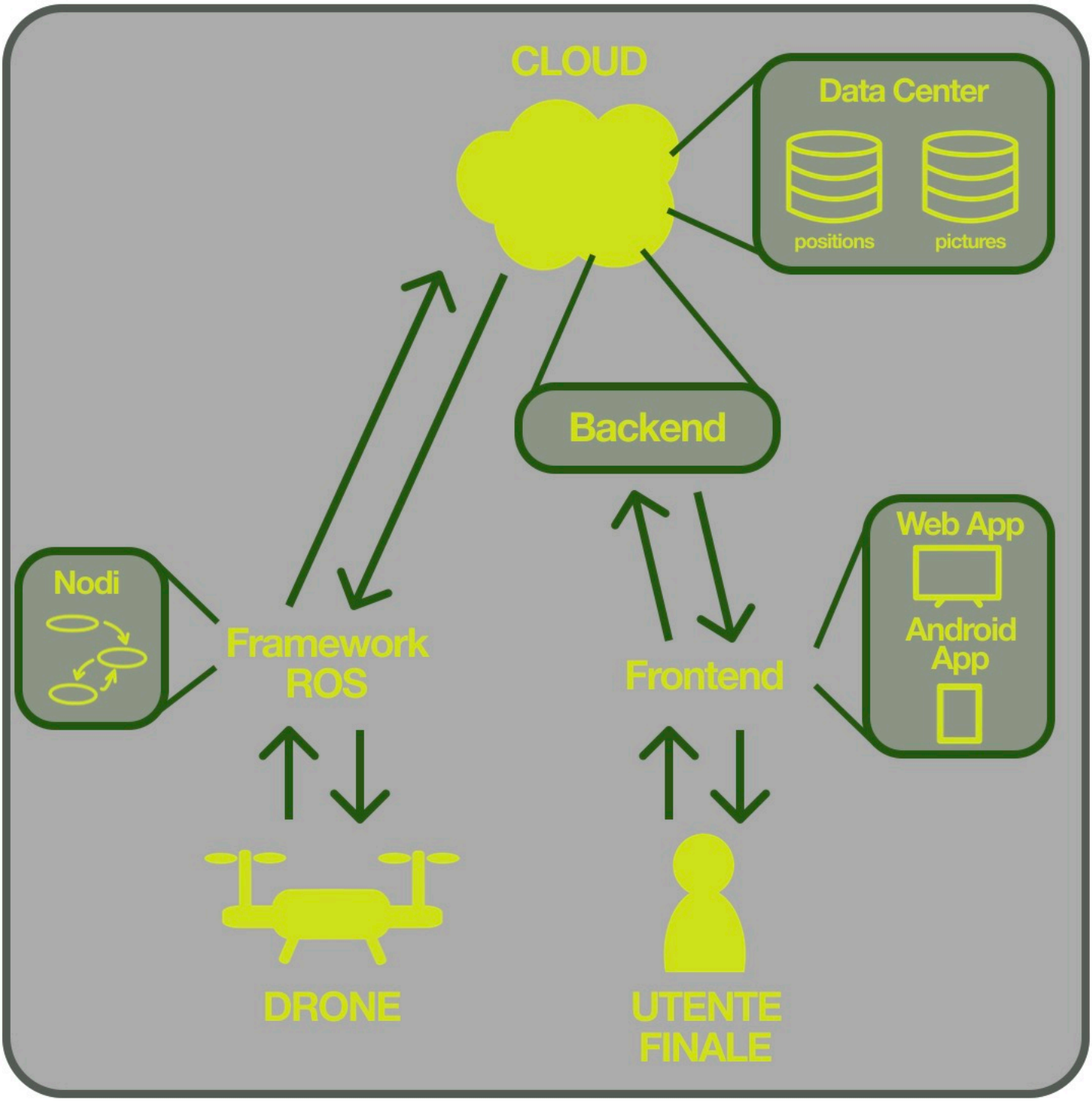

Figura 3.2: Rappresentazione del sistema distribuito

Lo schema in figura rappresenta il sistema distribuito nel Cloud, con i suoi elementi e le relazioni implementate tra di loro.

Implementazione del sistema locale

4.1 Applicazione web locale

Si parte quindi con l'analisi dell'implementazione del sistema locale, il quale servirà da guida per lo sviluppo successivo del sistema distribuito.

Come spiegato nel capitolo precedente, l'applicazione web locale è semplicemente dotata di una interfaccia HTML che descrive l'aspetto della pagina e di un file Javascript che definisce il comportamento interattivo della pagina.

L'interfaccia HTML è il punto di ingresso dell'applicazione web locale e ne definisce la struttura e l'aspetto: si può osservare la pagina iniziale (vedi Figura 4.1), chiamata index.html, a cui approda l'utente nel momento in cui vuole utilizzare il servizio. L'utente può qui scegliere a quale tra le tre posizioni inviare il drone per ricevere l'immagine corrispondente: ogni bottone rimanda infatti ad una funzione definita nel file Javascript a seconda della scelta. Viene qui riportato (vedi Codice 4.1) soltanto il corpo del codice index.html, in quanto il resto dello script è dettato esclusivamente dalle scelte stilistiche della pagina e per questo non rilevante nella spiegazione dell'implementazione del progetto.

Il codice Javascript (vedi Codice 4.2) presenta invece le definizioni di tutte le funzioni che vengono chiamate in index.html. Per prima cosa la funzione connectROS( ), a cui si rimanda ogni volta che viene caricata la pagina web, è la responsabile della connessione al server ROS e della creazione del topic *'chatter'*. In questo topic viene pubblicata una stringa indicante la posizione scelta ogni qualvolta l'utente seleziona una delle tre opzioni presentate dall'interfaccia web. Le tre funzioni position1( ), position2( ) e position3( ) sono coloro le quali si incaricano di pubblicare la stringa di posizione. Una volta resa pubblica la catena di caratteri, queste rimandano alla funzione redirect( ) che riceve a sua volta una stringa con il nome dell'immagine scattata dal drone e salvata localmente; la ricezione di questa informazione avviene attraverso la sottoscrizione al topic *'image_topic'*. Questo processo comincia con un delay di un minuto, per garantire il tempo necessario per lo spostamento del drone. A questo punto il codice recupera l'immagine e la carica nella stessa pagina web, permettendone la visualizzazione.

4. Implementazione del sistema locale

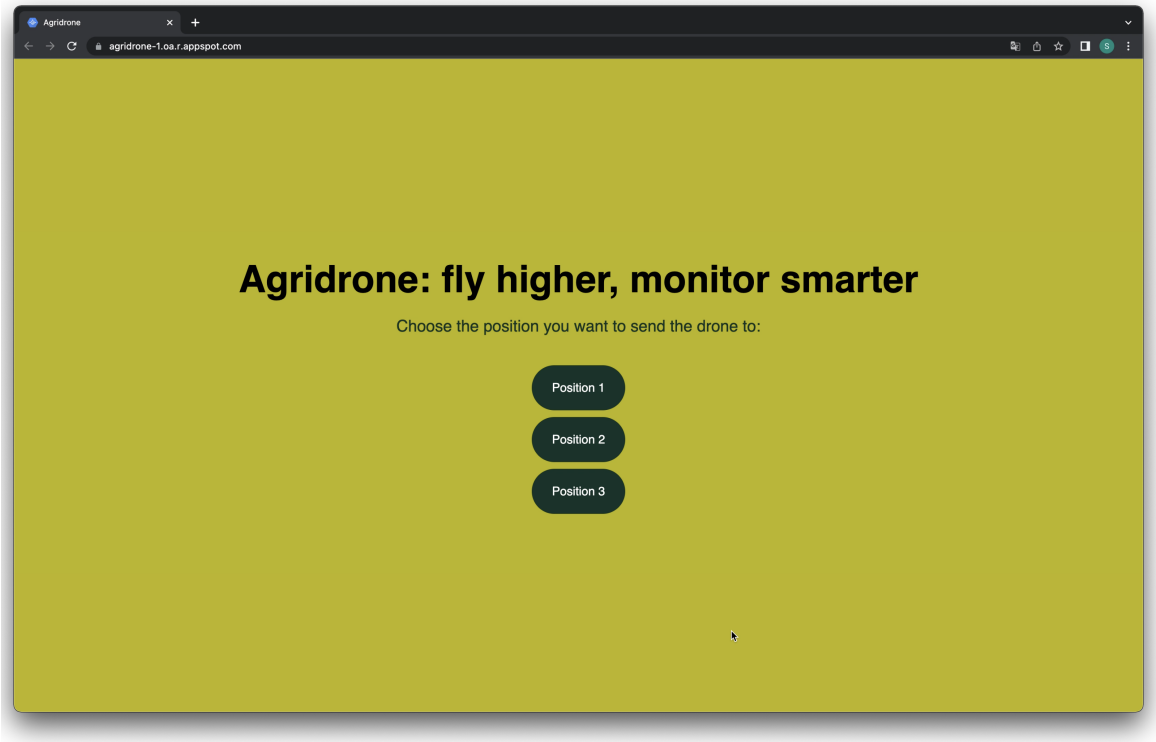

Figura 4.1: Pagina iniziale dell'interfaccia utente web

In questo schermata viene presentata la pagina iniziale dell'applicazione web, dove l'utente finale può scegliere a quale posizione inviare il drone premendo uno dei tre bottoni proposti.

```
68    <body onload="connectRos()">
69
70            <h1>Agridrone: fly higher, monitor smarter</h1>
71            <p>Choose the position you want to send the drone to:</p>
72            <div class="button-container">
73              <button class="button" onclick="position1()">Position 1</button>
74              <button class="button" onclick="position2()">Position 2</button>
75              <button class="button" onclick="position3()">Position 3</button>
76            </div>
```

Codice 4.1: index.html (sistema locale)

Questo tratto di codice riporta il corpo dell'interfaccia web locale, dove il click di ogni bottone rimanda alla rispettiva funzione definita nel codice JavaScript.

4. Implementazione del sistema locale

```
// Connecting to ROS
// -----------------

var ros = new ROSLIB.Ros({
    url : 'ws://localhost:9090'
});

var position = null;
var image = null;

function connectROS(){
    ros.on('connection', function() {
        console.log('Connected to websocket server.');
        position = new ROSLIB.Topic({
            ros : ros,
            name : '/chatter',
            messageType : 'std_msgs/String'
        });
    });

    ros.on('error', function(error) {
        console.log('Error connecting to websocket server: ', error);
    });

    ros.on('close', function() {
        console.log('Connection to websocket server closed.');
    });
}


function position1(){

    var str = new ROSLIB.Message({
        data:'Position 1'
    });

    position.publish(str);
    redirect();

}

function position2(){

    var str = new ROSLIB.Message({
        data:'Position 2'
    });
```

```
47
48      position.publish(str);
49      redirect();
50
51  }
52
53  function position3(){
54
55      var str = new ROSLIB.Message({
56          data:'Position 3'
57      });
58
59      position.publish(str);
60      redirect();
61
62  }
63
64  function redirect(){
65
66      image = new ROSLIB.Topic({
67          ros : ros,
68          name : '/image_topic',
69          messageType : 'std_msgs/String'
70      })
71
72      image.subscribe(callback);
73
74      function callback(message){
75          setTimeout(displayImage(), 60000);
76
77          function displayImage(){
78              var img = document.createElement("img");
79              img_path = "/home/sofia2110/" + message;
80              img.src = img_path;
81              document.body.appendChild(img);
82      }
83      }
84
85  }
```

Codice 4.2: codicejavascript.js

Il codice JavaScript permette la connessione al framework ROS ed ai rispettivi nodi e definisce le funzioni necessarie per agire secondo questa connessione.

4.2 Nodi ROS nel sistema locale

Si passa ora all'analisi di un altro elemento fondamentale nello sviluppo del sistema locale: la rete di nodi nel framework ROS.

Il pacchetto che implementa questi nodi prende il nome di agridrone_local e contiene un totale di 6 codici, scritti in linguaggio Python, in grado di leggere la posizione scelta dall'utente e pubblicata dalle funzioni del file JavaScript; i nodi indirizzano poi il drone affinchè possa scattare la fotografia corrispondente. In particolare, di questi 6 codici, i primi tre svolgono la funzione di inviare il drone ad una delle tre posizioni possibili e gli ultimi tre svolgono invece la funzione di scattare la foto e riportare il drone alla base iniziale. Il seguente schema (vedi Figura 4.2) rappresenta l'architettura ROS costruita per il sistema locale.

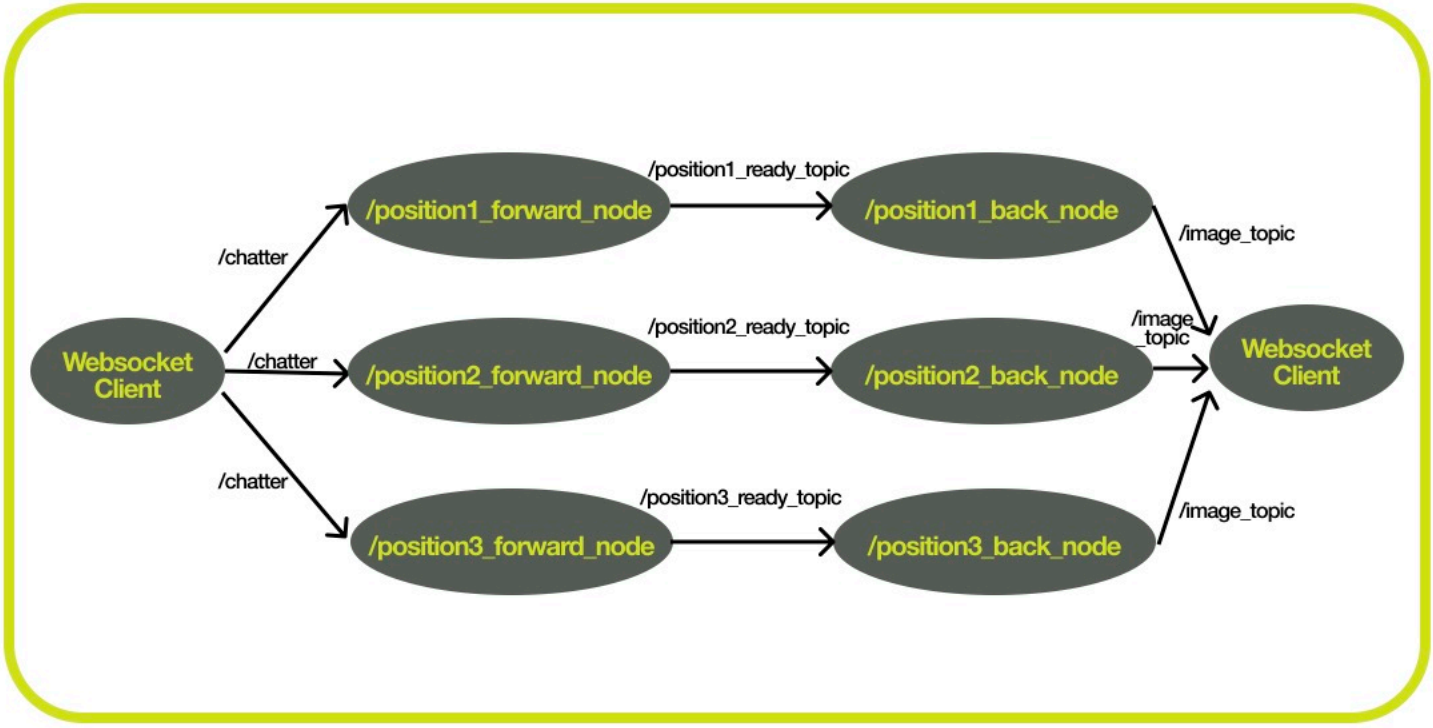

Figura 4.2: Rete di nodi ROS del sistema locale
Si può qui osservare la rete di nodi ROS utilizzati per la gestione del drone, con i rispettivi topic in cui essi pubblicano o a cui si sottoscrivono.

4. Implementazione del sistema locale

I primi tre nodi position1_forward.py, position2_forward.py e position3_forward.py si differenziano tra di loro soltanto per i particolari comandi di movimento del drone e per il nome dei rispettivi topic in cui pubblicano. Per questo, viene di seguito analizzato solo uno di questi codici.

Il nodo (vedi Codice 4.3) è allo stesso tempo un Publisher ed un Subscriber. In particolare esso si sottoscrive al topic *'chatter'*, che era stato creato ed utilizzato nel file Javascript nel contesto dell'interfaccia web, e riceve la stringa corrispondente alla posizione desiderata dall'utente in quel momento. Se la stringa che legge corrisponde alla posizione per cui è stato implementato il codice, il drone viene fatto muovere tramite i comandi della libreria ***pyardrone*** ed atterrare nel luogo corretto. Una volta fatto ciò, il nodo procede con il pubblicare la stringa "Ready" per aggiornare il sistema sullo stato del processo.

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
import time
from pyardrone import ARDrone

pub = None

def callback(data):

    rospy.loginfo(rospy.get_caller_id()+"I heard %s", data.data)

    drone = ARDrone()

    if data.data == "Position 1":

        drone.takeoff()
        time.sleep(5)
        now = time.time()
        while(time.time()-now < 3):
            drone.move(forward=0.2)
        drone.land()
        time.sleep(5)
        drone.close()

        pub.publish("Ready")

def listener_publisher():

    global pub
```

```
31
32      rospy.init_node('position1_forward_node', anonymous=True)
33      rospy.Subscriber("chatter", String, callback)
34      pub = rospy.Publisher('position1_ready_topic', String, queue_size=10)
35      rospy.spin()
36
37  if __name__ == '__main__':
38
39      try:
40          listener_publisher()
41      except rospy.ROSInterruptException:
42          pass
```

Codice 4.3: position1_forward.py (agridrone_local)

Il codice, che viene preso come esempio rappresentativo anche per i nodi position2_forward e position3_forward, permette il movimento del drone verso la posizione selezionata dall'utente.

Allo stesso modo, verrà presentato soltanto uno tra i codici position1_back.py, position2_back.py e position3_back.py.

Questo nodo (vedi Codice 4.4), che ancora una volta è un Publisher ed un Subscriber allo stesso tempo, si incarica di leggere la catena di caratteri pubblicata dal nodo precedente. Se la stringa risulta essere il messaggio "Ready", il drone procede con la cattura dell'immagine tramite i comandi della libreria ***OpenCV***. Successivamente, il nodo riporta il drone alla base iniziale con l'utilizzo della libreria ***pyardrone*** e pubblica il nome dell'immagine catturata. Questo nome, che contiene informazioni riguardo la data e l'ora dell'evento, verrà come anticipato sfruttato dal codice JavaScript per mostrare l'immagine nell'interfaccia web.

```
1   #!/usr/bin/env python3
2   import rospy
3   from std_msgs.msg import String
4   import time
5   from pyardrone import ARDrone
6   import cv2
7   import logging
8   import datetime
9
10  pub = None
11
12  def callback(data):
13
```

```
14      rospy.loginfo(rospy.get_caller_id()+"I heard %s", data.data)
15      drone = ARDrone()
16
17      if data.data == "Ready":
18
19          drone.video_ready.wait()
20          cap = cv2.VideoCapture(0)
21          logging.basicConfig(level=logging.DEBUG)
22          frame = cap.read()
23          image_name = "framep1" + datetime.datetime.now().strftime("%Y-%m-%d_%H:%M:%S")
24          cv2.imwrite(image_name, drone.frame)
25
26          drone.takeoff()
27          time.sleep(3)
28          now = time.time()
29          while(time.time()-now < 2):
30              drone.move(cw=0.5)
31          now = time.time()
32          while(time.time()-now < 5):
33              drone.move(forward=0.1)
34          now = time.time()
35          while(time.time()-now < 3):
36              drone.move(cw=0.6)
37          drone.land()
38          time.sleep(5)
39          drone.close()
40          pub.publish(image_name)
41
42  def listener_publisher():
43
44      global pub
45      rospy.init_node('position1_back_node', anonymous=True)
46      rospy.Subscriber('position1_ready_topic', String, callback)
47      pub = rospy.Publisher('image_topic', String, queue_size=10)
48      rospy.spin()
49
50  if __name__ == '__main__':
51      try:
52          listener_publisher()
53      except rospy.ROSInterruptException:
54          pass
```

Codice 4.4: position1_back.py (agridrone_local e agridrone_pkg)

Il codice, che può essere preso come esempio rappresentativo anche per i nodi position2_back e position3_back, scatta l'immagine da presentare all'utente e riporta il drone alla base iniziale.

Implementazione del sistema distribuito

5.1 Nodi ROS nell'Intranet

Una volta compreso come funziona il sistema di monitoraggio locale, è possibile proseguire con l'ampliamento di quest'ultimo in modo tale da poterlo espandere a livello Internet, ossia in modalità condivisa. Per questo, i prossimi paragrafi descrivono un sistema distribuito che riprende gli aspetti fondamentali del sistema locale appena studiato, con significative evoluzioni in prospettiva della distribuzione nel Cloud.

Viene in primo luogo analizzata la parte del sistema che comunica direttamente con il drone: la rete di nodi sviluppata utilizzando il framework ROS.

Il pacchetto implementato per descrivere questa parte di progetto, chiamato agridrone_pkg, contiene 8 nodi scritti in linguaggio Python che comunicano tra di loro a formare una rete in grado di gestire il drone nei suoi spostamenti, scattare la fotografia corrispondente alla posizione scelta e scambiare informazioni con il Cloud. A seguire (vedi Figura 5.1) uno schema che rappresenta le connessioni tra gli elementi nel framework ROS.

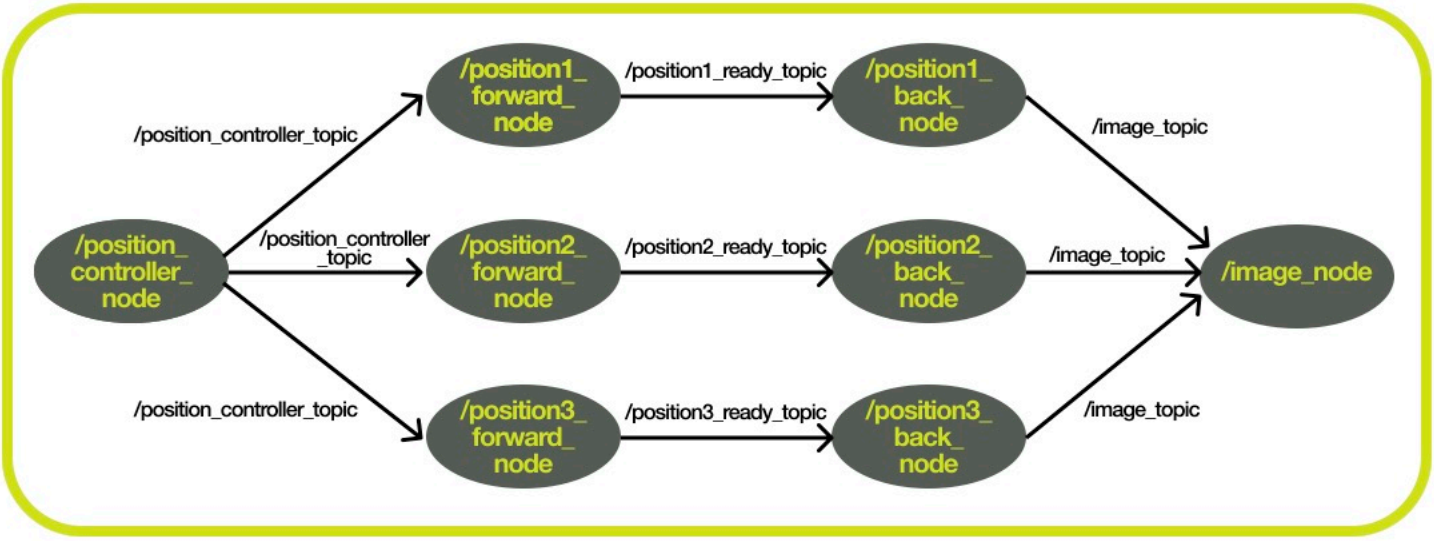

Figura 5.1: Rete di nodi ROS del sistema distribuito

Si può qui osservare lo schema dei nodi ROS utilizzati nel sistema distribuito nel Cloud.

5. Implementazione del sistema distribuito

Il prossimo passaggio analizza il codice di ogni componente della rete di elementi ROS per descrivere come viene implementato il comportamento del sistema a livello di Intranet.

Il primo nodo, position_controller.py (vedi Codice 5.1), è un Publisher. Questo utilizza infatti le funzioni della libreria ***mysql.connector*** per connettersi al database ***MySQL*** implementato presso la infrastruttura Google Cloud, prelevare l'ultima posizione inserita nella base di dati e pubblicare la stringa nel relativo topic. Il nodo ha una frequenza di pubblicazione di 10 Hz: per questo, prima di pubblicare è importante verificare che l'informazione letta non sia la posizione già precedentemente processata. Inoltre viene effettuato un controllo sulla data e l'ora delle istruzioni: il codice recupera la data di caricamento della posizione e, se questa è stata immagazzinata meno di tre minuti prima della lettura, la pubblica. Nel topic associato, *'position_controller_topic'*, vengono quindi pubblicate delle stringhe che possono variare tra "Position 1", "Position 2" o "Position 3" in base alla scelta dell'utente finale.

```python
#!/usr/bin/env python3
import rospy
import mysql.connector
from std_msgs.msg import String
from datetime import datetime, timedelta

def publisher():

    pub = rospy.Publisher('position_controller_topic', String, queue_size=10)
    rospy.init_node('position_controller_node', anonymous=True)
    rate = rospy.Rate(10)

    previous_time = ""

    while not rospy.is_shutdown():
        current_time = datetime.now()
        try:
            connection = mysql.connector.connect(host='34.65.22.89',
                                                 database='position-db',
                                                 user='user',
                                                 password='1234')
            cursor = connection.cursor()

            cursor.execute("SELECT * FROM positions WHERE MAX(id)")

            row = cursor.fetchone()

            current_time = datetime.now()
            command_time = datetime.strptime(row[2], '%Y-%m-%d %H:%M:%S')
```

5. Implementazione del sistema distribuito

```
29              command_time = datetime.strptime(row[2], '%Y-%m-%d %H:%M:%S')
30              time_difference = current_time - command_time
31
32              if time_difference.total_seconds() < 180 and previous_time != command_time:
33                  pub.publish(row[1])
34                  previous_time = current_time
35
36              rate.sleep()
37
38          except mysql.connector.Error as error:
39              print(format(error))
40
41          finally:
42
43              if connection.is_connected():
44
45                  cursor.close()
46                  connection.close()
47
48  if __name__ == '__main__':
49
50      try:
51          publisher()
52      except rospy.ROSInterruptException:
53          pass
```

Codice 5.1: position_controller.py (agridrone_pkg)
Il codice del nodo si collega al database Google Cloud grazie alla libreria dedicata e pubblica nel proprio topic la posizione selezionata dall'utente.

Come prima, i nodi di indirizzamento del drone position1_forward.py, position2_forward.py e position3_forward.py si differenziano tra di loro per i comandi di movimento del drone ed il nome dei rispettivi topic. Verrà quindi analizzato solo uno di questi codici.

Il nodo (vedi Codice 5.1), che è allo stesso tempo un Publisher ed un Subscriber, è sottoscritto al topic *'position_controller_topic'* e, se la stringa che legge corrisponde alla posizione per cui è stato implementato il codice, utilizza le funzioni della libreria ***pyardrone*** per pilotare il drone verso l'area desiderata. Una volta raggiunta la destinazione, esso pubblica la stringa "Ready" per aggiornare lo stato del processo.

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
import time
from pyardrone import ARDrone

pub = None

def callback(data):

    rospy.loginfo(rospy.get_caller_id()+"I heard %s", data.data)

    drone = ARDrone()

    if data.data == "Position 1":

        drone.takeoff()
        time.sleep(5)
        now = time.time()
        while(time.time()-now < 3):
            drone.move(forward=0.2)
        drone.land()
        time.sleep(5)
        drone.close()

        pub.publish("Ready")

def listener_publisher():

    global pub
    rospy.init_node('position1_forward_node', anonymous=True)
    pub = rospy.Publisher('position1_ready_topic', String, queue_size=10)
    rospy.Subscriber('position_controller_topic', String, callback)

    rospy.spin()

if __name__ == '__main__':

    try:
        listener_publisher()
    except rospy.ROSInterruptException:
        pass
```

Codice 5.2: position1_forward.py (agridrone_pkg)
Questo nodo, preso come esempio rappresentativo anche per i nodi position2_forward e position3_forward, permette lo spostamento del drone verso la posizione registrata nel database.

5. Implementazione del sistema distribuito

Ancora una volta, verrà studiato soltanto uno tra i nodi position1_back.py, position2_back.py e position3_back.py vista la loro somiglianza.

In questo caso il codice del nodo nel sistema distribuito rimane invariato rispetto a quello del sistema locale. Per questo, si rimanda alla visualizzazione del Codice 4.4.

Il nodo è allo stesso tempo un Publisher ed un Subscriber; è infatti sottoscritto al topic position1_ready_topic, che gli dà il via per iniziare il processo di cattura dell'immagine e di ritorno del drone alla base. Per riuscire ad utilizzare la telecamera del drone e catturare le immagini delle posizioni viene utilizzata la libreria ***cv2***, più comunemente conosciuta come ***OpenCV***. L'immagine viene poi salvata e nominata per mezzo di una stringa composta dal programma: il nome contiene infatti informazioni riguardanti la posizione, la data e l'ora dell'acquisizione del frame. Il drone viene fatto muovere verso la base grazie alle funzioni della libreria ***pyardrone***, come in precedenza. Infine il nodo pubblica il nome dell'immagine catturata, che viene salvata localmente nel dispositivo in cui sta lavorando ROS, in modo tale da permettere al prossimo nodo della rete di recuperare il frame e salvarlo nel database.

Per concludere si analizzerà l'ultimo componente del pacchetto: image.py (vedi Codice 5.3). Come prima, viene utilizzata la libreria ***mysql.connector*** per comunicare con il Cloud. In particolare, in questo caso, il nodo è semplicemente un Subscriber che legge una stringa dal topic *'image_topic'*. Questa sequenza di caratteri contiene il nome dell'immagine che è appena stata salvata localmente. Il codice recupera quindi il frame desiderato e lo carica nel database implementato nel Cloud. É importante notare come il caricamento dell'immagine avviene in formato di dato binario attraverso la funzione convert_data definita all'interno del nodo stesso. Verrà esplicitata in seguito la modalità di memorizzazione di questo tipo di informazioni in una base di dati ***MySQL***.

```python
#!/usr/bin/env python3
import rospy
import mysql.connector
from std_msgs.msg import String

def convert_data(file_name):
    with open(file_name, 'rb') as file:
        binary_data = file.read()
    return binary_data

def callback(data):

    try:
        connection = mysql.connector.connect(host='34.65.22.89',
                                             database='position-db',
                                             user='user',
                                             password='1234')
        cursor = connection.cursor()
        image_path = "/home/sofia2110/" + data.data
        image_data = convert_data(image_path)
        cursor.execute("INSERT INTO pictures(picture) VALUES (%s)", image_data)
        connection.commit()
    except mysql.connector.Error as error:
        print(format(error))

    finally:
        if connection.is_connected():

            cursor.close()
            connection.close()

def listener():

    rospy.init_node('image_node', anonymous=True)
    rospy.Subscriber('image_topic', String, callback)
    rospy.spin()

if __name__ == '__main__':

    try:
        listener()
    except rospy.ROSInterruptException:
        pass
```

Codice 5.3: image.py (agridrone_pkg)
L'ultimo nodo del pacchetto raccoglie l'immagine scattata dal drone e salvata localmente e la carica nel database implementato sul Cloud, in modo tale da renderla accessibile alle interfacce utente.

5.2 Backend e Frontend della piattaforma web

La successiva fase dell'analisi riguarda il livello più elevato del sistema, quello a diretto contatto con l'utente finale: le interfacce utente. In particolare si avvia lo studio dall'interfaccia web distribuita, nelle sue due parti Frontend e Backend.

Il codice per il Frontend della web app è stato scritto in linguaggio HTML. La pagina iniziale index.html a cui approda l'utente nel momento in cui vuole utilizzare il servizio è dal punto di vista stilistico la stessa interfaccia utilizzata nel sistema locale (vedi Figura 4.1). L'utente può qui decidere a quale tra le tre posizioni inviare il drone per ricevere l'immagine corrispondente: ogni bottone rimanda infatti ad un'altra pagina web a seconda della scelta. Come prima, viene riportato di seguito soltanto il corpo del codice index.html (vedi Codice 5.4), in quanto il resto del codice è dettato esclusivamente dalle scelte stilistiche della pagina e per questo non rilevante alla luce dello studio tecnico e realizzativo del progetto.

```
68      <body>
69              <h1>Agridrone: fly higher, monitor smarter</h1>
70              <p>Choose the position you want to send the drone to:</p>
71              <div class="button-container">
72                  <button class="button" onclick="location.href='/position1'">Position 1</button>
73                  <button class="button" onclick="location.href='/position2'">Position 2</button>
74                  <button class="button" onclick="location.href='/position3'">Position 3</button>
75              </div>
76      </body>
```

Codice 5.4: index.html (sistema distribuito)
Questo tratto di codice riporta il corpo dell'interfaccia web distribuita, dove il click di ogni bottone rimanda ad una pagina web dedicata.

Il Backend dell'applicazione web è invece sviluppato in linguaggio Java ed è formato da quattro pezzi di codice. I primi tre rappresentano una pagina web a cui viene reindirizzato l'utente una volta scelta la destinazione desiderata: essi sono Position1.java, Position2.java e Position3.java e sono codici praticamente identici. Per questo, viene descritto soltanto uno di essi (vedi Codice 5.5).

5. Implementazione del sistema distribuito

Una volta collegato alla base di dati implementata nell'infrastruttura Google Cloud, il codice inserisce una nuova riga nella tabella *positions* del database grazie alla libreria ***java.sql*** che permette la connessione tra il Backend Java ed il Cloud. In base alla scelta dell'utente ed il relativo reindirizzamento della pagina web, la stringa inserita nel database sarà "Position 1", "Position 2" o "Position 3". Si ricorda infatti che i nodi ROS leggeranno successivamente questa stringa e muoveranno di conseguenza il drone. L'aspetto dell'interfaccia descritto dal tratto di codice HTML di questo archivio viene riportato nella Figura 5.2; questa pagina vuole avere una funzione di semplice reindirizzamento e per questo presenta soltanto un breve avviso per l'utente.

```
1   package com.agridrone;
2
3   import java.io.*;
4   import java.sql.*;
5
6   import javax.servlet.annotation.WebServlet;
7   import javax.servlet.http.*;
8
9   @SuppressWarnings("serial")
10  @WebServlet(
11      name = "Position1",
12      urlPatterns = {"/position1"}
13   )
14
15  public class Position1 extends HttpServlet {
16
17
18  @Override
19    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
20
21    response.setContentType("text/html");
22      PrintWriter out = response.getWriter();
23
24      out.println("<html>");
25      out.println("<body bgcolor=\"#BAB700\">");
26      out.println("<h1>You will be redirected to the picture...</h1>");
27      out.println("</body>");
28      out.println("</html>");
29
30      try {
31
32        Class.forName("com.mysql.jdbc.Driver");
33        Connection connection = DriverManager.getConnection(
34            "jdbc:mysql://34.65.22.89/position-db?useSSL=false&allowPublicKeyRetrieval=true",
35            "user", "1234");
36        try {
37
38            String sql = "INSERT INTO positions (position) VALUES('Position 1')";
39            PreparedStatement statement = connection.prepareStatement(sql);
```

5. Implementazione del sistema distribuito

```
40              statement.executeQuery();
41
42            } finally {
43              connection.close();
44            }
45          } catch (ClassNotFoundException | SQLException e) {
46            e.printStackTrace();
47            response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
48          }
49          response.setHeader("Refresh", "120; url=/displayImage");
50
51        }
52      }
```

Codice 5.5: Position1.java

Questo codice, che fa parte del Backend dell'interfaccia web, inserisce nel database la posizione selezionata dall'utente finale.

Figura 5.2: Pagina di reindirizzamento dell'interfaccia utente web

Questa interfaccia, associata alle rispettive pagine del Backend, funge da ambiente di attesa.

5. Implementazione del sistema distribuito

L'ultimo codice che compone il Backend dell'interfaccia web è invece il file DisplayImage.java (vedi Codice 5.6). Questa pagina viene caricata automaticamente dalle pagine precedenti, associate agli altri tre file del Backend, due minuti dopo il reindirizzamento in esse. Ancora una volta, il codice connette l'applicazione web al database online tramite la libreria ***java.sql***. Qui raccoglie l'ultima istanza della tabella *pictures,* che contiene i dati binari dei frame catturati dal drone, ed utilizza un pezzo di codice HTML per mostrare questa immagine all'utente finale. L'aspetto della pagina descritto dal codice HTML inserito nell'archivio Backend viene riportato di seguito (vedi Figura 5.3).

```
1   package com.agridrone;
2
3   import java.io.*;
4   import java.nio.*;
5   import java.sql.*;
6   import java.util.UUID;
7   import javax.servlet.annotation.WebServlet;
8   import javax.servlet.http.*;
9
10  @SuppressWarnings("serial")
11  @WebServlet(
12      name = "DisplayImage",
13      urlPatterns = {"/displayImage"}
14   )
15  public class DisplayImage extends HttpServlet {
16      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
17          try {
18              Class.forName("com.mysql.jdbc.Driver");
19              Connection connection = DriverManager.getConnection(
20                      "jdbc:mysql://34.65.22.89/position-db?useSSL=false&allowPublicKeyRetrieval=true",
21                      "user", "1234");
22              String sql = "SELECT picture FROM pictures WHERE id = LAST_INSERT_ID()";
23              PreparedStatement statement = connection.prepareStatement(sql);
24              ResultSet result = statement.executeQuery();
25              if (result.next()) {
26                  String filename = UUID.randomUUID().toString() + ".jpg";
27                  Path tempFilePath = Files.createTempFile("image", filename);
28                  try (InputStream inputStream = result.getBinaryStream("picture")) {
29                      Files.copy(inputStream, tempFilePath, StandardCopyOption.REPLACE_EXISTING);
30                  }
31                  response.setContentType("text/html");
32                  String imageUrl = request.getContextPath() + "/displayImage?filename=" + filename;
33                  PrintWriter out = response.getWriter();
34                  out.println("<html>");
35                  out.println("<body bgcolor=\"#BAB700\">");
36                  out.println("<img src='" + imageUrl + "' alt='Image'>\"");
37                  out.println("</body>");
38                  out.println("</html>");
39
40                  Files.deleteIfExists(tempFilePath);
```

5. Implementazione del sistema distribuito

```
41
42                  } else {
43                      response.sendError(HttpServletResponse.SC_NOT_FOUND);
44                  }
45                  connection.close();
46              } catch (ClassNotFoundException | SQLException e) {
47                  e.printStackTrace();
48                  response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
49              }
50          }
51      }
```

Codice 5.6: DisplayImage.java

Il codice rappresenta il Backend dell'ultima pagina web per l'utilizzo del servizio di monitoraggio; essa permette la visualizzazione dell'ultima immagine caricata nel database Cloud.

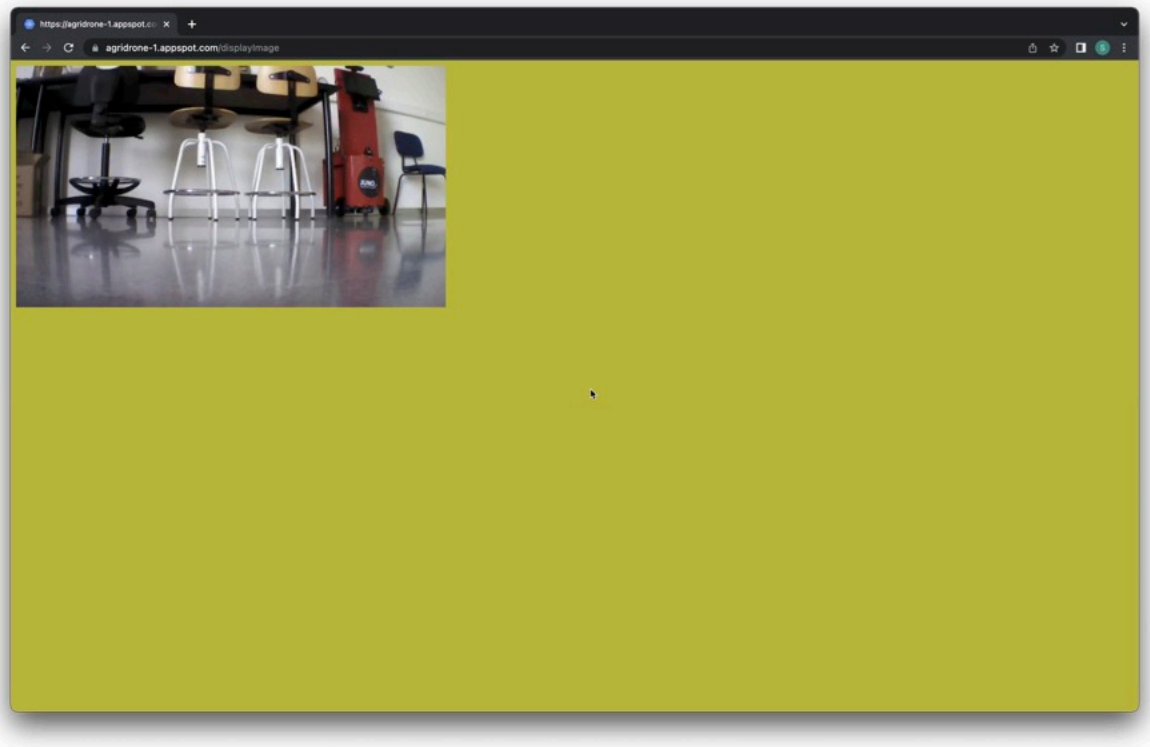

Figura 5.3: Pagina di visualizzazione dell'interfaccia utente web

L'interfaccia, associata al codice Backend appena osservato, espone all'utente lo stato della posizione selezionata attraverso la fotografia scattata dal drone.

L'intero progetto per l'interfaccia web è stato sviluppato utilizzando l'IDE Eclipse che ha permesso, con l'utilizzo del plugin chiamato Google Cloud Platform, di distribuire l'applicazione nel Cloud tramite Google App Engine.

5.3 Backend e Frontend dell'applicazione Android

Analogamente all'analisi condotta per l'interfaccia web, si procederà ad esaminare l'interfaccia mobile. Anche in questo caso, l'applicazione Android presenta una parte Frontend ed una Backend. Il Frontend è composto di tre file in linguaggio XML che descrivono le tre facciate presentate all'utente che utilizza l'applicazione per il servizio. Essendo lo script XML di queste interfacce un codice a puro scopo stilistico, viene qui mostrato soltanto il loro aspetto.

L'interfaccia iniziale activity_main.xml propone all'utente tre bottoni, con le rispettive tre posizioni a cui inviare il drone, tra cui scegliere (vedi Figura 5.4).

La seconda interfaccia position.xml, a cui l'utente viene rimandato una volta che preme uno dei tre bottoni, semplicemente rappresenta una breve facciata di attesa (vedi Figura 5.5).

Infine, l'ultima interfaccia image.xml mostra all'utente la fotografia scattata dal drone (vedi Figura 5.6).

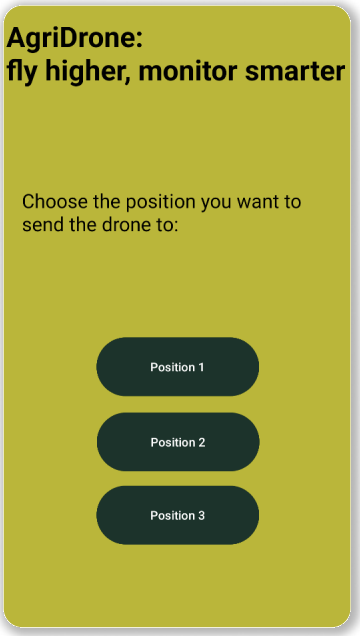

Figura 5.4: Pagina iniziale dell'interfaccia Android
Nella schermata iniziale l'utente può scegliere a quale posizione indirizzare il drone.

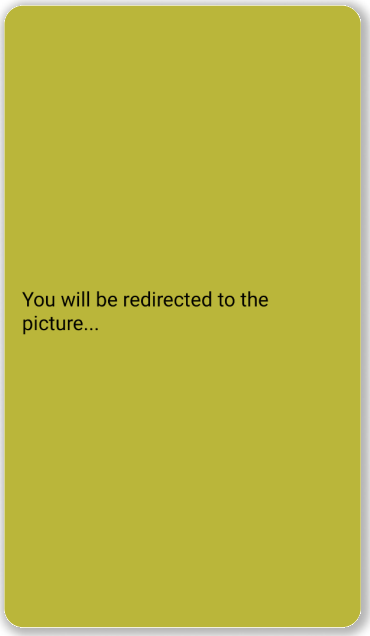

Figura 5.5: Pagina di reindirizzamento dell'interfaccia Android
La schermata rappresenta un portale per l'attesa del frame.

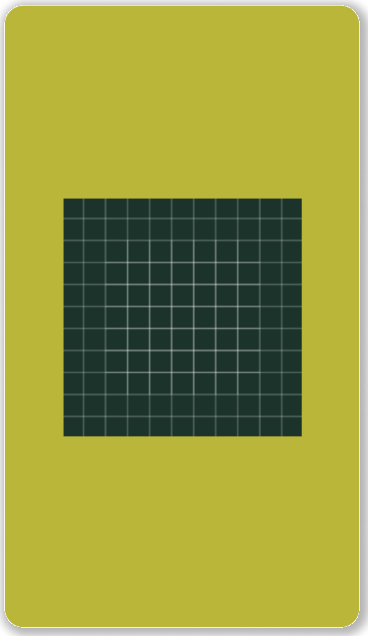

Figura 5.6: Pagina di visualizzazione dell'interfaccia Android
La schermata finale permette la visualizzazione del frame.

5. Implementazione del sistema distribuito

Si passa ora al Backend dell'applicazione, scritto in linguaggio Java, il quale è molto similare a quello sviluppato per l'applicazione web. Per prima cosa viene analizzato il codice Backend dell'activity principale, associato alla facciata iniziale: MainActivity.java (vedi Codice 5.7). In questo file viene semplicemente descritto il reindirizzamento alle activity associate alle posizioni di destinazione del drone dopo aver premuto il relativo bottone.

```java
package com.example.agridrone;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button1 = findViewById(R.id.button1);
        Button button2 = findViewById(R.id.button2);
        Button button3 = findViewById(R.id.button3);
        button1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Position1Activity.class);
                startActivity(intent);
            }
        });
        button2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Position2Activity.class);
                startActivity(intent);
            }
        });
        button3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Position3Activity.class);
                startActivity(intent);
            }
        });
    }
}
```

Codice 5.7: MainActivity.java
Il codice implementa il Backend associato alla schermata iniziale dell'applicazione Android, dove il click di ogni bottone rimanda rispettivamente ad un'altra activity.

5. Implementazione del sistema distribuito

I tre codici Backend successivi, Position1Activity.java,Position2Activity.java e Position3Activity.java, differiscono tra loro soltanto per la stringa indicante la posizione. Per questo, viene studiato solo uno di essi (vedi Codice 5.8). Tutti e tre i codici vengono associati alla stessa interfaccia position.xml. Il codice Backend connette l'applicazione Android al database ***MySQL*** di Google Cloud utilizzando la libreria ***java.sql***. Questo permette di inserire nella tabella *positions* una nuova istanza contenente la stringa relativa alla posizione scelta. Il codice reindirizza poi l'utente alla pagina successiva, permettendo di visualizzare l'immagine, con un delay di due minuti.

```java
package com.example.agridrone;

import android.os.*;
import android.content.Intent;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Position1Activity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.position);

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection connection = DriverManager.getConnection(
                    "jdbc:mysql://34.65.22.89/position-db?useSSL=false&allowPublicKeyRetrieval=true",
                    "user", "1234");
            try {
                String sql = "INSERT INTO positions (position) VALUES('Position 1')";
                PreparedStatement statement = connection.prepareStatement(sql);
                statement.executeQuery();
            } finally {
                connection.close();
            }
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }

        Handler handler=new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(Position1Activity.this, ImageActivity.class);
```

```
41                  startActivity(intent);
42              }
43          },120000);
44      }
45  }
```

Codice 5.8: Position1Activity.java
Il codice rappresenta l'activity associata alla pagina di reindirizzamento dell'applicazione Android e implementa il caricamento della posizione scelta dall'utente nel database Cloud.

L'ultimo file, ImageActivity.java (vedi Codice 5.9), è associato all'ultima interfaccia image.xml. Qui l'applicazione si connette nuovamente al database per mezzo della libreria ***java.sql*** e recupera l'ultima immagine caricata nella tabella *pictures*. Questa immagine, in formato di bit, viene rappresentata attraverso lo strumento ImageView dell'interfaccia.

```
1   package com.example.agridrone;
2
3   import android.os.*;
4   import android.util.Log;
5   import androidx.appcompat.app.AppCompatActivity;
6   import java.sql.*;
7   import android.graphics.*;
8   import android.widget.ImageView;
9
10  public class ImageActivity extends AppCompatActivity {
11
12      private ImageView imageView;
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.image);
18          ImageView imgbox = (ImageView) findViewById(R.id.imageView);
19
20          try {
21              Class.forName("com.mysql.jdbc.Driver");
22              Connection connection = DriverManager.getConnection(
23                      "jdbc:mysql://34.65.22.89/position-db?useSSL=false&allowPublicKeyRetrieval=true",
24                      "user", "1234");
25
26              try {
27                  String sql = "SELECT picture FROM pictures WHERE id = LAST_INSERT_ID()";
28                  PreparedStatement statement = connection.prepareStatement(sql);
29                  ResultSet result = statement.executeQuery();
30                  byte[] imgByte = null;
```

```
30                byte[] imgByte = null;
31                if (result.next()) {
32
33                    imgByte = result.getBytes("picture");
34                    Bitmap bmp = BitmapFactory.decodeByteArray(imgByte, 0, imgByte.length);
35                    imgbox.setImageBitmap(bmp);
36
37                }
38            }finally{
39                connection.close();
40            }
41            } catch (ClassNotFoundException | SQLException e) {
42            e.printStackTrace();
43            }
44        }
45    }
```

Codice 5.9: ImageActivity.java
L'ultimo codice Backend, associato alla pagina di visualizzazione dell'applicazione, permette all'utente di osservare il frame catturato dal drone nella posizione precedentemente selezionata.

L'intero progetto per l'applicazione Android è stato implementato con l'utilizzo di Android Studio, che ha permesso lo sviluppo e la simulazione dell'interfaccia.

5.4 Database MySQL nel Cloud

Come già accennato durante lo studio dei codici, i dati relativi alle posizioni selezionate dall'utente finale ed alle immagini catturate dal drone vengono immagazzinati in un database ***MySQL*** implementato nel Cloud. Di conseguenza, successivamente alla creazione dell'istanza *agridrone-mysql* in ***Google Cloud Platform SQL*** e alll'aggiunta all'interno di essa di un database chiamato *positions-db* e di un nuovo utente chiamato *user* per consentire l'accesso ai dati, sono finalmente state create le tabelle necessarie. Vengono quindi allegati i comandi di creazione delle tabelle (vedi Codice 5.10).

La prima tabella, *positions*, presenta tre attributi. In primo luogo, un numero identificativo che si autoincrementa ogni volta che viene inserita un'istanza. Successivamente si nota l'attributo position che contiene la stringa indicante la posizione desiderata. Infine, l'attributo time introduce autonomamente la data e l'ora dell'inserimento dell'istanza. Questi elementi permettono di distinguere ogni destinazione scelta dall'utente. Si può osservare nella Figura 5.7 come vengono memorizzate le diverse posizioni.

La seconda tabella, *pictures*, registra invece due attributi: il primo è ancora una volta un numero identificativo che viene autoincrementato, mentre il secondo è un campo di tipo BLOB (Binary Large Object) utilizzato per immagazzinare i dati binari delle immagini. Visto il tipo di dato e la conseguente rappresentazione, non comprensibile se non convertita, non viene mostrata la tabella con le sue istanze.

```
mysql> CREATE TABLE positions (
    -> id INT NOT NULL AUTO_INCREMENT,
    -> position VARCHAR(250) NOT NULL,
    -> time TIMESTAMP DEFAULT NOW(),
    -> PRIMARY KEY(id)
    -> );

mysql> CREATE TABLE pictures(
    -> id INT NOT NULL AUTO_INCREMENT,
    -> picture BLOB,
    -> PRIMARY KEY(id)
    -> );
```

Codice 5.10: Comandi di creazione in MySQL delle tabelle positions e pictures

id	position	time
1	Position 1	2023-05-30 19:05:02
2	Position 3	2023-05-30 19:09:31
3	Position 3	2023-06-01 13:43:24
4	Position 2	2023-06-01 13:46:36
5	Position 1	2023-06-01 15:31:24
6	Position 2	2023-06-01 15:44:08
7	Position 3	2023-06-01 15:47:47
8	Position 1	2023-06-02 09:41:07
9	Position 2	2023-06-02 09:50:43
10	Position 3	2023-06-02 09:54:28
11	Position 3	2023-06-02 09:57:22

Figura 5.7: Visualizzazione dei dati immagazzinati nella tabella positions

Applicazione del sistema all'agricoltura

6.1 Sistema di monitoraggio a servizio dell'agricoltura

Il sistema di monitoraggio distribuito nel Cloud appena descritto, basato sull'utilizzo del drone come vettore di sorveglianza, può trovare una preziosa applicazione nell'ambito dell'agricoltura di precisione. Il servizio di monitoraggio può infatti essere utilizzato dagli agricoltori, che rappresentano l'utente finale del sistema, per vigilare sulla sicurezza delle loro proprietà attraverso la continua ispezione delle aree coltivate. Grazie a questa soluzione affidabile ed efficiente, gli agricoltori possono quindi mantenere il controllo dei loro terreni agricoli, prevenire potenziali minacce ed adottare misure immediate per proteggere le loro colture; il tutto garantendo una costanza ed una continuità nel tempo del servizio.

Numerose imprese hanno già intrapreso iniziative nel campo della sicurezza mediante l'impiego dei droni, un settore che mostra notevoli prospettive di crescita e sviluppo. Un esempio tangibile di questa tendenza emergente è illustrato nella Figura 6.1.

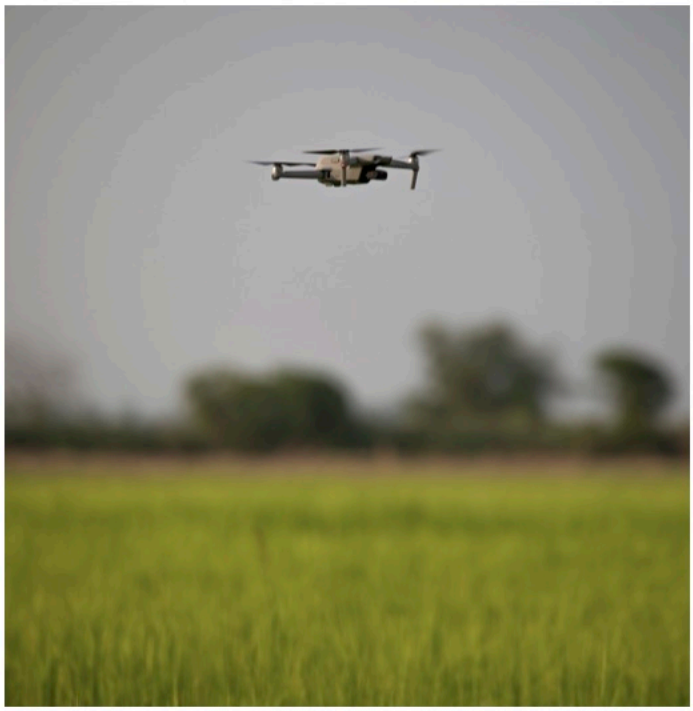

Figura 6.1: Utilizzo di un drone per il monitoraggio e la sorveglianza

In questa figura si può notare un drone DJI modello Mavic Pro 2. DJI è un'azienda leader nel settore dei droni, con sede in Cina, che negli ultimi anni ha continuato ad innovare ed ampliare la propria gamma di prodotti con un focus sulle soluzioni per l'agricoltura di precisione.[8]

Conclusioni e progetti futuri

7.1 Riassunto dei risultati ottenuti

Riassumendo, questo progetto è riuscito ad implementare con successo un sistema di vigilanza distribuito nel Cloud che utilizza un drone come vettore di controllo.

L'intero sistema è stato soggetto a test approfonditi, sia a livello locale che a livello Internet, ed ha dimostrato piena funzionalità ed affidabilità. Le interfacce web e mobile, esposte a utenti terzi per testarne l'utilizzabilità, sono riuscite a garantire un'esperienza finale intuitiva e soddisfacente grazie alla loro chiarezza e semplicità. Le prove di spostamento del drone hanno svolto un ruolo cruciale nel calibrare le tempistiche di memorizzazione dei dati e di risposta delle interfacce. Questo ha permesso di determinare il tempo necessario per l'acquisizione e l'elaborazione dei dati e di conseguenza di perfezionare l'esperienza dell'utente finale; l'obiettivo è stato infatti quello di raggiungere l'equilibrio ideale tra tempo di azione del servizio offerto dal sistema e tempo di attesa accettabile per l'utente per garantire un'esperienza fluida e reattiva.

Nel complesso, il progetto ha raggiunto con successo gli obiettivi prefissati, offrendo un sistema di vigilanza avanzato e potente per garantire la sicurezza dei campi agricoli.

7.2 Limiti e prospettive future del sistema

Nonostante i risultati positivi ottenuti, è importante riconoscere alcuni limiti e considerare le prospettive future per il sistema di vigilanza distribuito nel Cloud.

Uno dei principali limiti di questo progetto riguarda la variabilità di movimento del drone. Attualmente, lo spostamento del drone da una base all'altra viene effettuato attraverso una serie di comandi predefiniti che non

garantiscono risultati consistenti ad ogni lancio del servizio. Questo risulta in una scarsa precisione nel posizionamento del drone. Per superare questa limitazione, una soluzione potrebbe essere quella di utilizzare il modulo GPS integrato nel drone perché lo spostamento non venga più definito da una serie di movimenti ma da una localizzazione precisa della destinazione. L'uso del GPS migliorerebbe notevolmente la precisione e l'affidabilità del posizionamento del drone, consentendo una maggiore accuratezza nella sorveglianza ed una maggiore efficienza nell'esecuzione dell'attività.

Un ulteriore vincolo di questo sistema è rappresentato dall'archiviazione delle immagini nel database utilizzando il tipo BLOB. Questo tipo di memorizzazione dei dati potrebbe infatti risultare pesante in termini di risorse di sistema e spazio di archiviazione se questa applicazione passasse da un semplice progetto ad un servizio di mercato, aumentando quindi considerevolmente i propri numeri. Per affrontare questo limite e garantire un'efficiente gestione delle risorse, si potrebbe valutare l'utilizzo di un sistema di archiviazione dedicato per i file immagine, come ad esempio un filesystem separato o un servizio di archiviazione appositamente progettato per gestire grandi volumi di file.

Per quanto riguarda le prospettive future, il sistema attuale non effettua alcun tipo di ispezione sulle foto che raccoglie. Integrando algoritmi di visione artificiale e machine learning, già ampiamente disponibili sul mercato, si potrebbe espandere questo servizio oltre la semplice vigilanza. Lo studio delle immagini raccolte permetterebbe infatti l'utilizzo del sistema per svariate applicazioni di analisi degli appezzamenti e delle piante stesse. Ad esempio, sarebbe possibile rilevare eventuali malattie o danni alle piante, valutare il livello di maturazione dei frutti o creare una mappa dettagliata dei campi, identificando le diverse colture e analizzando la copertura vegetale.

In definitiva, il progetto rappresenta un solido punto di partenza per ulteriori sviluppi e miglioramenti. Grazie alla sua architettura distribuita nel Cloud, alla facilità d'uso delle interfacce utente e alle ricche prospettive future, il sistema si propone come una soluzione promettente per le diverse esigenze del settore agricolo.

Per concludere, viene riportato il collegamento ai video dimostrativi del funzionamento del servizio.

(ITA)
https://youtu.be/LYtO1yGHS5s

(ENG)
https://youtu.be/TpH28xZ9XDo

Bibliografia e riferimenti

[1] Mitchell C. Hunter and others, Agriculture in 2050: Recalibrating Targets for Sustainable Intensification, BioScience, Volume 67, Issue 4, April 2017, Pages 386–391, https://doi.org/10.1093/biosci/bix010

[2] PIERCE, Francis J.; NOWAK, Peter. Aspects of precision agriculture. Advances in agronomy, 1999, 67: 1-85.

[3] BASSO, Bruno, et al. Analyzing the effects of climate variability on spatial pattern of yield in a maize–wheat–soybean rotation. European Journal of Agronomy, 2007, 26.2: 82-91.

[4] BENVENUTI, Lorenzo; SARTORI, Luigi. Agricoltura di precisione: applicazioni in orticoltura. 2008.

[5] PATHAK, H., et al. Use of drones in agriculture: Potentials, Problems and Policy Needs. ICAR-National Institute of Abiotic Stress Management, 2020, 4-5.

[6] MADDIKUNTA, Praveen Kumar Reddy, et al. Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges. IEEE Sensors Journal, 2021, 21.16: 17608-17619.

[7] BOUAFIF, Hana; KAMOUN, Faouzi; IQBAL, Farkhund. Towards a better understanding of drone forensics: A case study of parrot AR drone 2.0. International Journal of Digital Crime and Forensics (IJDCF), 2020, 12.1: 35-57.

[8] DJI - https://www.dji.com/it

[9] PARROT - www.parrot.com

[10] ROS - http://wiki.ros.org

[11] LEE, Jihoon. Web applications for robots using rosbridge. Brown University, 2012.

[12] BOSS, Greg, et al. Cloud computing. IBM white paper, 2007, 321: 224-231.

[13] VARIA, Jinesh. Cloud architectures. White Paper of Amazon, jineshvaria. s3. amazonaws. com/public/cloudarchitectures-varia. pdf, 2008, 16.

[14] ZAHARIEV, Alexander. Google app engine. Helsinki University of Technology, 2009, 1-5.

[15] GOOGLE CLOUD PLATFORM - https://cloud.google.com

Dedicato alla mia famiglia
e soprattutto a te,
mamma