Parcheggio Intelligente

Nunzio D'Amore[1]*, Emanuele Lattanzi[2]

Sommario
Il presente lavoro tratta la realizzazione di un modellino raffigurante un parcheggio intelligente. Il parcheggio sarà dotato di vari sensori e attuatori per garantire sicurezza, comfort e personalizzazione. La struttura si basa su una rete di dispositivi interconnessi, in grado di comunicare e collaborare tra loro. La disponibilità di posti auto viene verificata tramite sensori di prossimità o un modello di Computer Vision. Tramite un'app dedicata, l'utente può monitorare in tempo reale la disponibilità dei posti auto, le condizioni ambientali all'interno del parcheggio e inviare comandi da remoto, migliorando così l'efficienza e la fruibilità del servizio.

Keywords
Parcheggio intelligente — Sensori — Computer Vision — App dedicata

[1] *Laurea Magistrale in Informatica e Innovazione Digitale, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
[2] *Docente di Programmazione per l'Internet of Things, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
***Corresponding author**: n.damore@campus.uniurb.it

Introduzione

In un mondo sempre più connesso e automatizzato, anche le soluzioni per la mobilità urbana stanno evolvendo verso sistemi intelligenti e personalizzati. Il parcheggio, spesso considerato un semplice spazio fisico, può diventare un elemento chiave per migliorare l'esperienza dell'utente e l'efficienza dei trasporti.

Questo progetto propone la realizzazione di un modellino di parcheggio intelligente, capace di integrare sensori, attuatori e connettività per offrire funzionalità avanzate in termini di sicurezza, comfort e controllo remoto.

Il progetto metterà in comunicazione micro controllori, server, smartphone e modelli di computer vision applicati ad un sistema di videosorveglianza, per indicare all'utente la disponibilità di posti auto e permettere la personalizzazione dell'ambiente, tenendo alto il livello di comfort e sicurezza.

Infatti, il parcheggio ha sistemi di allarme e di spegnimento automatico degli incendi. Inoltre, riesce a percepire la qualità dell'aria, attivando sistemi di ventilazione se necessario.

Un'app mobile dedicata consente la gestione del sistema in tempo reale, dimostrando come la tecnologia possa trasformare anche le operazioni più quotidiane in esperienze smart e su misura.

La comunicazione si baserà su un broker MQTT [1] che permetterà all'applicazione e la struttura del parcheggio di comunicare in tempo reale, riuscendo a trasmettere i comandi e lo stato della struttura.

1. Metodi

Il progetto vuole creare un modellino che sia quanto più fedele ad una reale applicazione di un parcheggio intelligente, che offra agli utenti una esperienza piacevole ed interattiva. Vuole, inoltre, garantire sicurezza tramite prevenzione e contenimento dei pericoli. In questa sezione descriveremo tutte le decisioni prese, considerando questi obiettivi, durante la fase di progettazione. Discuteremo i dispositivi, sensori ed attuatori coinvolti e descriveremo, poi, come questi sono tra loro interconnessi e quali sono le fasi di comunicazione.

1.1 Infrastruttura

Il parcheggio sarà diviso in due piani, poiché si vuole mostrare come l'unione di due sistemi possano collaborare allo stesso scopo. Ogni piano potrà ospitare un massimo di tre vetture. Al primo piano ci saranno sensori di prossimità, che permetteranno di rilevare la presenza di vetture. Al secondo piano ci sarà un modello di computer vision, che analizzerà le immagini del sistema di videosorveglianza.

Ogni piano è dotato di un sistema di ventilazione, che permetterà un ricircolo d'aria nel caso in cui la qualità rilevata sia scarsa.

Il parcheggio sarà dotato di un sistema di illuminazione RGB, che può essere controllata automaticamente. Inoltre, l'illuminazione potrà essere controllata da un addetto tramite l'apposita app.

Sarà presente un sistema di aria condizionata, con cui il sistema potrà garantire il mantenimento della temperatura desiderata. Anche in questo caso l'addetto può contollare il sistema dall'app.

Ci sarà, poi, un sistema antincendio tramite cui sarà in grado di rilevare gli incendi e di reagire tramite allarmi sonori e visivi, e successivamente tramite un sistema di spegnimento automatico.

Figura 1. Infrastruttura

1.2 Dispositivi

Come anticipato, diversi dispositivi verranno messi in comunicazione. I quattro principali sono: un microcontrollore **Arduino Mega 2560** [2], incaricato della gestione dei sensori e degli attuatori presenti nel parcheggio; un **sistema di videosorveglianza**; un **server locale** con accesso a Internet, che svolge una doppia funzione, fungendo sia da gateway per l'Arduino, sia da unità di elaborazione per l'analisi delle immagini provenienti dalla videosorveglianza con lo scopo di rilevare i posti liberi nel piano superiore del parcheggio; infine, uno **smartphone** dotato di un'applicazione che consente all'utente di visualizzare la disponibilità dei posti, le condizioni ambientali (temperatura, umidità, luminosità e qualità dell'aria) e di controllare l'illuminazione e l'aria condizionata.

Questi comunicheranno poi con il **broker MQTT** in cloud, che sarà il punto centrale della comunicazione, permettendo all'app di ricevere informazioni e controllare da remoto il parcheggio.

1.3 Sensori e Attuatori

I sensori presenti sono diversi e permetteranno di controllare, oltre alla disponibilità di posti auto, parametri ambientali come umidità, temperatura, luminosità e qualità dell'aria.

Utilizzeremo il sensore **MQ-2**, in grado di rilevare la presenza di gas quali GPL, propano, metano, alcol, idrogeno, fumo, butano. Questo ci permetterà di tenere sotto controllo la qualità dell'aria attivando il sistema di areazione.

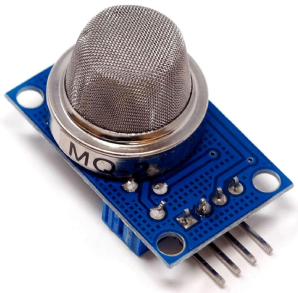

Figura 2. MQ-2

Il **sistema di ventilazione** è simulato attraverso delle ventole a 5V, presenti lateralmente su entrambi i piani. Questo verrà attivato gradualmente attraverso i pin analogici di Arduino dotati di *Power Width Modulation*. L'alimentazione delle ventole non passa per arduino, che le controllerà attraverso un transistor NPN **TIP31C**.

Il sistema di **aria condizionata**, è simulato attraverso un led che si accenderà rosso nel momento in cui il sistema viene impostato a caldo e blu quando viene impostato a freddo.

Un **sensore ad infrarossi** per la rilevazione di fiamme, permetterà di attivare un sistema di allarme sonoro e visivo e contemporaneamente un sistema automatico per la sedazione dell'incendio.

Figura 3. sensore IR

L'allarme sonoro viene simulato attraverso un **buzzer**, mentre quello visivo attraverso un **LED** rosso lampeggiante.

Figura 4. Buzzer

Figura 6. HC-SR04

Il sistema di sedazione dell'incendio verrà simulato attraverso un **nebulizzatore** posto nel retro della struttura. Il nebulizzatore ha un circuito dedicato per la riproduzione di frequenze ad ultrasuoni, che viene attivato, all'occorrenza, tramite un transistor NPN **TIP31C**.

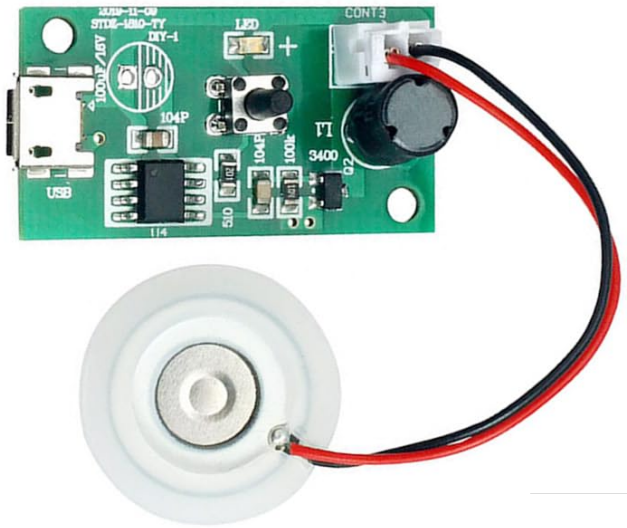

Figura 5. Nebulizzatore

Per la rilevazione di posti auto nel piano inferiore, vengono utilizzati tre sensori di prossimità ad ultrasuoni **HC-SR04**, che permettono di rilevare la presenza di veicoli.

Il sistema di videosorveglianza viene simulato attraverso attraverso uno **smartphone** che acquisirà, su richiesta, le immagini da mandare al server.

Per il controllo dei parametri ambientali, verrà utilizzato un sensore **DHT11** che permette di misurare la temperatura e l'umidità dell'ambiente.

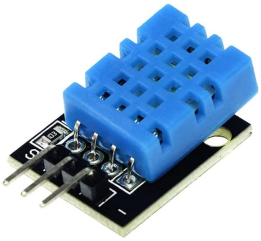

Figura 7. DHT11

Infine, il sistema di illuminazione viene implementato attraverso dei **LED RGB** che permettono di sceglierne il colore. Questi verranno gestiti tramite l'applicazione e potranno anche essere attivati e disattivati automaticamente sulla base di una **fotoresistenza**.

Figura 8. Fotoresistenza

1.4 Architettura di comunicazione

Tutti i sensori e attuatori descritti nella sezione 1.3 comunicano con Arduino, che li gestisce in maniera autonoma. A sua volta, Arduino usa la **trasmissione seriale**, per comunicare con il server locale che fa da gateway, sia per portare all'esterno lo stato dei sensori, sia per ricevere eventuali comandi

dall'esterno.

Anche il sistema di videosorveglianza comunica direttamente con il server locale tramite **USB**, che ne acquisirà le immagini per poi sottoporle ad analisi attraverso il modello di computer vision.

Al centro della comunicazione, c'è un **broker MQTT** in cloud, che permette al server locale e l'app dell'utente di comunicare. In questo modo, l'utente riesce, tramite l'app, ad ottenere in tempo reale le informazioni sul parcheggio e ad impartire comandi da remoto.

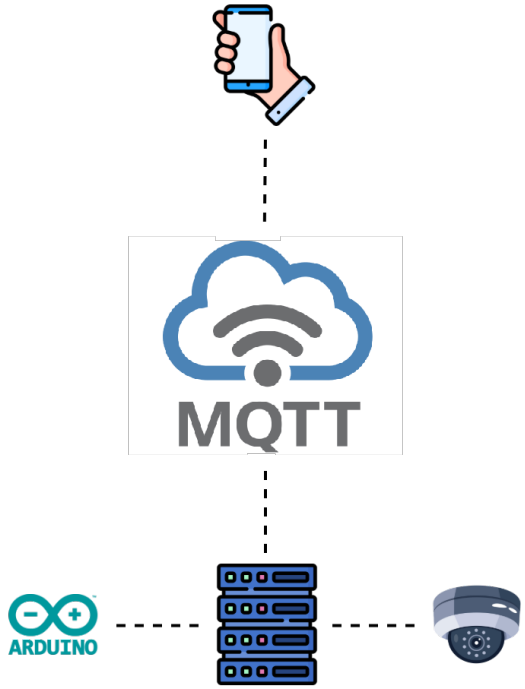

Figura 9. Architettura di comunicazione

1.5 Collegamenti hardware

In questa sezione descriveremo i collegamenti hardware tra i sensori e Arduino.

L'alimentazione è divisa in due parti, una alimenta arduino e l'altra alimenta le ventole e il nebulizzatore che richiedono più energia per funzionare, evitando di sovraccaricare arduino.

Tutti i sensori con output analogico come la fotoresistenza, il sensore di fiamma, e il sensore di qualità dell'aria, saranno collegati a pin Analog to Digital Converter che su arduino iniziano per A. In questo caso i sensori saranno collegati rispettivamente ai pin A0, A13 e A9.

La fotoresistenza e il sensore di fiamma saranno collegati a 5V e a GND passando per una resistenza ad alta precisione da 15kOhm. Tra la resistenza e il pin del sensore, viene collegato il cavo che porta al pin di lettura di arduino.

Per l'allarme sonoro e visivo verranno collegati al pin 13 il buzzer per l'allarme sonoro, mentre per l'allarme visivo verrà collegato il LED rosso al pin 12.

I sensori di prossimità hanno due pin, chiamati "trigger" (per mandare l'impulso ad ultrasuoni) ed "echo" (per la ricezione dell'impulso). Per ogni parcheggio avremo un sensore ad ultrasuoni collegato ai relativi pin di trigger e di echo. Da sinistra a destra avremo i parcheggi: "1A" collegato ai pin 50 per il trigger e 52 per l'echo; "1B" collegato ai pin 51 per il trigger e 53 per l'echo; e "1C" ai pin 46 per il trigger e 48 per l'echo.

Il sistema di ventilazione e di illuminazione saranno collegati a pin analogici che tramite un sistema di PWM riescono a modulare la tensione in uscita. Il sistema di ventilazione sarà collegato al pin 6 mentre quello di illuminazione avrà un pin per ogni colore RGB che saranno rispettivamente 2, 3 e 4.

L'illuminazione è divisa in 3 led per ogni piano. Questi led sono collegati in parallelo così che il malfunzionamento di un led non causi lo spegnimento degli altri. I due piani sono controllati insieme essendo collegati agli stessi pin.

Il sistema di aria condizionata invece, verrà collegato al pin 30 per il colore blu, e al pin 31 per il colore rosso.

Il nebulizzatore sarà controllato tramite il pin 24, con cui arduino controlla il relativo transistor. Questo quando attivato mette a massa il nebulizzatore, permettendo di attivarlo su richiesta. Il pin base viene collegato anche a GND tramite una resistenza da 1.5KOhm.

1.6 Software

Il software che gestisce il funzionamento del parcheggio intelligente è diviso in cinque parti, più il broker MQTT, che comunicano e collaborano per il funzionamento dell'intero sistema.

1.6.1 Server locale

Sul server locale sono in esecuzione **due programmi python**, uno che gestisce la comunicazione con arduino tramite porta seriale e l'altro che permette di ricevere le immagini dal sistema di videosorveglianza, per poi elaborarle con un modello di computer vision.

Gateway arduino Il programma consente al server locale di comunicare con arduino, basando la comunicazione sulla porta seriale con 9600 baud. Questo, inizializza la comunicazione con arduino, attendendo che sia pronto per la comunicazione. Successivamente, avvia un thread che è incaricato di ricevere tutti i messaggi di arduino. Il thread principale esegue l'iscrizione al topic dei comandi provenienti dagli utenti e,

contemporaneamente esegue richieste periodiche ad arduino per ottenere tutte le informazioni ambientali. Ricevuti i comandi dal broker MQTT, procede ad inviarli ad arduino che provvederà all'esecuzione.

Ogni comando inviato ad arduino viene accostato con un identificativo univoco, che arduino a sua volta accosta al messaggio di risposta. In questo modo un sistema ad eventi può essere implementato in modo che il thread di ricezione possa sapere come gestire correttamente il messaggio di risposta, consegnandolo ai dovuti listener che ne gestiscono il funzionamento.

Contemporaneamente, come dicevamo, ricevute le informazioni da arduino, queste vengono comunicate al broker MQTT in modalità "retain" all'apposito topic per lo stato. In questo modo, anche gli utenti che si iscrivono successivamente all'ultimo messaggio inviato, riescono a ricevere l'ultimo stato registrato. Questi messaggi vengono inviati con un QoS impostato su "at least once", per fare in modo che la ricezione dello stato sia garantita agli utenti, anche se più volte.

In questo modo, il software permette ad arduino di comunicare con l'esterno, funzionando di fatto da gateway.

Codice:
https://github.com/NunzioDA/IoTArduinoGateway

Analisi videosorveglianza Il software di analisi delle immagini preleva periodicamente un'immagine dal sistema di videosorveglianza e la sottopone al modello di computer vision, il quale rileva l'eventuale presenza di veicoli.

Prima dell'analisi, le immagini sono opportunamente suddivise in tre parti, isolando i parcheggi presenti nel piano superiore. Successivamente, il modello adatta le immagini alla rete sottostante, permettendone l'analisi. L'adattamento consiste in un resizing dell'immagine per assicurarci che questa rispetti i vincoli imposti dalla rete e, successivamente, una normalizzazione.

Il software permette, inoltre, di salvare le parti in cui l'immagine è divisa. Questa funzione è stata utilizzata in fase di addestramento, e può essere utilizzata per salvare quelle situazioni in cui il modello sbaglia, per includerle in un addestramento successivo.

Codice:
https://github.com/NunzioDA/IoTMLModel

1.6.2 Struttura del modello CNN

La rete neurale è una rete convoluzionale progettata per elaborare immagini a tre canali(RGB), con una dimensione di ingresso pari a 72x42 pixel. La rete è composta da quattro strati convoluzionali consecutivi, intervallati con layer di batch normalization. Il primo strato convoluzionale riceve l'immagine in ingresso e applica 32 filtri di dimensione 3x3, mantenendo invariate le dimensioni spaziali grazie a un padding di 1. Il secondo strato prende in ingresso i 32 canali prodotti dal precedente e applica 64 filtri, anch'essi 3x3 con padding 1. Successivamente, il terzo strato convoluzionale elabora i 64 canali generati applicando 128 filtri di dimensione 3x3, ancora con padding 1 per conservare le dimensioni. Il quarto prende i 128 canali e applica 256 filtri di dimensione 3x3 con padding 1.

L'output di ogni layer convoluzionale, passa per un layer di batch normalization, poi attraverso una funzione di attivazione relu e successivamente per un layer di max pooling con finestra 2x2 e stride 2 che riduce le dimensioni spaziali dell'output dimezzandole. Assumendo un'immagine di partenza 72x42, al termine dei quattro strati convoluzionali e del pooling, le dimensioni risultano ridotte, e verrà prodotto un tensore di dimensione 256x4x2.

Questo tensore viene quindi appiattito e passato a un primo strato completamente connesso che trasforma il tensore appiattito in 256 unità. Il risultato è poi elaborato da un secondo strato completamente connesso che riduce l'output a 128 unità. Infine, l'ultimo strato lo trasforma in una sola unità a cui viene applicata una funzione di attivazione sigmoidale, che comprime il valore di uscita in un intervallo compreso tra 0 e 1, rendendo la rete adatta a compiti di classificazione binaria, come la presenza o assenza di veicoli.

1.6.3 Addestramento

Come visto in precedenza il software permette di salvare le parti in cui l'immagine è divisa. Questo risulta fondamentale per la raccolta di immagini utili per addestrare il modello. Infatti, vi è stata una fase iniziale in cui sono state acquisite diverse immagini, in situazioni diverse.

Sono state acquisite immagini con luce naturale e luce artificiale, o ancora con i led della struttura accesi, spenti e con vari colori.

Sono state affrontate diverse fasi di addestramento, aggiungendo sempre più esempi per il modello in modo che riuscisse a performare bene anche in situazioni complesse, come in caso di scarsa illuminazione.

1.6.4 Arduino

Il **codice arduino**, gestisce tutti i sensori e gli attuatori, comunicando con l'esterno attraverso la porta seriale.

Il codice è strutturato in modo che ogni operazione sia non bloccante, garantendo così una comunicazione costante con l'esterno e che i comandi vengano gestiti nel più breve tempo possibile.

Arduino, controlla periodicamente la presenza di messaggi esterni, ed eventualmente ne gestisce il contenuto.

Nel messaggio i vari argomenti devono essere divisi attraverso dei punti e virgola. Tutti i messaggi iniziano con una stringa che deve essere un codice univoco, che arduino provvede ad inoltrare come primo argomento della risposta. In questo modo il ricevente riesce a capire a quale richiesta si collega la risposta. Successivamente, il contenuto dipende dal comando che si vuole impartire, aggiungendo in coda tutti i conseguenti parametri. Ad esempio per modificare il colore della luce, impartiremo il seguente comando:

$$2131241;light;color;255;79;10;$$

Quello che stiamo facendo è dire ad arduino che vogliamo impartire un comando riguardante l'illuminazione, per modificarne il colore ed impostare la componente rossa a 255 quella verde a 79 e quella blu a 10.

Arduino risponderà con il seguente messaggio, per indicare che l'operazione è andata a buon fine:

$$2131241;ok;$$

I comandi che arduino accetta dall'esterno e quindi i comandi che l'utente sarà in grado di impartire, riguardano la richiesta di informazioni, il controllo delle luci e dell'aria condizionata.

Tutti i comandi sono descritti nella tabella 1.

Comando	Argomenti	Funzione
light	color;r;g;b	permette di impostare il colore dell'illuminazione
light	on	permette di accendere la luce manualmente
light	off	permette di spegnere la luce manualmente
light	auto	permette di impostare la luce su automatico
info		permette di richiedere l'attuale stato dell'ambiente in JSON
air	temperature;t	imposta una certa temperatura target
air	warm	imposta l'aria condizionata su caldo
air	cool	imposta l'aria condizionata su freddo
air	off	spegnel'aria condizionata

Tabella 1. Comandi accettati da arduino

Nel caso del comando info, la risposta sarà una stringa JSON, in modo che sia facilmente consultabile e compatibile con altre fonti.

Nello stato, sono contenute informazioni quali: i posti auto disponibili al piano inferiore, la temperatura ambientale e target, l'umidità, qualità dell'aria, la luce rilevata e le informazioni sul sistema di illuminazione, come colore e modalità (manuale o automatico).


```
{
  "temperature": 22,
  "target_temperature": 20,
  "humidity": 35,
  "air": 90,
  "light": 464,
  "light_status": 0,
  "light_color": [
    255,
    255,
    255
  ],
  "light_mode": "Auto",
  "park": [
    1,
    1,
    0
  ]
}
```


Figura 10. Esempio risposta arduino al comando "info"

I valori letti dai sensori analogici non vengono usati direttamente, ma aggiornano il rispettivo valore tramite una media pesata. Questo permette di ridurre il rumore ed evitare cambi improvvisi dei valori ammorbidendo il segnale in ingresso. Di seguito è riportata la formula con cui il nuovo valore viene calcolato, dove $w1$ è il peso relativo al valore precedentemente calcolato, $w2$ è il peso del valore appena letto dal sensore.

$$NewValue = w1 * OldValue + w2 * SensorValue$$

Arduino è anche incaricato di gestire sistemi di sicurezza, andando quindi ad attivare il sistema di areazione in caso la qualità dell'aria non sia ottimale o di attivare il sistema antincendio.

Codice:
https://github.com/NunzioDA/IoTArduino

1.6.5 Videosorveglianza

Il sistema di videosorveglianza viene simulato attraverso uno smartphone su cui è in esecuzione un'**app flutter** che su richiesta fornisce le immagini della videocamera. La comunicazione in questo caso sarebbe dovuta avvenire tramite porta USB, ma per motivi tecnici relativi allo smartphone utilizzato e ai limiti dell'hotspot wifi in sede di esame, le immagini saranno trasferite dal sistema di videosorveglianza al server locale passando per un Web Server. Apposite API sono state create, infatti, per richiedere, mandare e recuperare le immagini. Queste sono scelte esclusivamente legate al lato pratico della simulazione in sede di esame, non sono per tanto da ritenersi scelte architetturali. L'architettura da considerarsi è quella riportata in figura 9.

Codice:
https://github.com/NunzioDA/IoTCameraApp

1.6.6 Broker MQTT

Il broker MQTT si trova in cloud, permettendo all'app e al server locale del parcheggio di comunicare. Il provider utilizzato è HiveMQ [1], che permette gratuitamente di creare un broker MQTT nel cloud. I topic impostati sono tre. Il primo topic, permette all'app dell'utente di inviare i comandi al server locale. Il secondo, permette ad arduino di comnicare il suo stato interno agli utenti. Il terzo, permette al server locale di comunicare le previsioni effettuate tramite il modello di ML all'app dell'utente.

Ogni topic inizia con il nome della struttura a cui fa riferimento per poi aggiungere la parte restante del topic. Ad esempio, nel nostro caso in cui abbiamo una sola struttura avremo i topic: "urbinoparkid/commands", "urbinoparkid/ambient/status",
"urbinoparkid/ambient/aipark".

1.6.7 App dell'utente finale

L'**app flutter dell'utente finale**, permette di ottenere tutte le informazioni utili sul parcheggio e di controllarne l'illuminazione e l'aria condizionata.

L'UI è molto semplice ed è composta da un'unica schermata principale e due dialog, di cui uno per la scelta del colore delle luci e l'altro per la scelta della temperatura target da impostare.

In alto, nella schermata principale, ci sono tutte le informazioni riguardanti l'ambiente come: luogo, temperatura, umidità, luminosità e qualità dell'aria. Subito sotto una griglia che mostra i posti d'auto liberi e occupati di entrambi i piani.

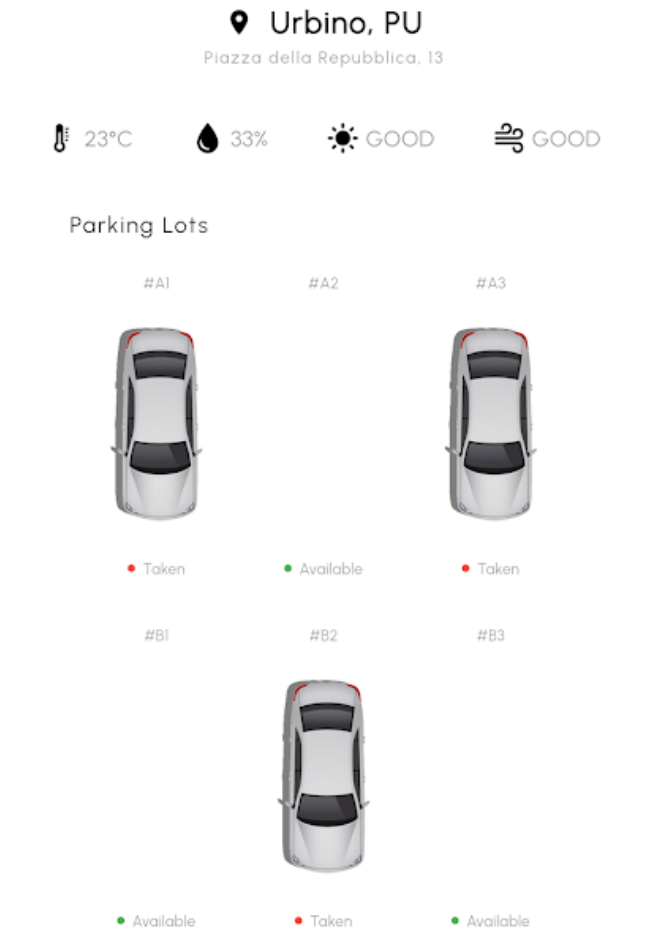

Figura 11. UI Applicazione utente finale

Ancora sotto, tutti i comandi relativi all'illuminazione, tra cui: un pulsante per cambiare il colore delle luci, uno per spegnere, uno per accendere e un pulsante per impostare la modalità automatica.

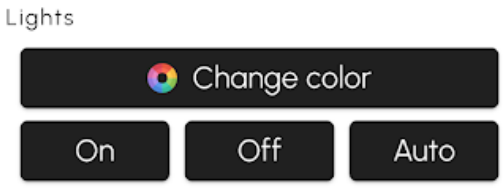

Figura 12. Controlli illuminazione

Il pulsante per cambiare il colore delle luci apre il dialog che permette di scegliere il colore desiderato. Una volta scelto il colore, l'applicazione procede inviando il comando visto precedentemente al broker MQTT che provvederà ad inoltrarlo al server locale.

Il dialog presenta un selettore di colore circolare, con un'anteprima al suo interno. L'anteprima viene inizialmente im-

postata al colore delle luci attualmente selezionato. Inoltre, nel dialog è possibile scegliere il colore bianco separatamente. Scelto il colore, può essere confermato attraverso l'apposito pulsante.

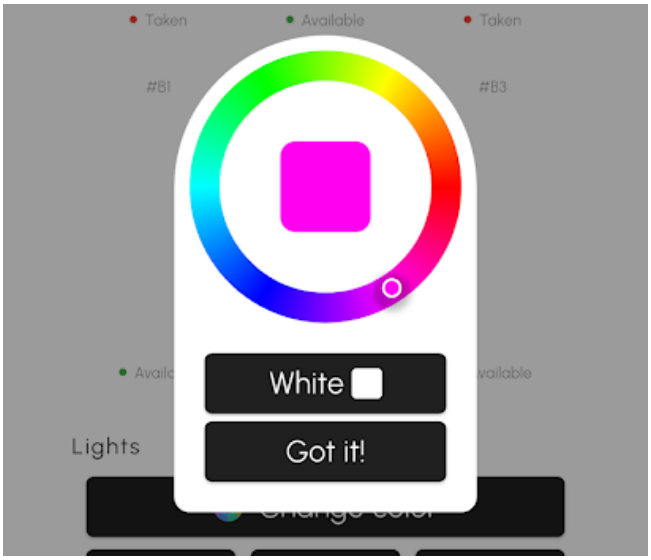

Figura 13. Dialog per la scelta del colore

Infine, è presente la sezione relativa al controllo dell'aria condizionata. Questa, permette di: accendere l'aria condizionata impostandola a caldo o freddo; spegnere l'aria condizionata; e, infine, di impostare una temperatura che il sistema cercherà di tenere in automatico.

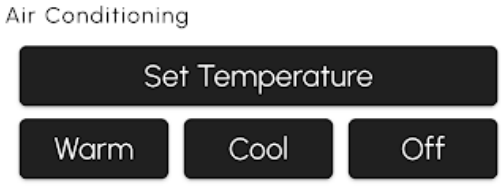

Figura 14. Controlli aria condizionata

Il pulsante che permette di impostare la modalità automatica, permette di scegliere la temperatura desiderata attraverso l'apposito dialog. Questo presenta un campo di testo per inserire la temperatura e un pulsante per effettuare la scelta.

Figura 15. Dialog per inserire la temperatura desiderata

L'app si mette in ascolto dello stato del parcheggio e delle previsioni del modello di AI iscrivendosi agli appositi topic, aggiornando il suo stato interno. Aggiornato lo stato, l'UI si adatta per rifletterne i cambiamenti.

Codice:
https://github.com/NunzioDA/IoTParkApp

2. Risultati

In questa sezione discuteremo i risultati ottenuti e mostreremo il funzionamento del sistema in diversi casi d'uso.

2.1 Segnale dei sensori

Il segnale come specificato in precedenza, viene ammorbidito attraverso un metodo di aggiornamento che usa una media pesata. Visualzziamo, come esempio, i grafici del sensore di luminosità con e senza l'elaborazione.

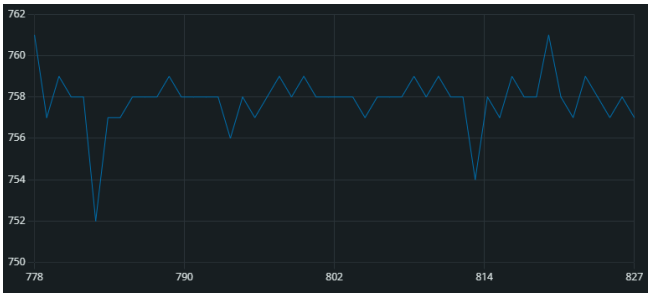

Figura 16. Andamento del valore letto dal sensore di luminosità nel tempo

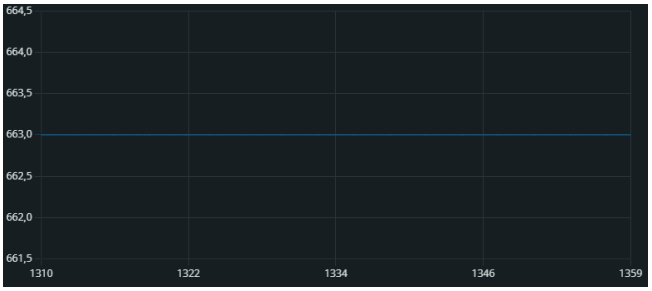

Figura 17. Andamento del valore letto dal sensore di luminosità nel tempo con elaborazione

Questo è particolarmente utile anche con i valori letti dal sensore di prossimità. Ammorbidire il segnale del sensore di prossimità ci permette di evitare sbalzi improvvisi nelle letture, migliorando l'affidabilità del valore, risucendo ad indicare con più precisione se è presente un veicolo o meno.

2.2 Parcheggi

La struttura riesce ad indicare in maniera molto affidabile la presenza di veicoli nella struttura, grazie sia ai sensori di prossimità che al sistema di videosorveglianza.

Figura 18. Screenshot del programma di analisi delle immagini

Nella figura 18, vediamo il programma in azione. Si vede come il modello riesce a distinguere i posti occupati (in verde) da quelli liberi (in rosso).

Tuttavia, entrambi hanno pregi e difetti e riescono a performare meglio dell'altro in determinate situazioni. Il modello riesce ad essere molto affidabile in situazioni normali, ma potrebbe avere difficoltà in caso di scarsa luminosità, portando a falsi negativi. Il problema può essere facilmente risolto tramite appositi sensori a visone notturna. I sensori di prossimità, d'altro canto, potrebbero riconoscere come veicoli anche oggetti o persone che momentaneamente coprono il sensore. A tal proposito, il modello, se addestrato correttamente, riesce a distinguere anche queste situazioni, come visibile in figura 19.

Figura 19. Esempio in cui il modello riesce a distinguere le macchine da persone

Il modello di machine learning ha anche un altro vantaggio: la videosorveglianza è un sistema che è normalmente già presente nelle strutture evitando costi e manutenzione aggiuntiva per l'installazione di ulteriori sensori.

Infine, in figura 20 vediamo come il modello riesca a performare bene anche in situazioni molto difficili di scarsa luminosità e con veicoli scuri.

Figura 20. Esempio in cui il modello riesce a funzionare anche in situazioni di scarsa luminosità

I sensori di prossimità si comportano molto bene, riuscendo ad indicare in modo affidabile la presenza di veicoli.

Questi però, come anticipato, riportano il problema per cui non riescono a distinguere i veicoli da qualsiasi altro oggetto.

2.3 Applicazione

L'applicazione riesce a riportare tutte le informazioni raccolte tramite i sensori garantendo comodità nell'esperienza degli utenti. Infatti, come visto in precedenza vengono visualizzate nella home, tutte le informazioni ambientali e i posti occupati. Tramite la comunicazione MQTT, riesce quasi in tempo reale a raccogliere informazioni ed inviare comandi.

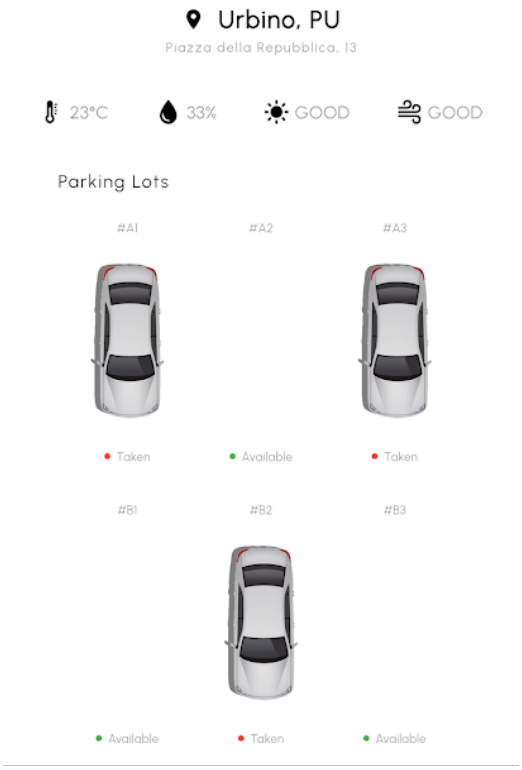

Figura 21. Esempio dati visualizzati dall'applicazione

2.4 Sistema di sicurezza

Il sistema antincendio riesce a reagire ai pericoli, attivando il sistema di spegnimento automatico degli incendi, garantendo un'esperienza sicura per gli utenti che ne usufruiscono. In figura 22, possiamo vedere il led rosso lampeggiante accendersi e l'attivazione del sistema per lo spegnimento automatico dell'incendio.

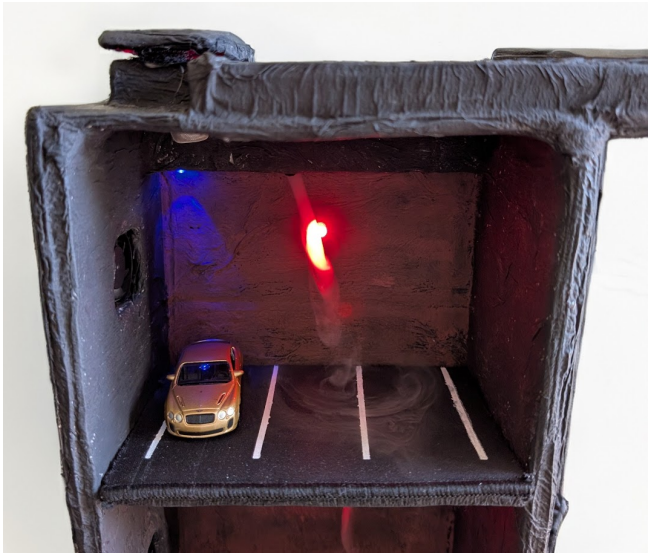

Figura 22. Esempio di attivazione del sistema antincendio

2.5 Illuminazione

Dall'app l'utente riesce a gestire l'illuminaizone in tempo reale. Riesce anche a cambiarne il colore come visibile in figura 23

Figura 23. Esempio di due colori dell'illuminazione della struttura

2.6 Aria condizonata

L'utente riesce a controllare anche l'aria condizionata dall'app, potendo impostare direttamente su caldo o freddo, oppure impostando una temperatura target che il sistema provvederà a mantenere in automatico. In figura 24 vediamo il led che simula il sistema di aria condizionata accendersi rosso per simulare aria calda e blu per simulare aria fredda.

Figura 24. Esempio di funzionamento dell'aria condizionata

3. Conclusioni

In questo progetto è stato creato un parcheggio intelligente, mettendo in comunicazione diversi dispositivi, integrando informazioni provenienti da diversi sensori e sistemi.

L'utente riesce quindi ad avere costantemente informazioni sull'ambiente del parcheggio e sui posti disponibili, migliorandone l'esperienza e l'efficienza.

Ulteriori funzioni potrebbero essere implementate, raccogliendo tutti i dati ricevuti dai sensori e i comandi impartiti usando un database. Questo permetterebbe di controllare le statistiche di utilizzo, i posti più occupati, i comandi più utilizzati, ecc; potendo intraprendere scelte volte a migliorare l'esperienza utente.

O ancora si potrebbe raccogliere lo storico dei sensori, utilizzando un time-series database, per rilevare anomalie, o inefficienze dei sistemi di prevenzione. Ad esempio, si potrebbero analizzare i dati storici del sensore della qualità dell'aria per anticipare o rilevare anomalie del sistema di ventilazione e richiedere l'attenzione dei tecnici per attività di manutenzione.

Riferimenti bibliografici

[1] HiveMQ GmbH. HiveMQ Cloud - MQTT Cloud Broker. `https://www.hivemq.com/`, 2025. Accessed: 2025-05-30.

[2] Arduino. Arduino - Official Website. `https://www.arduino.cc/`, 2025. Accessed: 2025-05-30.