



SEMAFORO INTELLIGENTE

Luca Evangelisti
Alessandro Manucci
Gianluca Milani

Data:
30 Maggio, 2025



INDICE

01 Introduzione (G)

02 Funzionalità (A) implementate

03 Architettura del sistema (G)

04 Logica semaforo intelligente (A)

05 Comunicazione Dati e DataBase (L)

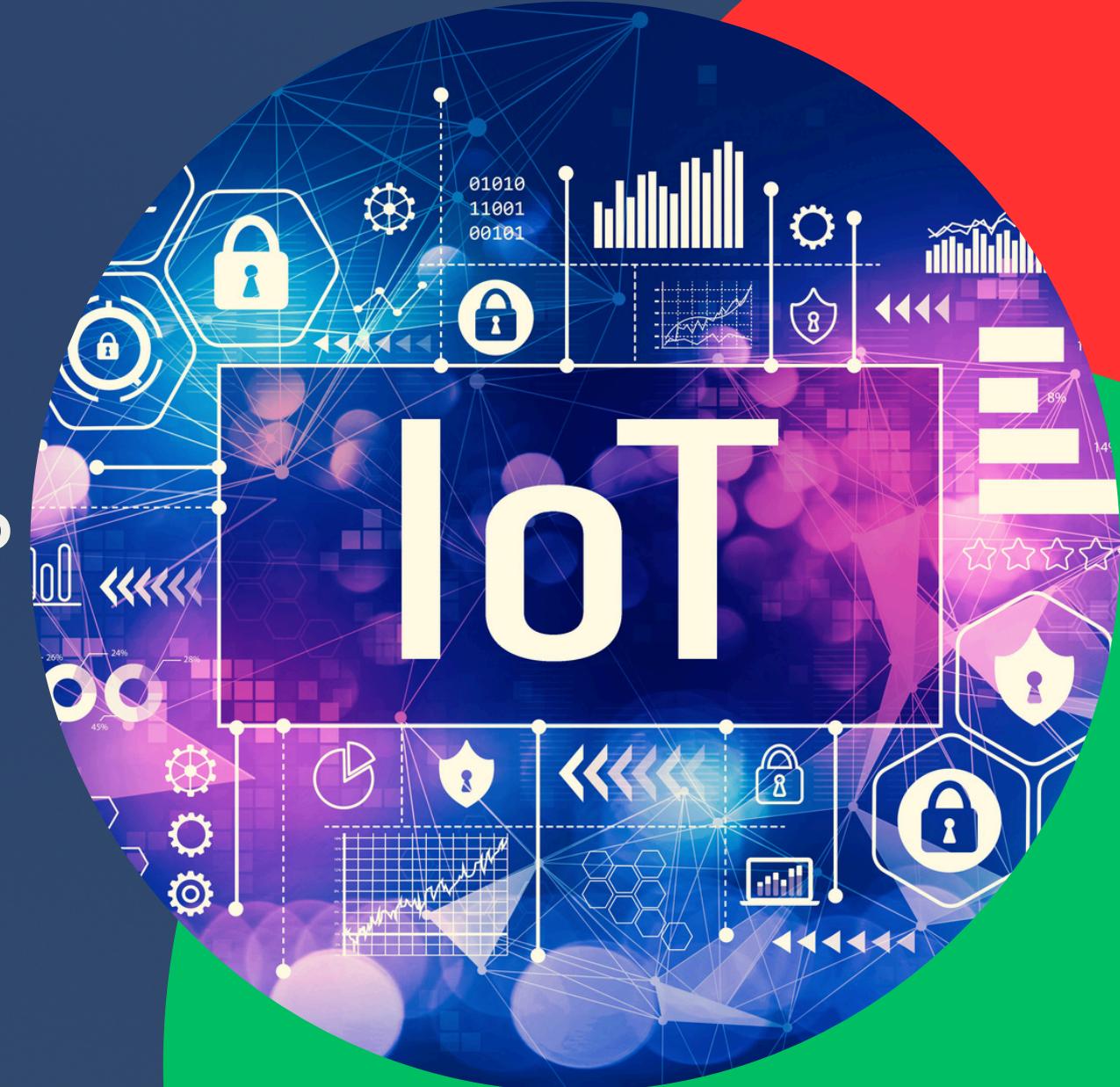
06 WebApp (G)

07 Avvio e Spegnimento del sistema (L)

08 Implementazioni future (A)

09 Schema Circuitale e Flusso Logico (A/L)

10 Percorso Progettuale (G/A)



INTRODUZIONE

Questo progetto propone un sistema IoT per il controllo intelligente di un incrocio semaforico e per il suo corretto funzionamento sono necessari:



Micro:bit

Due schede connesse a sensori e attuatori, programmate in MicroPython. La Micro:bit A gestisce i semafori in modalità adattiva, mentre la B fa da ponte radio con il PC



WebApp Flask

Consente il monitoraggio e il controllo in tempo reale del sistema



Script Python

Trasferisce i dati al database e riceve comandi



InfluxDB

Utilizzato come DataBase



Grafana

Utilizzato per la visualizzazione in grafici dei dati in tempo reale

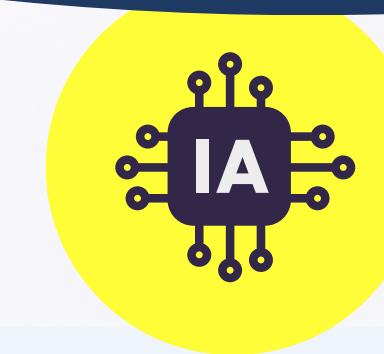


FUNZIONALITA' IMPLEMENTATE



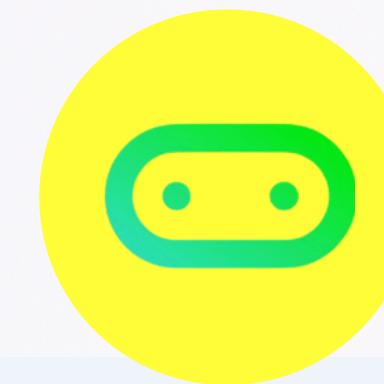
Lettura dei Sensori di Presenza Veicolare

La funzione `check_sensors()` legge i valori dei sensori analogici sulle tre vie (Nord, Est, Sud) e rileva la presenza di veicoli confrontando il segnale con una soglia (`SOGLIA`). Le variazioni di stato generano eventi trasmessi via radio.



Logica Semaforica Intelligente Dinamica

La funzione `servi(idx)` gestisce dinamicamente i semafori: assegna il verde a una via per un tempo minimo, lo estende se serve, poi passa al giallo e infine al rosso. La selezione segue un criterio round-robin, bilanciando il traffico su tutte le vie.



Comunicazione Radio **Micro:bit A → Micro:bit B**

Il Micro:bit A invia dati via radio (es. `STATE` o `SENSOR`) al Micro:bit B, che li riceve e li inoltra al PC tramite `UART`. I messaggi contengono info su semafori e sensori, codificati con timestamp e via interessata.



FUNZIONALITA' IMPLEMENTATE



Logging su InfluxDB 2.x via UART e Python

Lo script `microbit_to_influx.py` legge i dati via **UART**, li valida, corregge errori comuni, li converte nel formato InfluxDB e li scrive asincronamente nel bucket `microbit`.

Es correzione errore:
"VEDE" → "VERDE"



Dashboard Interattive in Grafana

- Le dashboard realizzate in **Grafana** includono:
- serie temporali dello stato dei semafori (rosso/giallo/verde),
 - attivazioni binarie dei sensori per ciascuna via,
 - visualizzazione in tempo reale con intervalli selezionabili (default: -5m).



Interfaccia Web Flask (WebApp)

La **WebApp Flask** offre:

- Visualizzazione semafori dinamica
- Stato sensori in tempo reale
- Integrazione **Grafana** via **iframe**
- Pannello comandi per modalità speciali
- Statistiche semaforiche dedicate



FUNZIONALITA' IMPLEMENTATE



Modalità Notturna (Night Mode)

Tramite il comando CMD;NIGHT;ON, il sistema disattiva la logica semaforica intelligente e imposta tutti i LED gialli dei semafori a lampeggio (on/off ogni 500ms), simulando la modalità notturna standard.



Modalità Emergenza (Uno alla Volta)

La modalità Emergenza permette:

- Attivazione singola tra Nord, Est e Sud
- Verde forzato per la via selezionata
- Rosso per tutti gli altri semafori
- Sospensione temporanea della logica standard



Calcolo Statistiche in Tempo Reale

Ogni 5 secondi, la WebApp aggiorna:

- Conteggio dei veicoli rilevati
- Numero di cambi semaforici per via
- Durata media del verde per semaforo

I dati provengono da InfluxDB e sono ottenuti con query Flux dinamiche



ARCHITETTURA DEL SISTEMA GENERALE

N	Livello	Descrizione
01	Edge-Dispositivo	Micro:bit A (controllo locale): legge sensori analogici (P3, P4, P10) e gestisce LED (P0,1,2, P8,12,13, P14,15,16). Invia eventi SENSOR e STATE via radio. Gestione locale di modalità Notte/Emergenza.
02	Fog-Gateway	Micro:bit B come ponte radio ↔ seriale. Lo script microbit_to_influx.py filtra, valida e inoltra i dati a InfluxDB. Espone anche comandi locali (Flask @5001) per modalità speciali.
03	Cloud – Piattaforma	InfluxDB (localhost:8086) memorizza eventi e sensori in formato time-series. Grafana crea dashboard embedded nella WebApp, con query su intervalli temporali (es. now-5m).
04	App - Frontend/API	WebApp Flask (porta 5000): espone API REST (/api/stati, /api/sensori, /api/night_mode, /stats) e interfaccia live. HTML5, CSS3, JS. Grafici Grafana, pulsanti notte/emergenza, statistiche



ARCHITETTURA DEL SISTEMA

Generale

Edge-Dispositivo

Micro:bit A – Controllo

Micro:bit A – Controllo
Sensori analogici: P3, P4, P10
LED RGB su pin digitali (P0,1,2, P8,12,13, P14,15,16)
Gestisce ciclo semaforico e invia eventi SENSOR/STATE
Gestione locale modalità Notte/Emergenza

Fog-Gateway

Micro:bit B – Gateway
Riceve dati via radio e inoltra via UART al PC
Script microbit_to_influx.py: parsing, correzione e invio su InfluxDB
Espone comandi via Flask su porta 5001
Gestione locale di comandi Notte/Emergenza

Cloud-piattaforma

InfluxDB
Time-series DB locale su porta 8086
Bucket “microbit” con metriche semafori/sensori

Grafana

Dashboard storico semafori e sensori
Aggiornamento automatico (5s)

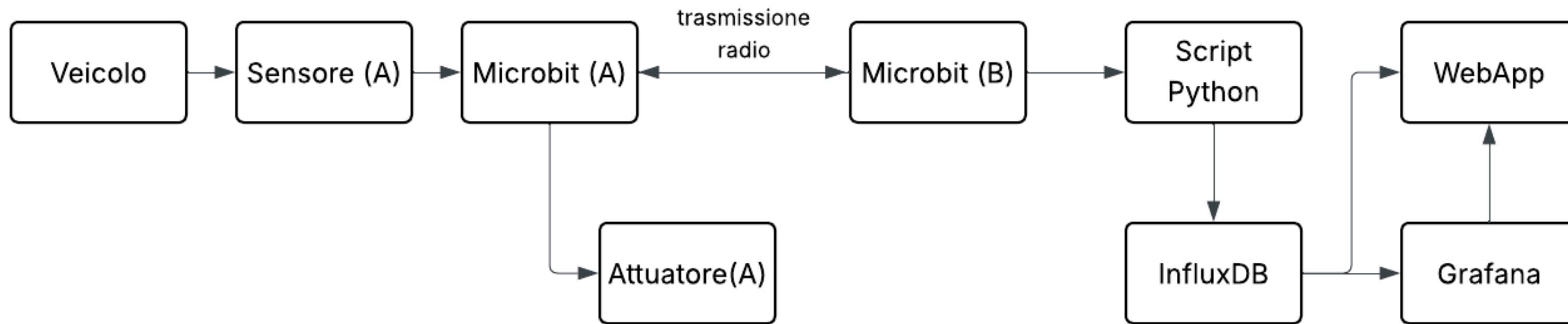
App – Frontend/API

WebApp Flask
-Espone **API REST**: /api/stati, /api/sensori, /night_mode, /emergency, /stats
-UI in HTML/CSS/JS, layout responsive
-**Grafici Grafana** embedded (1s), -statistiche aggiornate **ogni 5s**
-**Pulsanti Notte/Emergenza** con feedback visivi



ARCHITETTURA DEL SISTEMA

Generale





ARCHITETTURA DEL SISTEMA

LAYER

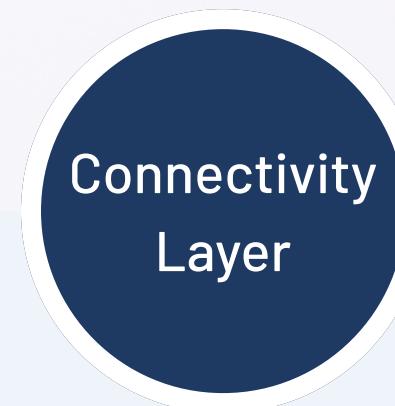


Micro:bit A – Controllo

gestisce i semafori e legge i sensori analogici (Nord, Est, Sud).

Sensori rilevano il passaggio dei veicoli; attuatori (LED/semafori) visualizzano lo stato.

Pulsanti hardware simulano l'attivazione di modalità speciali (es. emergenza). È il livello che interagisce con il mondo reale, attuando la logica semaforica locale.



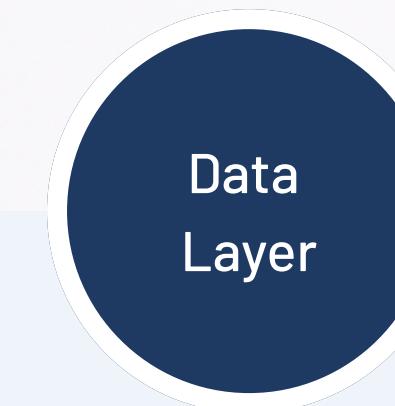
La comunicazione tra i dispositivi avviene via **radio** (RF Micro:bit) e **seriale UART**.

Micro:bit B agisce da **gateway** tra rete radio e il backend su PC.

Uno **script Python** elabora i dati ricevuti e li inoltra tramite **API REST**.

Questo layer garantisce la **trasmissione asincrona e affidabile** dal campo al sistema centrale.

Collega il **mondo fisico** ai livelli **software**.



I dati vengono archiviati in **InfluxDB 2.x** in un bucket dedicato.

Lo script **Python** esegue **parsing, validazione e scrittura** dei dati (stato, sensori).

Il sistema calcola **metriche** come **numero veicoli, cambi di stato e tempo medio su verde**.

I dati sono consultabili da **Grafana** e **WebApp** tramite query temporali.

Questo livello permette **analisi storiche e monitoraggio continuo**.



ARCHITETTURA DEL SISTEMA

LAYER

Twin Layer

La WebApp mostra un **modello digitale sincronizzato** dell'incrocio reale. Visualizza in **tempo reale**: stato semafori, sensori e modalità attiva. Grafana integra dashboard storiche (es. traffico, cambi stato). **Indicatori aggiornati ogni 5s**: veicoli rilevati, tempo medio verde, cambi. È il digital twin che consente monitoraggio e analisi immediata.

Business Layer

Analizza **metriche** per **supportare decisioni sulla viabilità**. Potenziale per tuning semaforico e gestione urbana smart. **Dati utili per scenari futuri**: machine learning e notifiche intelligenti. Può **integrarsi con sistemi di smart city e cruscotti decisionali**. Converte i dati in valore per pianificazione e governance urbana.

LOGICA SEMAFORO INTELLIGENTE



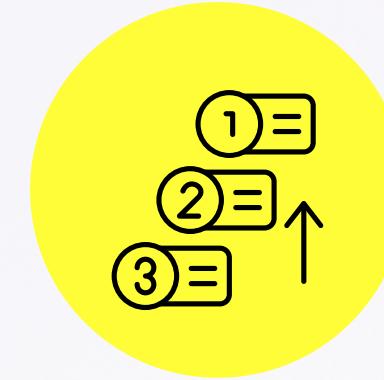
Inizializzazione e Stato Base

All'avvio, la funzione `tutti_rossi()` imposta tutti i semafori su ROSSO: LED rosso acceso, gli altri spenti. Lo stato viene trasmesso via radio e l'indice `idx` viene azzerato per iniziare la rotazione.



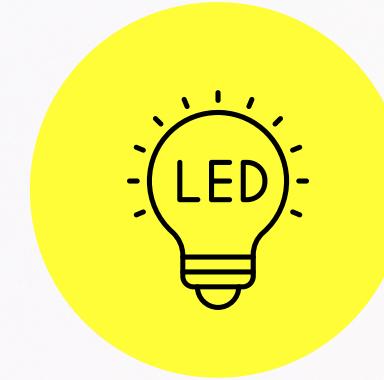
Controllo continuo dei sensori

La funzione `check_sensors()` legge continuamente lo stato analogico dei sensori. Se un valore analogico supera la soglia di attivazione definita dalla costante `SOGLIA` (200), viene considerato attivo.



Ciclo Principale e Selezione Priorità

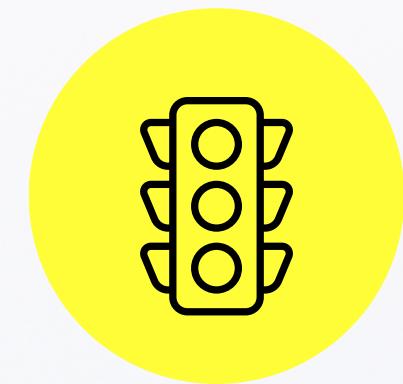
Nel loop principale, il sistema rileva le vie attive (sensori sopra soglia). Se presenti, serve la prima via attiva partendo da `idx`, garantendo equità. Se nessuna via è attiva, si procede in rotazione classica (`servi(idx)`).



Codice ciclo per accensione led

```
Nel loop principale, vengono analizzati i seguenti sensori:  
vals = [v['sens'].read_analog()  
for v in vie]  
active = [i for i, v in  
enumerate(vals) if v > SOGLIA]
```

LOGICA SEMAFORO INTELLIGENTE



Servizio di una Via: Funzione servi(v_idx)

La funzione principale imposta tutti i semafori su rosso (tutti_rossi()), poi attiva il verde solo per la via selezionata. Questo segnale viene inviato via radio e avvia il ciclo semaforico.



Controllo continuo dei sensori

La fase verde dura almeno GREEN_TIME_MIN. Può essere estesa se il sensore resta attivo e non ci sono altre richieste, fino a un massimo (MAX_GREEN_TIME). Passato questo tempo, anche con il sensore ancora premuto, si passa al cambio di stato.

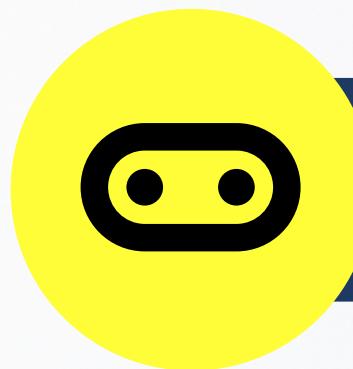


Casistica: Nord premuto e altri due sensori premuti

Se più sensori sono premuti durante il verde di Nord, il sistema segue l'ordine ciclico Nord → Est → Sud. Anche se Sud è premuto prima di Est, verrà servito dopo. Questa scelta assicura coerenza, evitando comportamenti confusi in caso di input simultanei.

COMUNICAZIONE DATI

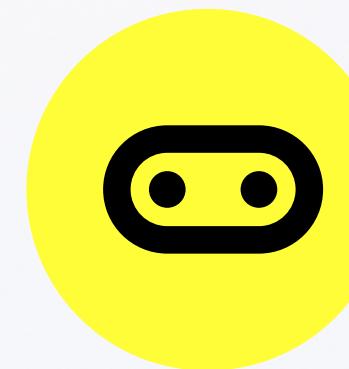
Il sistema realizzato implementa una comunicazione dati distribuita tra dispositivi e livelli software eterogenei. Il flusso dei dati si snoda in quattro segmenti principali:



01

Micro:bit A

Micro:bit A controlla l'incrocio e invia dati via radio a Micro:bit B, che li inoltra al PC tramite USB. Usano radio a 2.4 GHz con configurazione condivisa (channel 3, power 7, group 42). I messaggi includono stato dei semafori, sensori e comandi (STATE, SENSOR).



02

Micro:bit B e PC

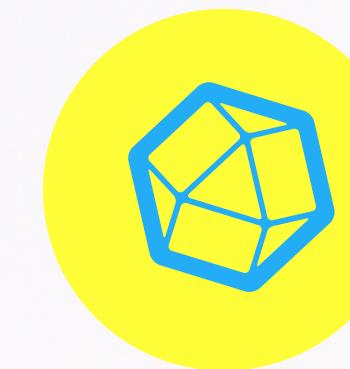
Micro:bit B comunica col PC via porta seriale USB (COM5), ricevendo dati da A e inoltrando comandi (CMD) dal web. Lo script microbit_to_influx.py gestisce lettura, validazione e salvataggio su InfluxDB. I dati sono processati in tempo reale con timestamp.



03

Backend Web e Micro:bit

La web app Flask consente all'utente di inviare comandi come "notte" o "emergenza" tramite pulsanti. Il backend scrive sulla seriale comandi strutturati (es. CMD;NIGHT;ON). Micro:bit B li trasmette via radio a Micro:bit A, che esegue le azioni.



04

WebApp, InfluxDB e Grafana

La web app Flask consente all'utente di inviare comandi come "notte" o "emergenza" tramite pulsanti. Il backend scrive sulla seriale comandi strutturati (es. CMD;NIGHT;ON). Micro:bit B li trasmette via radio a Micro:bit A, che esegue le azioni.

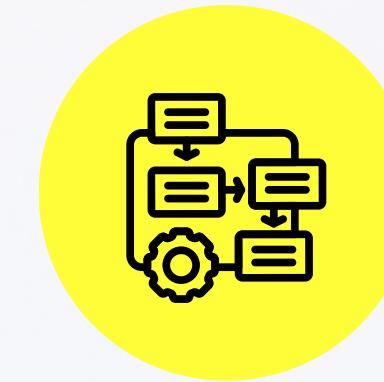
DATABASE

Il sistema utilizza InfluxDB 2.x come database time-series per raccogliere e analizzare i dati di sensori e attuatori raccogliendo dati in tempo reale



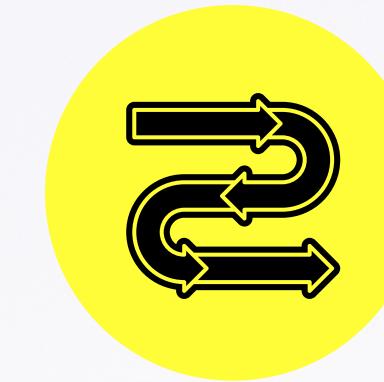
Caratteristiche della configurazione

Il sistema usa InfluxDB 2.x come time-series database, in esecuzione locale su Windows tramite influxd.exe. La configurazione include host su localhost:8086, bucket e organizzazione chiamati microbit, con accesso gestito via token nello script Python.



Flusso di Scrittura dei Dati

Lo script microbit_to_influx.py scrive i dati in InfluxDB traducendo i messaggi ricevuti via seriale da Micro:bit B nel line protocol. I dati includono misure (es. semaforo o sensore), tag (via) e field (stato o attivazione), con timestamp implicito o esplicito.



Flusso di Lettura dei Dati

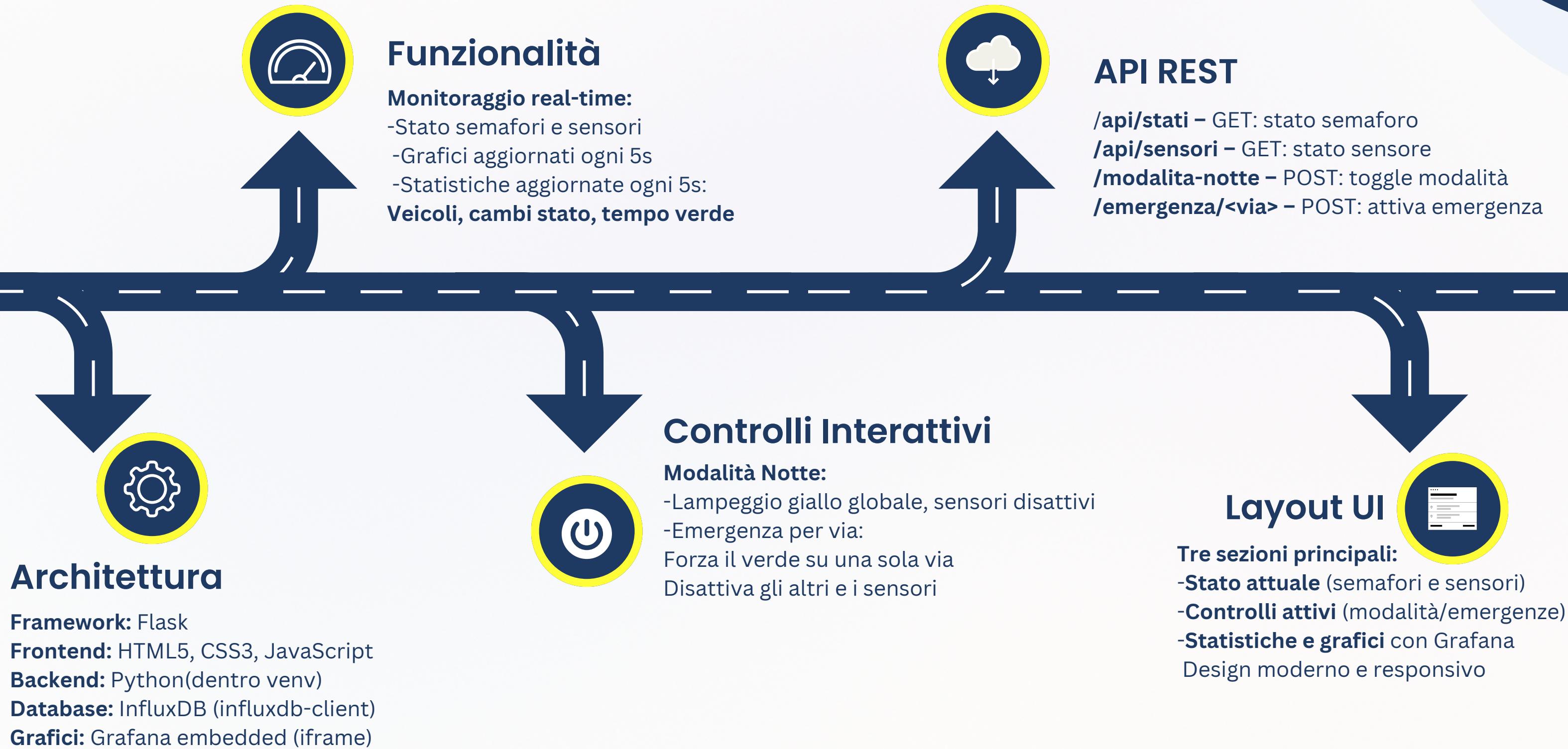
Il database InfluxDB è letto dalla WebApp Flask tramite API che usano query Flux e restituiscono dati JSON per stato e statistiche in tempo reale. Grafana interroga direttamente il bucket microbit, aggiornando ogni 5 secondi con grafici su semafori e sensori.



Statistiche Calcolate

Ogni 10 secondi il sistema calcola per ciascuna via: numero di veicoli rilevati, cambi di stato del semaforo e durata media del verde. Le metriche sono estratte da InfluxDB e visualizzate in una sezione dedicata della WebApp.

WEBAPP



AVVIO E SPEGNIMENTO



1. Verificare che Grafana sia avviato; in caso contrario, usare net start grafana da PowerShell
2. Avviare InfluxDB tramite PowerShell eseguendo influxd.exe dalla cartella corretta (porta: localhost:8086)
3. Collegare il Micro:bit gateway via USB e flashare lo script tramite editor
4. Alimentare il Micro:bit semaforo con la batteria
5. Eseguire microbit_to_influx.py dal terminale nella sua cartella
6. Attivare la virtual environment della WebApp Flask ed eseguire app.py (accesso via <http://127.0.0.1:5000>)



1. Terminare l'esecuzione di app.py dal terminale
2. Terminare lo script microbit_to_influx.py
3. Scollegare il Micro:bit gateway dalla porta USB
4. Spegnere il Micro:bit del semaforo
5. Chiudere influxd.exe da PowerShell
6. Se necessario, arrestare Grafana con net stop grafana

IMPLEMENTAZIONI FUTURE



Rilevamento Multisensore per Stima del Traffico

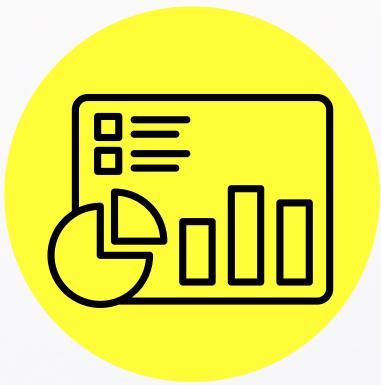
L'attuale configurazione prevede un singolo sensore per ciascuna via. Un'estensione naturale consiste nell'aggiunta di sensori in serie lungo ogni corsia d'accesso all'incrocio. Ciò consentirebbe la stima approssimativa della lunghezza della fila di veicoli, abilitando meccanismi di prioritizzazione più raffinati e proporzionali alla congestione reale. Tali dati potrebbero inoltre essere utilizzati per modellare e prevedere il flusso veicolare medio per intervallo temporale.



Integrazione con Sistemi Smart City

L'intero sistema potrebbe essere trasformato in nodo di una rete urbana interconnessa. Attraverso protocolli MQTT o LoRaWAN, il semaforo smart invierebbe periodicamente i dati a una piattaforma centrale di gestione urbana, facilitando il controllo aggregato e la pianificazione del traffico cittadino.

IMPLEMENTAZIONI FUTURE



Dashboard Avanzata con Storico Interattivo

Estensione della webapp per includere una sezione storica consultabile, in cui siano visibili in forma tabellare o grafica i dati raccolti: numero di veicoli per fascia oraria, durata media delle fasi semaforiche e frequenza delle emergenze.

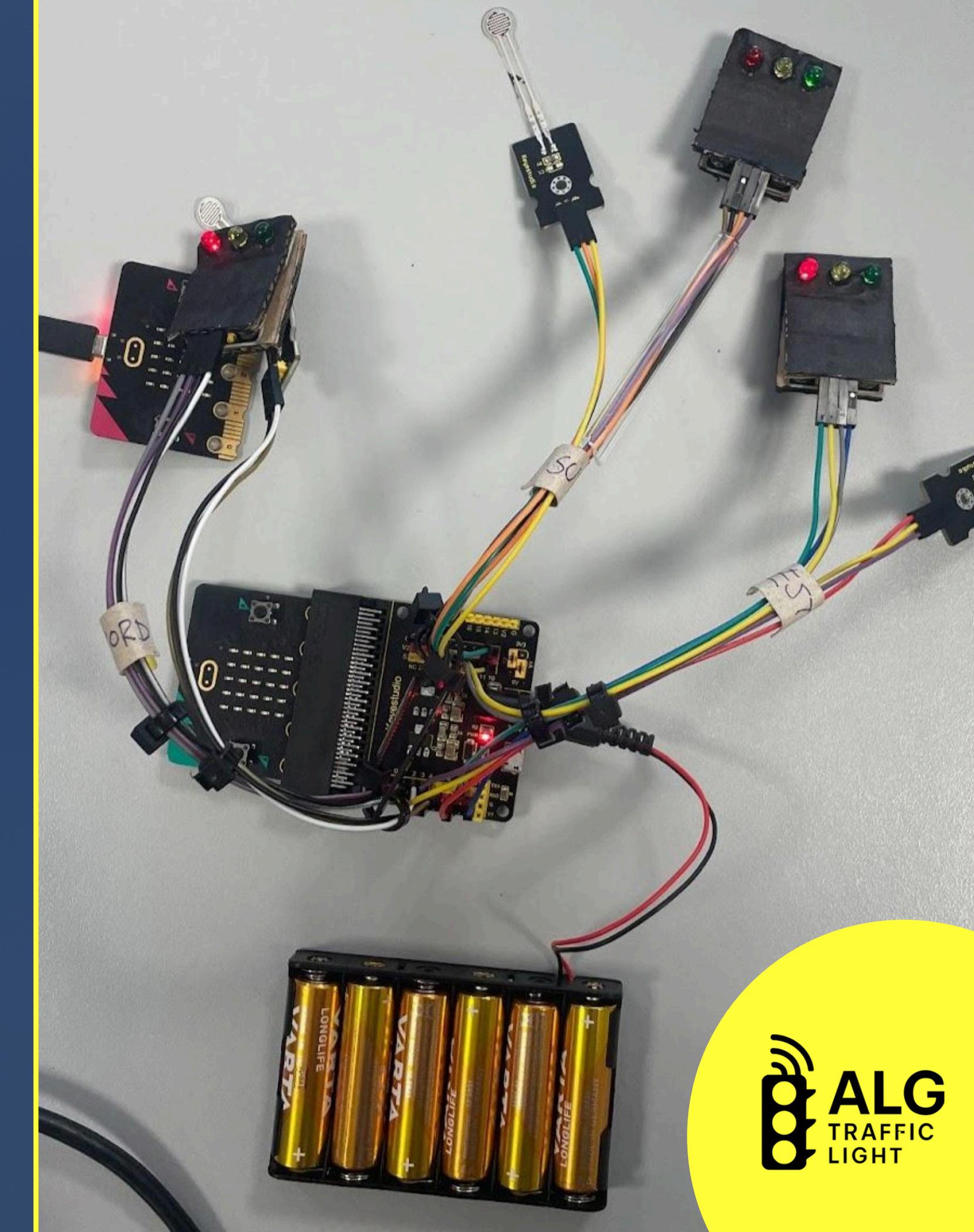


Notifiche Automatiche di Anomalia

L'integrazione con servizi di messaggistica (es. Telegram Bot, Email SMTP) abiliterebbe l'invio di notifiche automatiche in caso di eventi critici: sensori offline, guasti radio, congestioni anomale, durata eccessiva in emergenza.

SCHEMA CIRCUITALE

Direzione	Componente	Signal(s)	VCC (v)	GND
Nord	Sensore Press.	P3	P5	
	LED Rosso	P0		
	LED Giallo	P1		
	LED Verde	P2		
Est	Sensore Press.	P4	P2	
	LED Rosso	P8		
	LED Giallo	P12		
	LED Verde	P13		
Sud	Sensore Press.	P10	P11	
	LED Rosso	P14		
	LED Giallo	P15		
	LED Verde	P16		



SCELTE TECNICHE SUI PIN – MICRO:BIT V2

Sensori di Pressione – Richiesta di pin analogici

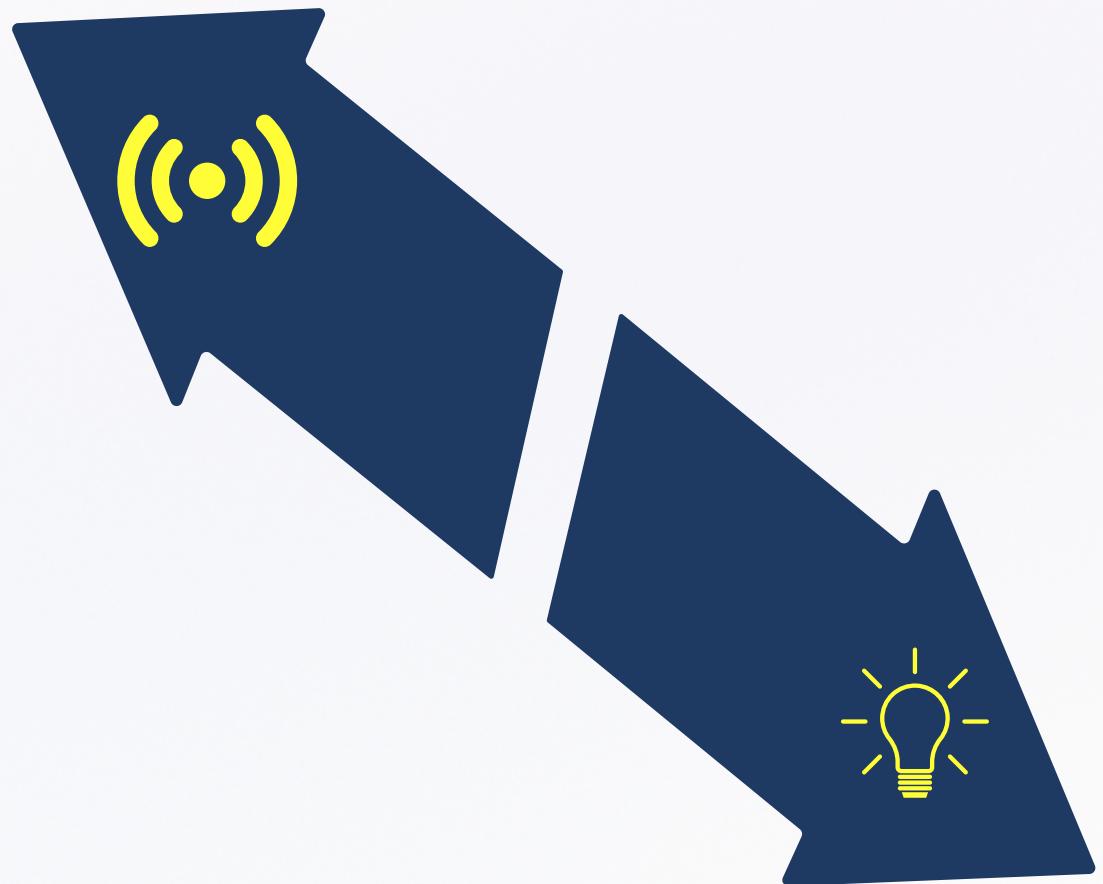
I sensori forniscono valori analogici, quindi necessitano di pin **ADC**.

Micro:bit v2 supporta lettura analogica solo su: **P0, P1, P2, P3, P4, P10**

Assegnazioni scelte:

- P3 → Sensore Nord
- P4 → Sensore Est
- P10 → Sensore Sud

Questa scelta evita errori critici e assicura lettura accurata.



LED – Compatibili con pin digitali

I LED non richiedono lettura analogica.

Possono essere collegati a **qualsiasi pin digitale** non condiviso con sensori.

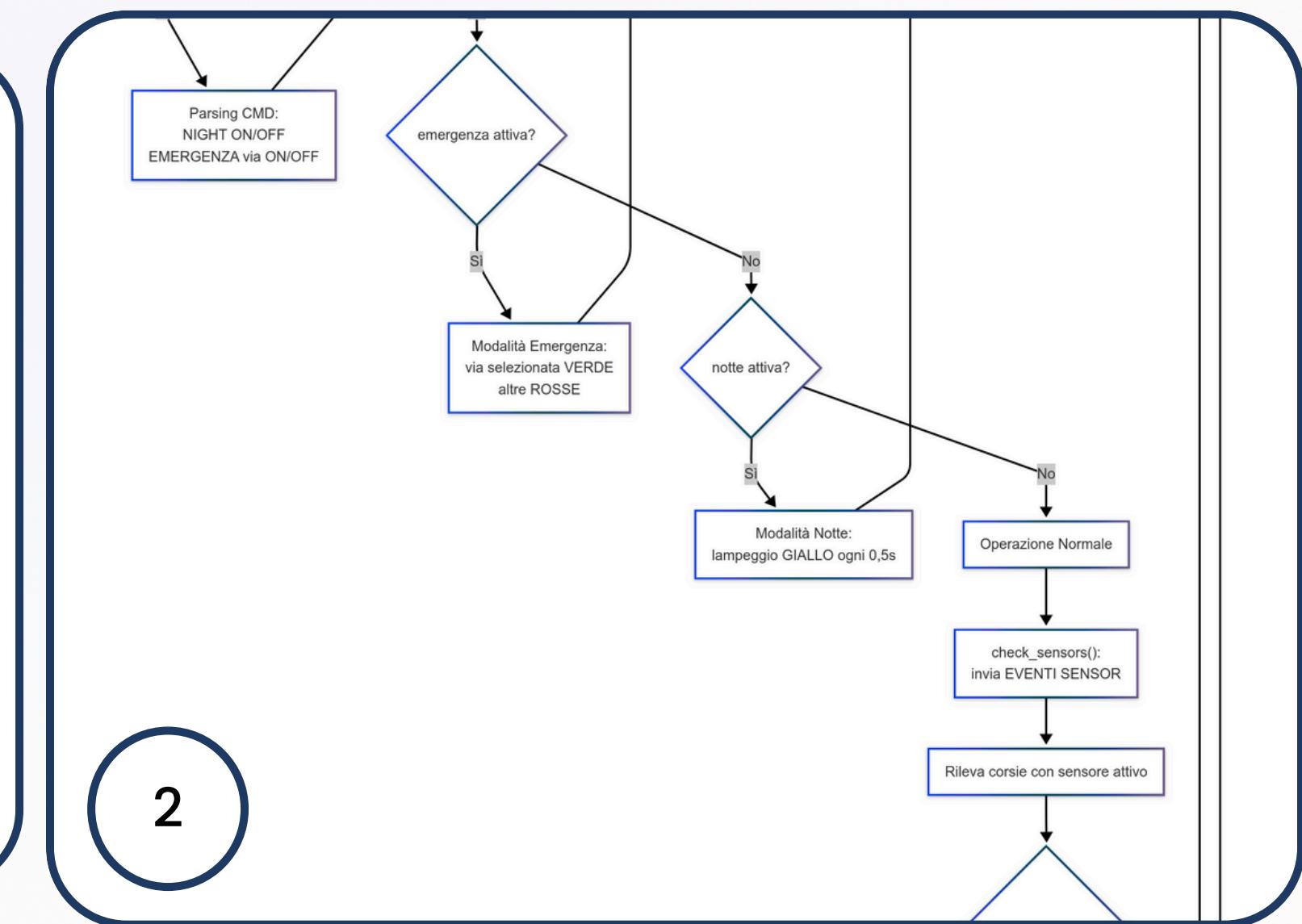
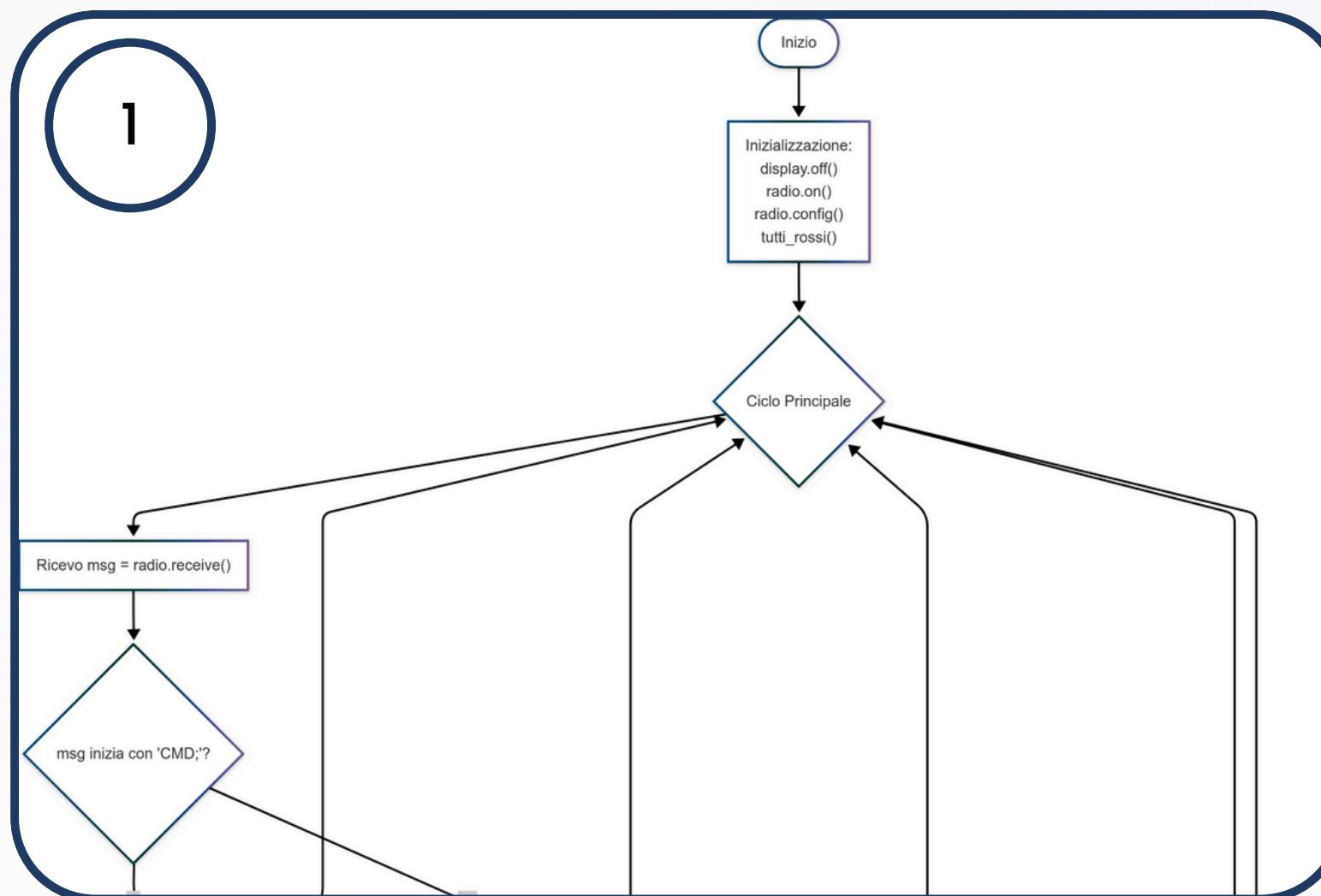
Assegnazioni logiche:

- Nord → P0, P1, P2
- Est → P8, P12, P13
- Sud → P14, P15, P16

P0–P2 sono analogici, ma usati come digitali non crea problemi.



FLUSSO LOGICO



PERCORSI PROGETTUALI

DIARIO

Fase 1 - Analisi e concept

Definiti requisiti e casi d'uso (normale, notte, emergenza).

Scelta architettura: 2 Micro:bit, InfluxDB + Grafana, WebApp Flask.

Fase 2 - Setup hardware e ambiente

Montato circuito: Micro:bit A (semafori + sensori),
Micro:bit B (gateway radio-seriale).
Installati InfluxDB, Grafana, ambiente venv Python.

Fase 3 - Script Micro:bit A

Logica semaforica con funzione tutti_rossi(),
invio messaggi STATE;... e SENSOR;... via radio.
Fix sfarfallio con sleep(50) dopo ogni invio

Fase 4 - Script Micro:bit B

Inoltro bidirezionale radio↔seriale.
Gestione timeout/buffer, stabilizzazione trasmissione.

Fase 5 - Ingestione dati

Script microbit_to_influx.py:
parsing messaggi,
correzione errori (es. "VEDE"),
conversione in line protocol.
Visualizzazione immediata in Grafana.

PERCORSI PROGETTUALI

DIARIO

Fase 6 - Emergenze e statistiche

Aggiunta modalità emergenza e endpoint /stats con calcolo: veicoli, cambi stato, tempo verde medio.

Fase 7 - Backend Flask

Creazione di app.py con API REST:
/api/stati, /api/sensori, /modalita-notte,
/emergenza.

Gestione conflitti COM5 e processi concorrenti.

Fase 8 - Frontend WebApp

UI HTML+CSS+JS responsive con pulsanti interattivi, icone FontAwesome, polling ottimizzato (1s / 10s).

Fase 10 - Documentazione

Redatti: schema circuitale, flowchart, diario, istruzioni di avvio/spegnimento e sezioni tecniche complete.

Fase 9 - Testing e collaudo

Test su pressioni, modalità, cicli e comunicazione.
Stabilità raggiunta con input simultanei.

PERCORSI PROGETTUALI DIFFICOLTÀ INCONTRATE

Cablaggio complesso e rischio errori

- Numerosi collegamenti tra sensori di pressione e LED RGB su Micro:bit.
- Cavetteria fitta, pin ravvicinati → rischio corti e errori.
- Difficoltà nel distinguere pin analogici/digitali/VCC/GND.

SOLUZIONE

- Uso dello shield Sensor Shield V2.0 → facilitato l'inserimento dei moduli.
- Pianificazione fisica ordinata e revisione continua del wiring.
- Etichette e fascette per identificare sensori e direzioni (Nord/Est/Sud).

Errore PermissionError (COM5)

- Conflitto tra microbit_to_influx.py e app.py sulla porta seriale COM5.
Entrambi cercavano di accedere contemporaneamente → errore di permesso (codice 13).

SOLUZIONE

- Separazione dei ruoli:**
Solo microbit_to_influx.py legge da seriale + scrive su InfluxDB.
app.py legge solo dal database, non tocca la porta.
Risultato: sistema stabile, modulare e scalabile.



GRAZIE PER L'ATTENZONE

+39 075 43583



[Repository GitHub](#)



info@algtrafficlight.com

