Progetto Laboratoriale:
Programmazione LEGO EV3 Mindstorm con Python e MicroPython

Campus Leonardo Da Vinci Umbertide A.S. 2024-2025 – Torpei Alessia

Introduzione Progetto

In questo progetto, si esplorerà l'architettura client-server per la gestione di un sistema LEGO EV3 Mindstorm, utilizzando Python e MicroPython. L'obiettivo principale è acquisire i dati (*Figura 1*) da un motore alimentato da una turbina eolica (wind turbine, *Figura 2*), con comunicazione tra il PC (client) e il brick EV3 (server) tramite socket.

```
while True:
    if start_acquisition[0]:  # Controlla se l'acquisizione è attiva
        # Lettura della velocità del motore
        speed = motore.speed()
        speed_str = str(speed)  # Conversione in stringa per l'invio

        try:
            # Invio della velocità al client
            client.send(speed_str.encode())
        except Exception as e:
            brick.display.text(str(e))

    wait(1000)  # Attende 1 secondo prima della successiva lettura e invio
```

Figura 1: acquisizione dei dati

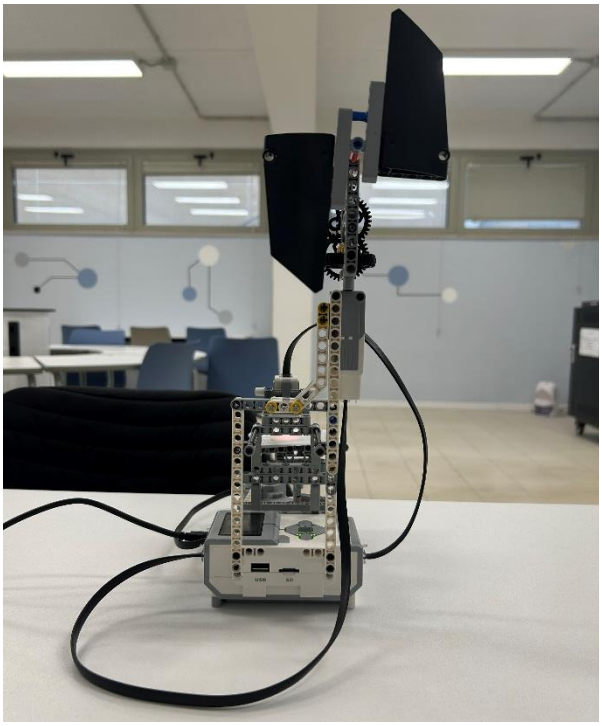

Figura 2: Wind Turbine

1. Basi di Python

Python è un linguaggio di programmazione molto popolare grazie alla sua sintassi semplice e leggibile. Alcuni concetti base che sono stati utilizzati nel progetto includono:

Indentazione: In Python, l'indentazione del codice è fondamentale per definire i blocchi di istruzioni all'interno di costrutti come if, for, e while. A differenza di altri linguaggi, Python non usa parentesi graffe per delimitare il corpo delle istruzioni, ma si affida all'indentazione per stabilire la struttura del programma.

Variabili: Le variabili in Python sono utilizzate per memorizzare i dati. Non è necessario dichiarare il tipo di variabile esplicitamente, poiché Python è un linguaggio a tipizzazione dinamica. Le variabili possono contenere numeri, stringhe, liste, oggetti e altri tipi di dati.

Condizioni (if-else): Le strutture condizionali permettono al programma di prendere decisioni in base a determinate condizioni. Se la condizione è vera, il blocco di codice all'interno dell'if viene eseguito, altrimenti si può eseguire un altro blocco di codice nel caso dell'else. (*Figura 3*)

Cicli (while, for): I cicli sono utilizzati per ripetere un blocco di codice più volte. Un ciclo for è usato per iterare su una sequenza (come una lista o una gamma di numeri), mentre un ciclo while continua a ripetersi finché una condizione specificata è vera.

```
python

if data == "START_MOTOR":
    ...
elif data == "STOP":
    ...
```

Figura 3: esempio di condizioni e indentazione

2. VS Code come IDE

Per sviluppare il progetto, è stato scelto **Visual Studio Code (VS Code)** come ambiente di sviluppo. VS Code è un IDE versatile che supporta numerosi linguaggi di programmazione e offre una vasta gamma di estensioni per migliorare la produttività:

Estensione Python: Questa estensione fornisce supporto per la scrittura, il debugging e l'esecuzione di codice Python. Essa include funzionalità come l'evidenziazione della sintassi, l'autocompletamento e il controllo degli errori in tempo reale.

Estensione MicroPython per LEGO EV3: Questa estensione consente di scrivere e caricare direttamente il codice Python sul LEGO EV3 Brick. Permette di interagire con i motori e i sensori LEGO attraverso il framework MicroPython, una versione ridotta di Python progettata per dispositivi embedded.

VS Code è stato scelto per la sua leggerezza, la configurabilità e la capacità di integrare facilmente queste estensioni.

3. Architettura Client-Server

Nel progetto, è stata adottata un'architettura **client-server**. Questo modello è ampiamente utilizzato per applicazioni distribuite, dove due entità comunicano tra loro per scambiarsi dati e richieste di servizio. In questo caso:

Il PC agisce come client, inviando richieste al brick EV3 per ottenere dati (come la velocità del motore alimentato dalla turbina eolica). Il client è responsabile per l'invio delle richieste e la visualizzazione dei dati ricevuti.

Il LEGO EV3 agisce come server, ricevendo le richieste dal client e rispondendo con i dati richiesti. In pratica, il brick EV3 ascolta le connessioni in arrivo dal client e, quando riceve una richiesta, esegue il comando corrispondente (ad esempio, acquisizione dei dati del motore) e invia una risposta.

Questa architettura consente una comunicazione bidirezionale tra il PC e il brick tramite una rete (*Figura 4*).

```python
server = socket.socket()
server.bind(('', 12345))
server.listen(1)
client, _ = server.accept()
```

Figura 4: connessione server-client

4. Uso delle Librerie LEGO EV3

Il controllo dei motori e dei sensori del LEGO EV3 viene realizzato grazie a una serie di librerie specifiche per il dispositivo. Queste librerie, scritte in MicroPython, permettono di interagire con l'hardware del brick e gestire sensori e motori.

Caricamento del codice sul brick: Il codice Python, sviluppato su un IDE come VS Code, può essere caricato direttamente sul brick EV3 tramite una connessione Wi-Fi o USB. Una volta caricato, il codice può essere eseguito dal brick per controllare i motori, i sensori e altri dispositivi connessi.

Controllo dei motori: Le librerie consentono di controllare i motori LEGO EV3, ad esempio per farli girare a velocità specifiche o fermarli dopo un determinato periodo.

Controllo dei sensori: Si può anche leggere il valore di sensori come il sensore di distanza o il sensore tattile, che possono essere utilizzati per raccogliere dati e reagire agli input fisici. (*Figura 5*)

```python
motore = Motor(Port.A)
sensore = ColorSensor(Port.S1)
```

Figura 5: inizializzazione motori e sensori

5. Socket tra PC e Brick

Un **socket** è un endpoint per la comunicazione tra due dispositivi attraverso una rete. Nel nostro caso, il socket permette la comunicazione tra il PC (client) e il brick EV3 (server). L'utilizzo dei socket consente di inviare e ricevere dati tra il client e il server attraverso una connessione di rete (Wi-Fi o USB).

Il processo di comunicazione avviene in due fasi principali:

Connessione: Il client (PC) si connette al server (brick EV3) specificando l'indirizzo IP del brick e una porta su cui il server sta ascoltando (*Figura 6*).

Scambio di dati: Una volta stabilita la connessione, il client può inviare richieste di dati al server, che risponde con i dati richiesti (ad esempio, i valori del motore).

Questo metodo di comunicazione consente al client di inviare comandi in tempo reale al server e ricevere informazioni in modo continuo e interattivo.

```python
self.client = socket.socket()
self.client.connect((self.ip_entry.get(), 12345))
```

Figura 6: connessione dal client

6. Thread: Gestione dei Processi

In un'applicazione client-server, può essere necessario gestire più processi contemporaneamente, come l'invio e la ricezione di messaggi. Per evitare che un'operazione blocchi l'esecuzione di altre, si utilizza il **threading**. Un **thread** è un'unità di esecuzione indipendente che può operare in parallelo con altri thread (*Figura 7*).

Thread di ricezione: Gestisce la ricezione dei dati dal server (brick EV3) senza bloccare il flusso principale dell'applicazione.

Thread di invio: Si occupa dell'invio dei comandi al brick, come ad esempio l'invio di una richiesta per l'acquisizione dei dati del motore.

L'uso di thread permette di gestire simultaneamente la comunicazione in entrambe le direzioni (dal PC al brick e viceversa) senza rallentare l'interfaccia utente o altre operazioni.

```python
self.thread_conn = threading.Thread(target=self.connetti_ev3)
self.thread_conn.daemon = True
self.thread_conn.start()
```

Figura 7: uso di thread

7. GUI con Tkinter

Per facilitare l'interazione dell'utente con il sistema, è stata progettata un'**interfaccia grafica** (GUI, *Figura 8*) utilizzando **Tkinter**, una libreria standard di Python per la creazione di applicazioni desktop.

Frame: I frame in Tkinter sono contenitori per i widget (componenti dell'interfaccia utente come pulsanti, etichette, caselle di testo, ecc.) e permettono di organizzare visivamente i vari elementi.

Pack: Il metodo pack() è utilizzato per organizzare i widget all'interno dei frame. Esso gestisce il posizionamento dei widget in modo semplice e automatico (*Figura 9*).

Eventi: Tkinter è basato su un sistema di eventi, dove ogni azione dell'utente (come il clic di un pulsante, *Figura 10*) genera un evento che può essere intercettato dal programma per eseguire una determinata funzione. Ad esempio, un pulsante può essere collegato a una funzione che avvia l'acquisizione dei dati.

La GUI permette all'utente di interagire facilmente con il sistema, visualizzare i dati acquisiti e controllare i motori e i sensori in modo intuitivo.

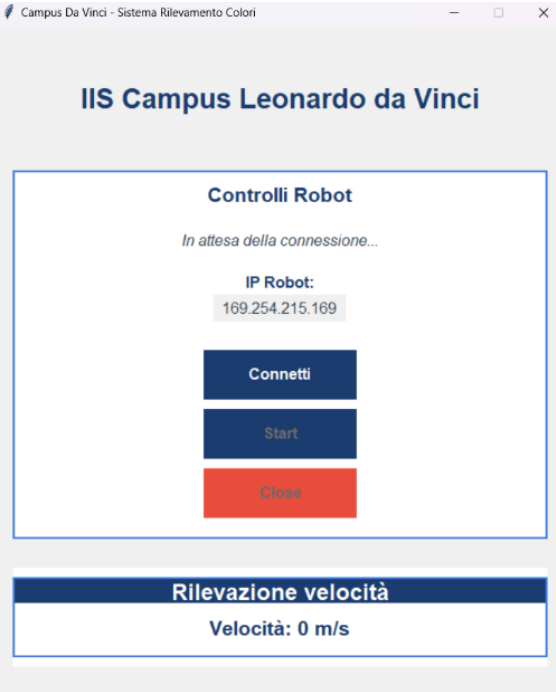

Figura 8: GUI

```python
self.color_display.pack(fill='y', expand=True, pady=10)
```

Figura 9: Gestione del layout con pack

```python
self.btn_connect = tk.Button(..., command=self.avvia_connessione)
```

Figura 10: Eventi sui pulsanti

Concludendo, il progetto ha permesso di applicare e integrare una serie di concetti teorici legati alla programmazione in Python, alla comunicazione tramite socket, e alla creazione di un'interfaccia grafica, tutto nel contesto dell'interazione con un sistema LEGO EV3 Mindstorm