

# RELAZIONE

Kevin Petrone, Marta Di Peri, Marika Cirincione

# Installazione Python



1. Scaricare l'ultima versione di Python dal sito ufficiale:

<https://www.python.org/downloads/> .

2. Avviare l'installer:

- Su Windows: Selezionare l'opzione "**Install launcher for all users (recommended)**" e "**Add Python to PATH**" e cliccare su "**Install Now**".
- Su macOS/Linux: Utilizzare il gestore di pacchetti (ad esempio anaconda)

3. Verificare l'installazione eseguendo tramite cmd:

python --version

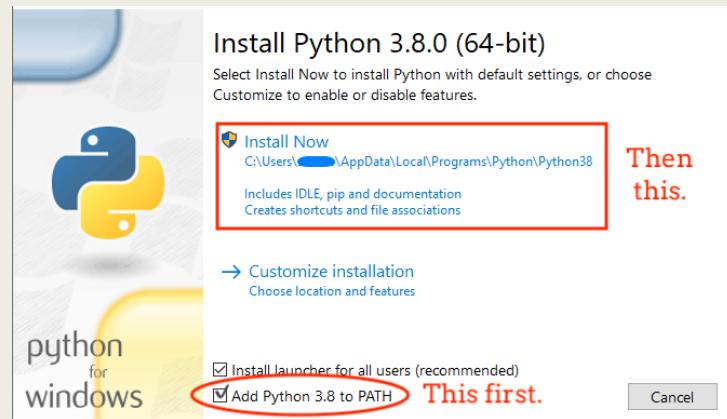
4. Se non sono già selezionati per default, selezionare tutte le opzioni per avere la suite completa. Cliccare su "**Next**".

5. Selezionare la prima opzione: "**for all users**", cliccare su "**Install**" e al successivo messaggio di conferma, rispondere "**si**".

6. Se l'installazione è andata a buon fine, selezionare la disattivazione della lunghezza massima dei path.

**Nota Bene:** Sarà necessario installare tre librerie essenziali tramite il terminale PowerShell di Windows:  
Apri PowerShell e esegui i seguenti comandi uno alla volta:

- Per installare OpenCV (per la gestione di immagini e video):  
`pip3 install opencv-python`
- Per installare NumPy (per il calcolo matematico e operazioni con matrici):  
`pip3 install numpy`
- Per installare ikpy (per calcoli di cinematica inversa):  
`pip install ikpy`



```
Windows PowerShell
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
Prova la nuova PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\utente> pip3 install opencv-python
Requirement already satisfied: opencv-python in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from opencv-python) (2.1.3)
[...]
[!] A new release of pip is available: 23.0.0 > 22.0.2
[!] To update, run: C:\Users\utente\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
PS C:\Users\utente> pip3 install numpy
Requirement already satisfied: numpy in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from opencv-python) (2.1.3)
Requirement already satisfied: scipy in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from numpy) (1.15.1)
Requirement already satisfied: sympy>=1.4.0 in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from numpy) (1.13.1)
Requirement already satisfied: openpyxl>=4.1.0 in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from openpyxl>=4.1.0) (4.1.0)
[...]
[!] A new release of pip is available: 23.0.0 > 22.0.2
[!] To update, run: C:\Users\utente\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
PS C:\Users\utente> pip3 install ikpy
Requirement already satisfied: ikpy in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from ikpy) (2.1.3)
Requirement already satisfied: numpy in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from ikpy) (2.1.3)
Requirement already satisfied: scipy in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from ikpy) (1.15.1)
Requirement already satisfied: sympy>=1.4.0 in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from ikpy) (1.13.1)
Requirement already satisfied: openpyxl>=4.1.0 in c:\users\utente\appdata\local\programs\python\python311\lib\site-packages (from openpyxl>=4.1.0) (4.1.0)
[...]
[!] A new release of pip is available: 23.0.0 > 22.0.2
[!] To update, run: C:\Users\utente\AppData\Local\Programs\Python\Python311\python.exe -m pip install --upgrade pip
PS C:\Users\utente>
```

# Installazione Webots



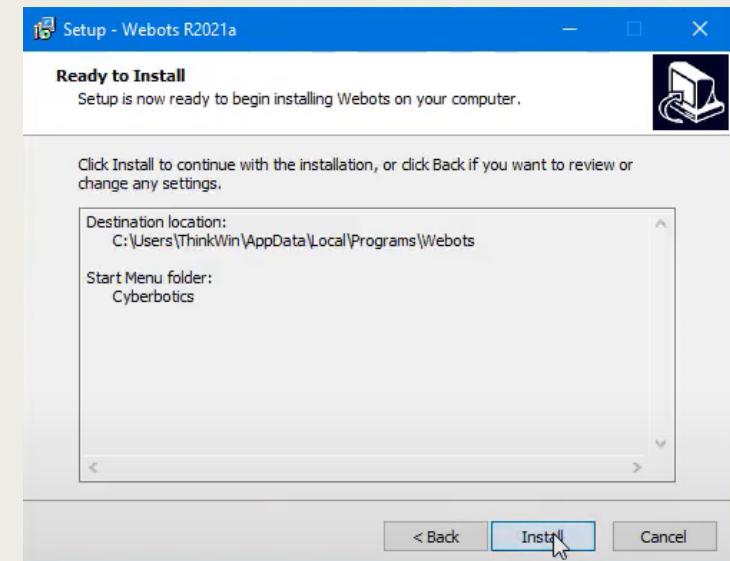
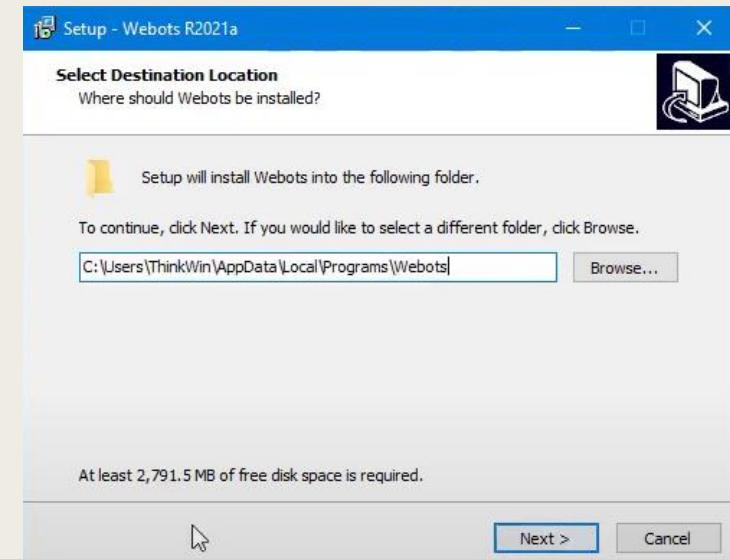
1. Scaricare Webots dal sito ufficiale:

<https://cyberbotics.com/>.

2. Seguire le istruzioni di installazione specifiche per il sistema operativo:

- Windows: Scaricare l'eseguibile e seguire la procedura guidata.
- macOS/Linux: Estrarre il file scaricato e avviare l'installer.

3. Verificare l'installazione avviando Webots e controllando la schermata iniziale.



# Installazione ROS su WSL

(Ubuntu 20.04 e ROS Noetic)



## 1. Installazione di WSL:

- Aprire PowerShell come amministratore ed eseguire:

```
wsl --install
```

- Riavviare il pc, successivamente premere “windows + R” e digitare “**optionalfeatures**”. Qui abilitare Piattaforma macchina virtuale, Sottosistema Windows per Linux, Piattaforma Windows Hypervision.
- Dopo la ricerca dei file cliccare su **riavvia ora**.

```
Amministratore: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

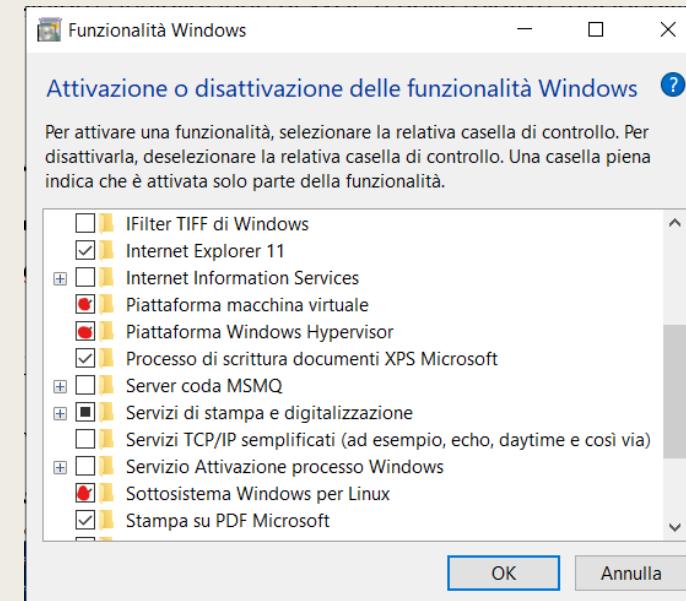
Prova la nuova PowerShell multipiattaforma https://aka.ms/psscoring6

PS C:\Windows\system32> wsl --install
Installazione del componente facoltativo di Windows: Microsoft-Windows-Subsystem-Linux

Strumento Gestione e manutenzione immagini distribuzione
Versione: 10.0.19041.3636

Versione immagine: 10.0.19045.5198

Attivazione funzionalità
[=====100.0%=====]
Operazione completata.
Installazione: Ubuntu
Ubuntu è stato installato.
L'operazione richiesta è stata eseguita. Le modifiche avranno effetto al riavvio del sistema.
PS C:\Windows\system32>
```



# Installazione ROS su WSL



## 2. Installare Ubuntu 20.04v:

- Scarica Ubuntu da Microsoft store <https://shorturl.at/QcyMh>
- una volta eseguito, se richiesto, inserire un username UNIX e una password. «non dimenticarli»

*Nota: Durante la digitazione della password, non verrà mostrato nulla a schermo. In caso di errore, utilizzare il tasto backspace per correggere e riprovare.*

Successivamente, confermare la password.

- Dopo la conferma si avvierà l'installazione di Ubuntu, al termine eseguire questi comandi:

```
sudo apt update
```

Dopo il caricamento, digita:

```
sudo apt upgrade
```

- Digitare Y nella domanda: Do you want to continue?
- Dopo il caricamento, digita:

```
sudo apt autoremove
```

- Infine, chiudere Ubuntu.

```
kevin@DESKTOP-P7VQNL:~$ 
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit https://aka.ms/wslusers
Enter a UNIX username: Kevin
adduser: Please enter a username matching the regular expression configured
via the NAME_REGEX[_SYSTEM] configuration variable. Use the '--force-badname'
option to skip this check or reconfigure NAME_REGEX.
New UNIX username? Kevin
Retype new password:
password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.67.4-microsoft-standard-WSL2 x86_64)

 * Documentation: http://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Fri Dec 13 16:39:23 CET 2024
System load: 0.35      Processes:          42
Usage of /:   0.1% of 1086.85GB  Users logged in:    0
Memory usage: 3%        IPv4 address for eth0: 172.31.101.84
Swap usage:  0B

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESP Apps to receive additional future security updates.
See https://ubuntu.com/esp or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

This message is shown once a day. To disable it please create the
/home/kevin/.bashlogin file.
kevin@DESKTOP-P7VQNL:~$
```

```
kevin@DESKTOP-P7VQNL:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
#
# Patches available for the local privilege escalation issue in needrestart
# tracked by CVE-2024-48990, CVE-2024-48991, CVE-2024-48992, and CVE-2024-10224
# For more see: https://ubuntu.com/blog/needrestart-local-privilege-escalation
#
The following NEW packages will be installed:
  python3-packaging python3-pyparsing ubuntu-pro-client ubuntu-pro-client-l10n
The following packages will be upgraded:
  accountservice apparmor apport apt apt-utils base-files bind9-dnsutils bind9-h
  e2fsprogs fdisk fwupd fwupd-signed gawk gcc-10-base git git-man iputils-ping ip
  libarchive13 libblkid1 libc-bin libc6 libcap2 libcap2-bin libcom-err2 libcurl3-
  libglib2.0-data libgnutls30 libgpgme11 libgssapi-krb5-2 libk5crypto3 liblklbe 1
  libpam-cap libpam-modules libpam-modules-bin libpam-runtime libpam-systemd libp
  libsoup2.4-1 libsqlite3-0 libssh2 libssl1.1 libstdc++6 libsystemd0 libt
  libmotd-news-config mount multipath-tools nano netplan.io open-iscsi open-vm-tools
  python3-configobj python3-cryptography python3-debian python3-distro-info pytho
  python3-software-properties python3-twisted python3-twisted-bin python3-update-
  systemd-sysv systemd-timesyncd tar tcpcdump tzdata ubuntu-advantage-tools udev u
  163 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
108 standard LTS security updates
Need to get 102 MB of archives.
After this operation, 60.4 MB disk space will be freed.
Do you want to continue? [Y/n] Y
```

# Installazione ROS su WSL



## 3. Installazione ROS Noetic:

- Apri WSL e aggiungere i repository ROS:

```
sudo apt update && sudo apt upgrade -y
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt install curl -y
```

```
curl -SSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

- Aggiornare i pacchetti e installare ROS Noetic:

```
sudo apt update && sudo apt upgrade
```

- Installa ROS Noetic:

```
sudo apt install ros-noetic-desktop-full -y
```

- Configurare l'ambiente:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

- Verificare l'installazione:

```
roscore  
printenv | grep ROS
```

```
kevin@DESKTOP-P7VQN4L:~$ rosversion -d  
noetic  
kevin@DESKTOP-P7VQN4L:~$ printenv | grep ROS  
ROS_VERSION=1  
ROS_PYTHON_VERSION=3  
ROS_PACKAGE_PATH=/opt/ros/noetic/share  
ROSLISP_PACKAGE_DIRECTORIES=  
ROS_ETC_DIR=/opt/ros/noetic/etc/ros  
ROS_MASTER_URI=http://localhost:11311  
ROS_ROOT=/opt/ros/noetic/share/ros  
ROS_DISTRO=noetic  
kevin@DESKTOP-P7VQN4L:~$
```

# Collegare ROS a Webots



## 1. Prerequisiti e Installazione

- Assicurati di avere ROS installato e configurato sul tuo sistema.

```
sudo apt-get install python3-pip  
pip3 install rospy
```

## 2. Creazione Publisher (writer\_node.py)

- Creare un nuovo file python chiamato writer\_node.py
- Implementare il [codice publisher](#) con la logica per:
  - Raccogliere input utente
  - Elaborare in formato JSON
  - Scrivere su file JSON
- Configurazione percorso:
  - Modificare il percorso del file JSON al rigo 8, usando il formato corretto
  - Utilizzare forward slash (/) invece di backslash (\)
  - Attenzione l'indirizzo deve iniziare con /mnt/c/Users/... e non C:/Users/

## 3. Configurazione File JSON

- Creare file fsa\_message.json nella cartella fruit\_sorting\_ctrl\_opencv che si trova nella stessa directory del codice Webots
- Strutturare il messaggio di default secondo il formato richiesto nel codice di Webots, esempio: [codice fsa message](#)

# Collegare ROS a Webots



## 4. Modifiche al Codice Webots, implementare le seguenti funzionalità

Funzione per leggere il file JSON, Sistema di fallback in caso di errore, Monitoraggio modifiche file

### A. Setup Iniziale

```
import os # Per gestire i file
import time # Per gestire i timestamp

# Percorso del file
file_path = 'fsa_message.json'
last_modified_time = -1 # Timestamp ultima modifica
first_load = True # Flag per primo caricamento
```

### B. Funzione base

```
def leggi_stringa_da_file(percorso_file):
    """Legge il contenuto del file di configurazione"""
    try:
        with open(percorso_file, 'r') as file:
            return file.read().strip()
    except Exception as e:
        print(f"Errore durante la lettura del file: {e}")
        return None
```

### C. Sistema di Monitoraggio File

```
def is_file_modified(percorso_file, last_modified_time):
    "Controlla se il file è stato modificato"
    global first_load

    try:
        current_modified_time = os.path.getmtime(percorso_file)
        if current_modified_time != last_modified_time:
            first_load = True #Reset del flag nuovo caricamento
            return current_modified_time
        return last_modified_time
    except Exception as e:
        print(f"Errore nel controllo della modifica del file: {e}")
        return last_modified_time
```

### D. Sistema di caricamento

```
def load_fsa_message(percorso_file, last_modified_time):
    """Carica e gestisce il messaggio di configurazione"""
    # Controlla se il file è stato modificato
    last_modified_time = is_file_modified(percorso_file, last_modified_time)

    # Se il file è stato modificato, rilegge e analizza il messaggio
    if last_modified_time != -1:
        #Leggi il nuovo messaggio
        message = leggi_stringa_da_file(percorso_file)

        if message:
            # Processa il messaggio (implementare parse_message secondo necessità)
            FSA = parse_fsa_message(message)
            if FSA:
                return FSA, last_modified_time
            else:
                print("Errore: File FSA non trovato o non leggibile")
                exit(1) # Termina il programma con codice di errore
        else:
            print("Errore: Impossibile controllare le modifiche al file FSA")
            exit(1) # Termina il programma con codice di errore
```

### E. Implementazione nel codice Principale

```
# Inizializzazione
message = leggi_stringa_da_file(file_path)

FSA = parse_message(message)

-----
while robot.step(timestep) != -1:
    #Carica il messaggio FSA se il file è stato modificato
    FSA, last_modified_time = load_fsa_message(file_path, last_modified_time)
```

# Eseguire ROS con Webots

## FASE 1: Preparazione dell'Ambiente ROS

Per iniziare, avviamo due terminali WSL separati. Questi terminali ci permetteranno di gestire separatamente il core di ROS e il nodo writer. È importante aprirli come amministratore per avere i permessi necessari.

### Terminale 1

avviamo `roscore`, il "cervello" di ROS. Questo terminale deve rimanere attivo per tutta l'operazione.

```
kevin@DESKTOP-P7VQN4L:~$ roscore
... logging to /home/kevin/.ros/log/50b075b8-e347-11ef-921f-e1f124c52289/roslaunch-DESKTOP-P7VQN4L-1170.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://DESKTOP-P7VQN4L:40771/
ros_comm version 1.17.0

SUMMARY
=====

PARAMETERS
* /rosdistro: noetic
* /rosversion: 1.17.0

NODES

auto-starting new master
process[master]: started with pid [1178]
ROS_MASTER_URI=http://DESKTOP-P7VQN4L:11311

setting /run_id to 50b075b8-e347-11ef-921f-e1f124c52289
process[rosout-1]: started with pid [1195]
started core service [/rosout]
```

### Terminale 2

Nel secondo terminale WSL, eseguiamo lo script `writer_node.py` nel secondo terminale. Il sistema sarà pronto a ricevere input e modificare il file JSON.

```
kevin@DESKTOP-P7VQN4L:~$ python3 writer_node.py
[INFO] [1738708153.013837]: Inserisci i dettagli delle fasi nel formato:
[INFO] [1738708153.015198]: Esempio: 3, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)
Inserisci il numero e i dettagli di ogni stato: ■
```



# Eseguire ROS con Webots



## FASE 2: Esecuzione del Sistema

Ora avviamo Webots e tramite writer\_node.py, possiamo inviare comandi al robot. Il nodo writer aggiornerà il file JSON nella directory di Windows. Webots rileverà automaticamente le modifiche e aggiornerà il comportamento del robot. Monitoriamo il comportamento del robot in Webots, verificando che le modifiche al file JSON vengano applicate correttamente. Controlliamo eventuali messaggi di errore nei terminali WSL per diagnosticare eventuali problemi.

```
Inserisci i numeri e i dettagli di ogni stato: 4, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4), (1,G2,2,02,3)
[INFO] [1738710988.816544]: Contenuto fasi: [(1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4), (1,G2,2,02,3)]
[INFO] [1738710988.818989]: Fasi dopo la rimozione parentesi: 1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4), (1,G2,2,02,3)
[INFO] [1738710988.821927]: Fasi separate: ['1,G1,1,01,5', '(1,G2,2,02,3)', '(2,01,1,G1,4)', '(1,G2,2,02,3']"
[INFO] [1738710988.824252]: Fase pulita: 1,G1,1,01,5
[INFO] [1738710988.826296]: Fase pulita: 1,G2,2,02,3
[INFO] [1738710988.828826]: Fase pulita: 2,01,1,G1,4
[INFO] [1738710988.831812]: Fase pulita: 1,G2,2,02,3
[INFO] [1738710988.846294]: Dati scritti nel file: /mnt/c/Users/utente/Desktop/ROS_UniversalRobotV3Python/UniversalRobot3.on
[INFO] [1738710988.848389]: Dati scritti nel file JSON: {"fasi": [{"fase": "1,G1,1,01,5", "fase_pulita": "1,G1,1,01,5"}, {"fase": "(1,G2,2,02,3)", "fase_pulita": "(1,G2,2,02,3"}]}
[INFO] [1738710988.851947]: Invio il messaggio: 4, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4), (1,G2,2,02,3)
Vuoi inserire altri dati? (s/n): s
[INFO] [1738710990.542971]: Inserisci i dettagli delle fasi nel formato:
[INFO] [1738710990.544314]: Esempio: 3, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)
Inserisci il numero e i dettagli di ogni stato: 3, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)
[INFO] [1738710994.861579]: Contenuto fasi: [(1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)]
[INFO] [1738710994.864034]: Fasi dopo la rimozione parentesi: 1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)
[INFO] [1738710994.866451]: Fasi separate: ['1,G1,1,01,5', '(1,G2,2,02,3)', '(2,01,1,G1,4']
[INFO] [1738710994.887272]: Fase pulita: 1,G1,1,01,5
[INFO] [1738710994.870504]: Fase pulita: 1,G2,2,02,3
[INFO] [1738710994.922881]: Fase pulita: 2,01,1,G1,4
[INFO] [1738710994.887073]: Dati scritti nel file: /mnt/c/Users/utente/Desktop/ROS_UniversalRobotV3Python/UniversalRobot3.on
[INFO] [1738710994.888849]: Dati scritti nel file JSON: {"fasi": [{"fase": "1,G1,1,01,5", "fase_pulita": "1,G1,1,01,5"}, {"fase": "(1,G2,2,02,3)", "fase_pulita": "(1,G2,2,02,3"}]}
[INFO] [1738710994.891010]: Invio il messaggio: 3, (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)
Vuoi inserire altri dati? (s/n): n
kevin@DESKTOP-PRVWML: ~
```

```
{} fsa_message.json 1 ×  
C: > Users > utente > Desktop > ROS_UniversalRobotV3Python > Universal  
1 3 (1,G1,1,01,5), (1,G2,2,02,3), (2,01,1,G1,4)  
→
```

# ROS

# JSON

# WEBOTS

# Lettura di un file JSON su Webots



## 1. Lettura del file JSON in Python

Il codice Python usa la funzione `leggi_stringa_da_file()` per leggere il contenuto del file.  
Analizziamo questa funzione

```
# Funzione per leggere la stringa dal file
def leggi_stringa_da_file(percorso_file):
    try:
        with open(percorso_file, 'r') as file:
            return file.read().strip() # Rimuove gli spazi vuoti e legge il contenuto
    except Exception as e:
        print(f"Errore durante la lettura del file: {e}")
        return None
```

- `open(percorso_file, 'r')`: apre il file in modalità lettura ('r').
- `file.read()`: legge l'intero contenuto del file.
- `strip()`: rimuove eventuali spazi bianchi o caratteri di nuova riga all'inizio e alla fine.
- Blocco try-except: se si verifica un errore (es. file non trovato), viene stampato un messaggio d'errore e restituito None.

## 2. Controllo se il file esiste e contiene dati

```
# Esegui il parsing del messaggio
FSA = parse_fsa_message(message)

if FSA is None:
    print("Errore: Parsing del messaggio FSA fallito")
    exit(1) # Termina il programma con codice di errore
```

- Dopo aver letto il file, il codice verifica se `message` non è None. Se `message` è vuoto, il programma termina con `exit(1)`.
- Questo garantisce che non si tenti di analizzare un file inesistente o corrotto.

# Lettura di un file JSON su Webots



## 3. Parsing del messaggio e creazione della configurazione FSA

La funzione `parse_fsa_message(message)` si occupa di convertire la stringa letta dal file in una struttura dati più facile da gestire (un dizionario).

- Estrazione del numero di stati: Il codice prende il primo valore della stringa (separato da ,), lo converte in intero e lo memorizza in `num_states`.
- Estrazione delle configurazioni di stato: Usa una regex per trovare tutti i blocchi che contengono configurazioni di stato nel formato (1,G1,2,02,5). Ogni blocco è memorizzato nella lista `state_configs`.
- Verifica della coerenza tra il numero di stati e i dati trovati: Se il numero di stati nel file non corrisponde al numero di configurazioni trovate, viene mostrato un errore.
- Creazione del dizionario FSA: Per ogni stato trovato, si creano le sue proprietà:
  - `trigger`: il numero dello stato successivo
  - `requirements`: quanti oggetti devono essere presenti nei contenitori
  - `delay`: il ritardo prima del cambio di stato
- Aggiunta dello stato HALT: Lo stato di arresto (HALT) ha un trigger di -1, nessun requisito e un delay di 0.
- Debug: Se `first_load` è True, viene stampata una tabella con tutti gli stati e le loro configurazioni.
- Restituzione del dizionario FSA: Se tutto è andato bene, la funzione restituisce il dizionario `fsa`.

## 4. Monitoraggio delle modifiche al file

```
# Funzione per controllare se il file e' stato modificato
def is_file_modified(percorso_file, last_modified_time):
    global first_load_is_file_modified

    try:
        current_modified_time = os.path.getmtime(percorso_file)
        if current_modified_time != last_modified_time:
            return current_modified_time # Il file e' stato modificato
            first_load = True
        return last_modified_time # Nessuna modifica
    except Exception as e:
        print(f"Errore nel controllo della modifica del file: {e}")
    return last_modified_time
```

Viene usata la funzione `is_file_modified()` per controllare se il file `fsa_message.json` è stato modificato.

Passaggi:

- 1.Si ottiene il tempo dell'ultima modifica con `os.path.getmtime(percorso_file)`.
- 2.Se il valore è diverso dal precedente, significa che il file è cambiato.

# Lettura di un file JSON su Webots



## 5. Ricaricamento FSA se il file cambia

Se il file viene modificato, `load_fsa_message()` rilegge e riprocesso il messaggio.

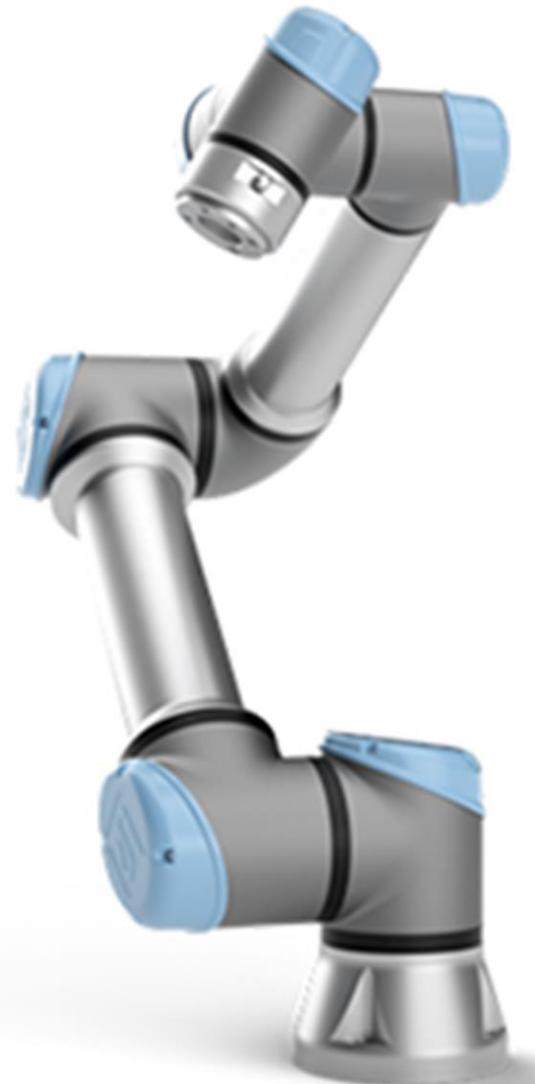
```
# Funzione per leggere e analizzare il messaggio FSA
def load_fsa_message(percorso_file, last_modified_time):
    # Controlla se il file è stato modificato
    last_modified_time = is_file_modified(percorso_file, last_modified_time)

    # Se il file è stato modificato, rilegge e analizza il messaggio
    if last_modified_time != -1:
        message = leggi_stringa_da_file(percorso_file)

        if message:
            FSA = parse_fsa_message(message)
            if FSA is None:
                print("Errore: Parsing del messaggio FSA fallito")
                exit(1) # Termina il programma con codice di errore
            else:
                print("Errore: File FSA non trovato o non leggibile")
                exit(1) # Termina il programma con codice di errore
        else:
            print("Errore: Impossibile controllare le modifiche al file FSA")
            exit(1) # Termina il programma con codice di errore

    return FSA, last_modified_time
```

- Chiama la funzione `is_file_modified()` per verificare se il file è stato modificato rispetto all'ultima volta che è stato letto.
- Se il file è stato modificato, `last_modified_time` viene aggiornato con il nuovo timestamp. Altrimenti, il valore di `last_modified_time` rimane invariato.
- Se il file è stato modificato, allora si procede alla rilettura.
- Viene richiamata la funzione `leggi_stringa_da_file()` per leggere il nuovo contenuto del file.
- Se il file è stato letto correttamente (cioè `message` non è `None` o vuoto), allora il contenuto viene passato alla funzione `parse_fsa_message()` per convertirlo in un dizionario di stati FSA.
- Se la funzione `parse_fsa_message()` restituisce `None`, significa che si è verificato un errore nel parsing del messaggio e viene stampato un errore e il programma termina con `exit(1)`. Se il file non è stato letto correttamente, viene stampato un errore e il programma viene terminato.
- Se `is_file_modified()` non riesce a determinare se il file è stato modificato, viene stampato un errore e il programma termina .
- Alla fine della funzione, viene restituito:
  - Il nuovo dizionario FSA contenente la configurazione aggiornata.
  - Il valore aggiornato di `last_modified_time`.



	Reach 850 mm / 33.5 in
	Payload 5 kg / 11 lbs
	Footprint Ø 149 mm
	Weight 18,4 kg / 45.4 lbs

# Il Robot UR5e

Il UR5e è un robot collaborativo apprezzato per la sua flessibilità, semplicità d'uso e capacità di operare a fianco degli esseri umani senza richiedere barriere protettive. Dispone di 6 gradi di libertà, che gli conferiscono un'elevata mobilità e versatilità nei movimenti. Grazie a queste caratteristiche, viene anche definito un "manipolatore diretto".

Le componenti principali di un robot possono essere suddivise in due categorie principali:

1. Manipolatore
2. Sistema di controllo e programmazione

Con il seguente link: [Cyberbotics UR5e Guide](#), puoi visualizzare il robot UR5e direttamente online. Questo strumento permette di esplorare il funzionamento del robot in un ambiente virtuale interattivo, consentendo di muovere i suoi giunti e attuatori per comprendere meglio le sue capacità di movimento e configurazione.



**UNIVERSAL ROBOTS**

# Organizzazione codice e strutture usate

## FSA (Finite State Automaton):

- Implementa una macchina a stati finiti definita da un **messaggio di configurazione**
- Ogni stato contiene: **trigger** per lo stato successivo, **requisiti** (conteggi per bin verdi/arancioni), e **delay**
- Gli stati vengono gestiti in `main_state()` che controlla **transizioni** e **requisiti**

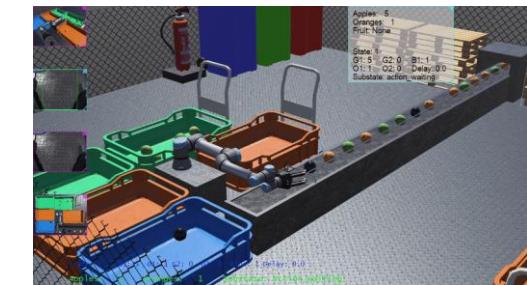
## Strutture Principali:

- Array di posizioni target per il robot
- Contatori per frutti e bin
- Sistema di substati per le azioni del robot (waiting, picking, rotating, dropping, rotate\_back)
- Display informativo e sistema di riconoscimento frutti via computer vision

## Organizzazione Codice:

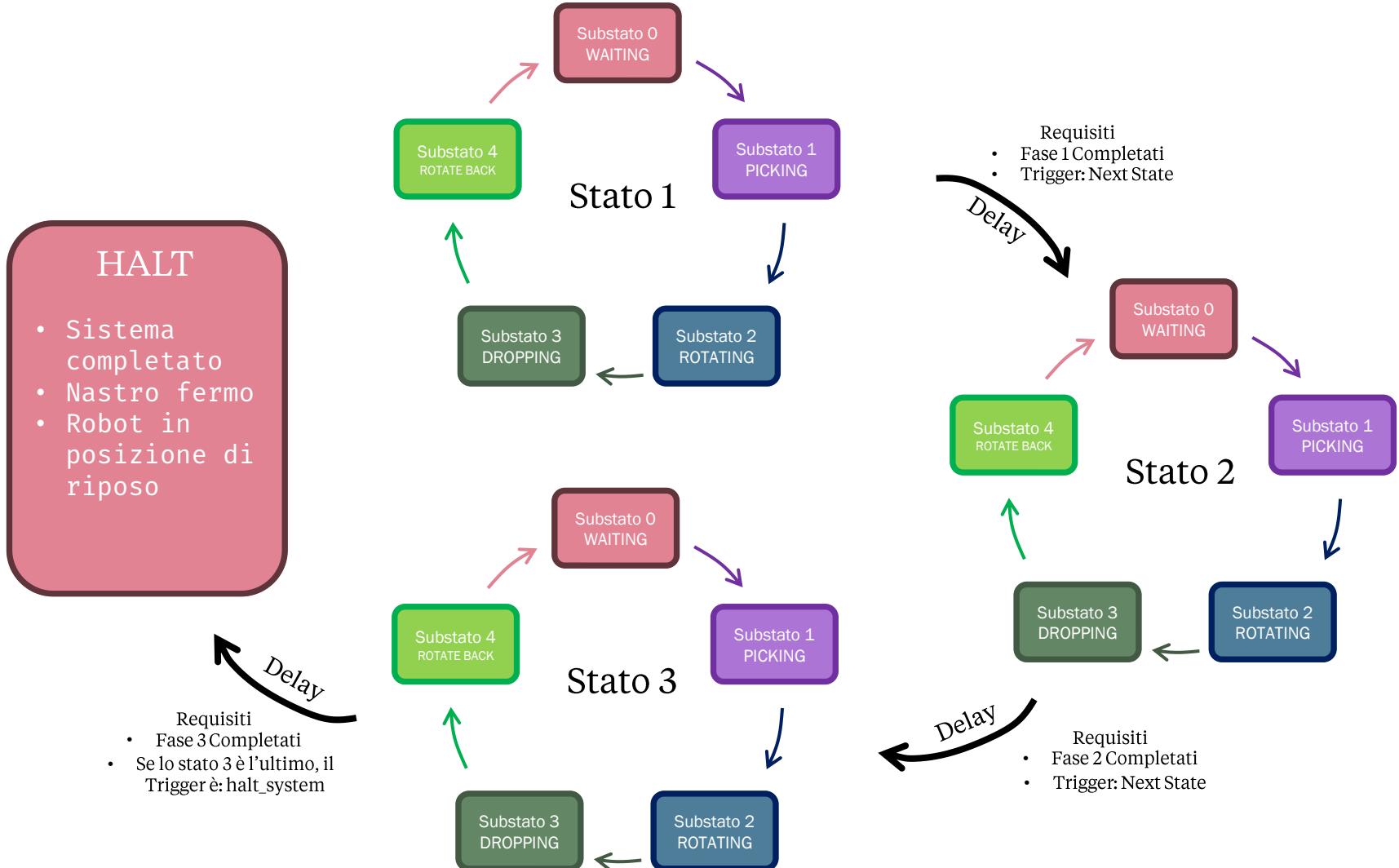
- Inizializzazione dei dispositivi robot (motori, sensori, camera, display)
- Funzioni helper per display e audio
- Macchina a substati implementata come **dizionario di funzioni**
- Loop principale che alterna tra gestione dello stato FSA e esecuzione dei substati
- Sistema di computer vision per identificare i frutti sul nastro

Il codice segue un'architettura gerarchica con FSA di alto livello che gestisce gli stati generali, mentre i substati gestiscono le azioni specifiche del robot.



[Visualizza  
il codice](#)

# Automa a Stati Finiti (FSA)



# Importazione e Configurazione Base

Questa sezione rappresenta le fondamenta del sistema. Include tutte le librerie necessarie per il funzionamento del robot, gestendo l'elaborazione delle immagini (OpenCV), le operazioni matematiche (NumPy), il controllo del robot (**Supervisor**) e varie utility di sistema.

L'inizializzazione del **robot** e del **nastro trasportatore** è cruciale: qui vengono stabiliti i collegamenti con l'hardware e verificata la corretta comunicazione con tutti i componenti. Il sistema implementa anche controlli di sicurezza per garantire che tutti i componenti siano correttamente inizializzati prima di procedere con le operazioni.

```
import cv2 # OpenCV per l'elaborazione delle immagini
import numpy as np # NumPy per operazioni numeriche e array
from controller import Supervisor # Libreria Webots per controllare il robot
import math # Libreria per operazioni matematiche
import time # Libreria per gestione del tempo
import re # Libreria per le espressioni regolari
import os # Per gestire i file
```

```
robot = Supervisor()
conveyor_belt = robot.getFromDef("conveyor_belt")

if conveyor_belt is None:
    print("Errore: Nodo 'conveyor_belt' non trovato nel file .wbt.")
else:
    speed_field = conveyor_belt.getField("speed")
    if speed_field is None:
        print("Errore: Campo 'speed' non trovato nel nodo 'conveyor_belt'.")
    speed_field.setSFFloat(0.15)
```

# Parametri di Configurazione

Il sistema utilizza una serie complessa di **parametri** che definiscono il suo comportamento. Questi includono:

- **Parametri temporali** per la sincronizzazione delle operazioni
- **Contatori** per il monitoraggio della frutta processata
- **Flag di sistema** per il controllo dello stato
- **Variabili di gestione del tempo** per coordinare le operazioni
- **Contatori** specifici per ogni contenitore

Questa configurazione permette una gestione flessibile e precisa del sistema, consentendo facili modifiche al comportamento senza alterare la logica di base.

```
# --- Parametri Base ---
timestep = 32
current_state = 1
numero_stati = 0
first_load = True

# --- Stati e Contatori ---
fruit = -1
fruit_names = ['Orange', 'Apple', 'Rottenapple']
apple, orange, rottenapple = 0, 0, 0
state_apple_count = 0
state_orange_count = 0

# --- Flag di Sistema ---
counter = 0
is_process_complete = False
main_state_changed = False
elapsed_time = 0

# --- Gestione Tempo ---
state_start_time = 0
state_delay_active = False
state_delay_end_time = 0

# --- Contatori Contenitori ---
counter_01 = 0
counter_G1 = 0
counter_02 = 0
counter_G2 = 0
counter_binblue = 0

# --- Definizioni di variabile dopo le altre
# dichiarazioni di variabile globale ---
bin_green1, bin_green2 = "bin_green1", "bin_green2"
bin_orange1, bin_orange2 = "bin_orange1", "bin_orange2"

# --- Dizionario per tenere traccia dei
# conteggi nei contenitori ---
state_bin_counts = {
    "bin_green1": 0,
    "bin_green2": 0,
    "bin_orange1": 0,
    "bin_orange2": 0
}
```

# Commenti codice «target position»

Le posizioni target rappresentano un elemento critico del sistema. Sono state calibrate accuratamente per:

- Garantire movimenti precisi del robot
- Evitare collisioni
- Ottimizzare i tempi di movimento
- Assicurare una presa sicura della frutta
- Permettere un rilascio accurato nei contenitori

Ogni posizione è definita da coordinate specifiche per i giunti del robot, calcolate per massimizzare l'efficienza e la sicurezza delle operazioni.

```
# --- Posizioni Target ---
target_HALT = [0, -1.57, 0, -1.57, 0] # (HALT)
target_positions = [
    [-1.570796, -1.87972, -2.139774, -2.363176, -1.50971], # 01
    [0, -1.87972, -2.139774, -2.363176, -1.50971], # G1
    [-1, -1.67972, +1.539774, -2.163176, -1.50971], # B1
    [0, -1.57, 0, -1.57, 0], # HALT
    [1.570796, -1.87972, -2.139774, -2.363176, -1.50971], # 02
    [1, -1.87972, -2.139774, -2.363176, -1.50971] # G2
]
```

Questa sezione gestisce l'inizializzazione e la configurazione di tutti i dispositivi hardware:

- Motori del gripper per la presa della frutta
- Motori del braccio robotico UR5e
- Sensori di distanza e posizione
- Sistema di visione (camera)
- Display per il feedback visivo
- Sistema audio per il feedback sonoro

Ogni dispositivo viene configurato con parametri specifici e viene verificato il suo corretto funzionamento prima dell'avvio delle operazioni.

```
# Inizializzazione motori i Gripper
hand_motors = []
hand_motors.append(robot.getDevice('finger_1_joint_1'))
hand_motors.append(robot.getDevice('finger_2_joint_1'))
hand_motors.append(robot.getDevice('finger_middle_joint_1'))

# Inizializzazione motori UR5e
ur_motors = []
ur_motors.append(robot.getDevice('shoulder_pan_joint'))
ur_motors.append(robot.getDevice('shoulder_lift_joint'))
ur_motors.append(robot.getDevice('elbow_joint'))
ur_motors.append(robot.getDevice('wrist_1_joint'))
ur_motors.append(robot.getDevice('wrist_2_joint'))

# Configurazione Velocita'
for i in range(5):
    ur_motors[i].setVelocity(speed)

# Inizializzazione Sensori
distance_sensor = robot.getDevice('distance sensor')
distance_sensor.enable(timestep)

# Inizializzazione Camera e Display
position_sensor = robot.getDevice('wrist_1_joint_sensor')
position_sensor.enable(timestep)

# Inizializzazione della telecamera
camera = robot.getDevice('camera')
camera.enable(timestep)

# Inizializzazione del display
display = robot.getDevice('display')
display.attachCamera(camera)
display.setColor(0x00FF00)
display.setFont('Verdana', 16, True)

# Inizializzazione speaker
speaker = robot.getDevice('speaker')
```

# Commenti codice «pannello info»

La funzione ‘**draw\_info\_panel()**’ serve a mostrare in tempo reale lo stato e i vari contatori del sistema robotico su un display, permettendo di monitorare facilmente il funzionamento del robot. Ottiene il riferimento al display del robot e verifica che sia disponibile. Configura lo sfondo del display (bianco con leggera trasparenza). Imposta il font (Arial 14px) e il colore del testo (nero).

```
def draw_info_panel():
    info_display = robot.getDevice("info_display")

    if info_display is None:
        print("Errore: Display non trovato!")
        return

    info_display.setColor(0xFFFFFFFF) # Sfondo bianco
    info_display.setAlpha(0.8) # Leggera trasparenza
    info_display.fillRect(0, 0, 270, 180)

    info_display.setFont("Arial", 14, True)
    info_display.setColor(0x000000) # Testo nero
    info_display.setAlpha(1.0) # Testo completamente opaco
    current_requirements = FSA[current_state]["requirements"] if current_state in FSA else {}
    # Calcola i requisiti totali per mele e arance nello stato corrente
    required_apples = sum(count for bin_name, count in current_requirements.items() if bin_name.startswith("bin_green"))
    required_oranges = sum(count for bin_name, count in current_requirements.items() if bin_name.startswith("bin_orange"))
    # Calcola il delay rimanente
    if state_delay_active and not is_process_complete:
        remaining_delay = max(0, state_delay_end_time - robot.getTime())
    else:
        remaining_delay = 0

    info_display.drawText(f"Apples: {apple:3d} {state_apple_count}|{required_apples}", 10, 10)
    info_display.drawText(f"Oranges: {orange:3d} {state_orange_count}|{required_oranges}", 10, 30)
    info_display.drawText(f"Fruit: {fruit_names[fruit]} if fruit != -1 else 'None'", 10, 50)
    info_display.drawText(f"State: {current_state}|{numero_stati}", 10, 100)
    g1_req = current_requirements.get(bin_green1, 0)
    g2_req = current_requirements.get(bin_green2, 0)
    o1_req = current_requirements.get(bin_orange1, 0)
    o2_req = current_requirements.get(bin_orange2, 0)
    info_display.drawText(f"G1: {counter_G1} {state_bin_counts[bin_green1]}|{g1_req}", 10, 120)
    info_display.drawText(f"G2: {counter_G2} {state_bin_counts[bin_green2]}|{g2_req}", 10, 140)
    info_display.drawText(f"O1: {counter_O1} {state_bin_counts[bin_orange1]}|{o1_req}", 10, 160)
    info_display.drawText(f"O2: {counter_O2} {state_bin_counts[bin_orange2]}|{o2_req}", 10, 180)
    info_display.drawText(f"B1: {counter_binblue}", 180, 120)
    info_display.drawText(f"Delay: {remaining_delay:.1f}", 180, 140)
    info_display.drawText(f"Substate: {'END' if current_substate == 'END' else (current_substate._name__ if hasattr(current_substate, '__name__') else 'Unknown')}", 10, 160)
```



Il monitoraggio fornisce feedback in tempo reale sullo stato del sistema:

- Conteggio della frutta processata
  - Stato attuale del sistema
  - Requisiti di completamento
  - Tempi di elaborazione
  - Stato dei contenitori
  - Eventuali errori o anomalie
- Queste informazioni sono visualizzate su un display dedicato e aggiornate in tempo reale.

# Commenti codice «messaggio FSA parte 1»

```
def parse_fsa_message(message):
    try:
        num_states = int(message.split(',')[0].strip())
        state_configs = re.findall(r'\\((\\d+, (?G|O)\\d+, \\d+, (?G|O)\\d+, \\d+))', message)

        if len(state_configs) != num_states:
            print(f"Errore: Il numero di stati ({num_states}) non corrisponde al numero di configurazioni ({len(state_configs)})")
            return None

        fsa = {}

        for i, config in enumerate(state_configs, start=1):
            values = config.split(',')
            g1_count = int(values[0])
            g1_bin = values[1]
            o1_count = int(values[2])
            o1_bin = values[3]
            delay = int(values[4])

            bin_green = f"bin_green{g1_bin[-1]}"
            bin_orange = f"bin_orange{o1_bin[-1]}"

            next_state = i + 1 if i < num_states else -1

            fsa[i] = {
                "trigger": next_state,
                "requirements": {
                    bin_green: g1_count,
                    bin_orange: o1_count
                },
                "delay": delay
            }

        return fsa

    except Exception as e:
        print(f"Errore nel parsing del messaggio FSA: {e}")
        return None

message = "3, (1,G1,1,01,5),(1,G2,2,02,3),(2,01,1,G1,4)"
fsa = parse_fsa_message(message)
```

Questa funzione analizza un messaggio che descrive un **automa a stati finiti (FSA)** per gestire la configurazione di contenitori verdi e arancioni.

Prende in input una stringa nel formato "numero\_stati, (config1),(config2),..." dove **ogni configurazione è una sequenza di 5 valori**: quantità per contenitore verde, nome contenitore verde, quantità per contenitore arancione, nome contenitore arancione, e ritardo. Nota: L'ordine dei contenitori nella configurazione è intercambiabile, quindi una configurazione può essere scritta sia come (2,G1,1,01,5) che come (1,01,2,G1,5), mantenendo lo stesso significato.

# Commenti codice «messaggio FSA parte 2»

```
def parse_fsa_message(message):
    try:
        num_states = int(message.split(',')[0].strip())
        state_configs = re.findall(r'\\((\\d+, (?G|O)\\d+, \\d+, (?G|O)\\d+, \\d+))', message)

        if len(state_configs) != num_states:
            print(f"Errore: Il numero di stati ({num_states}) non corrisponde al numero di configurazioni ({len(state_configs)})")
            return None

        fsa = {}

        for i, config in enumerate(state_configs, start=1):
            values = config.split(',')
            g1_count = int(values[0])
            g1_bin = values[1]
            o1_count = int(values[2])
            o1_bin = values[3]
            delay = int(values[4])

            bin_green = f"bin_green{g1_bin[-1]}"
            bin_orange = f"bin_orange{o1_bin[-1]}"

            next_state = i + 1 if i < num_states else -1

            fsa[i] = {
                "trigger": next_state,
                "requirements": {
                    bin_green: g1_count,
                    bin_orange: o1_count
                },
                "delay": delay
            }

        return fsa

    except Exception as e:
        print(f"Errore nel parsing del messaggio FSA: {e}")
        return None

message = "3, (1,G1,1,01,5),(1,G2,2,02,3),(2,01,1,G1,4)"

fsa = parse_fsa_message(message)
```

## La funzione:

- Estrae il numero di stati dalla stringa
- Usa regex per trovare le configurazioni tra parentesi
- Crea un dizionario dove ogni stato ha:
  - Il trigger per lo stato successivo
  - I requisiti (quanti oggetti nei contenitori)
  - Un ritardo di esecuzione

Restituisce il dizionario con la configurazione FSA o None se ci sono errori.

# Gestione messaggio Json tramite ROS

La funzione `Apre` per leggere il contenuto di `fsa_message.json`, rimuovendo eventuali spazi vuoti. Se il file non esiste o non è leggibile, gestisce l'errore mostrando un messaggio e interrompendo l'esecuzione. Richiama `(parse_fsa_message)` per elaborare il messaggio in un formato comprensibile.

```
def leggi_stringa_da_file(percorso_file):
    try:
        with open(percorso_file, 'r') as file:
            # Rimuove gli spazi vuoti e legge
            return file.read().strip()
    except Exception as e:
        print(f"Errore durante la lettura del file: {e}")
        return None

message = leggi_stringa_da_file('fsa_message.json')

if message is None:
    print("Errore: File FSA non trovato o non leggibile")
    exit(1) # Termina il programma con codice di errore

FSA = parse_fsa_message(message)

if FSA is None:
    print("Errore: Parsing del messaggio FSA fallito")
    exit(1) # Termina il programma con codice di errore

last_modified_time = None
```

La funzione `controlla se il file Json è stato modificato` rispetto all'ultima volta che è stato verificato. Ottienendo l'ora dell'ultima modifica e la confronta con `last_modified_time`. Se il file è stato modificato, ritorna il nuovo timestamp di modifica. Altrimenti ritorna il valore originale di `last_modified_time`.

```
def is_file_modified(percorso_file, last_modified_time):
    global first_load

    try:
        current_modified_time = os.path.getmtime(percorso_file)
        if current_modified_time != last_modified_time:
            return current_modified_time
        first_load = True
    return last_modified_time # Nessuna modifica
except Exception as e:
    print(f"Errore nel controllo della modifica del file: {e}")
    return last_modified_time
```

La funzione `ricarica e analizza un file FSA solo se è stato modificato`. Usa `is_file_modified` per verificare se il file è stato aggiornato. Se l'analisi fallisce o il file non è leggibile, stampa un messaggio di errore e termina il programma. Ritorna l'oggetto FSA risultante dal parsing e l'ultimo timestamp di modifica del file.

```
--+
def load_fsa_message(percorso_file, last_modified_time):
    # Controlla se il file è stato modificato
    last_modified_time = is_file_modified(percorso_file, last_modified_time)

    # Se il file è stato modificato, rilegge e analizza il messaggio
    if last_modified_time != -1:
        message = leggi_stringa_da_file(percorso_file)

        if message:
            FSA = parse_fsa_message(message)
            if FSA is None:
                print("Errore: Parsing del messaggio FSA fallito")
                exit(1) # Termina il programma con codice di errore
            else:
                print("Errore: File FSA non trovato o non leggibile")
                exit(1) # Termina il programma con codice di errore
        else:
            print("Errore: Impossibile controllare le modifiche al file FSA")
            exit(1) # Termina il programma con codice di errore

    return FSA, last_modified_time
```

# Gestione Target Position

Questa funzione determina dove posizionare i frutti in base al loro tipo e allo stato corrente del sistema. Prende due input:

- fruit: tipo di frutto (0=arancia, 1=mela, 2=mela marcia)
- state: stato attuale del sistema

**Le mele marce (2):** vanno sempre nel cestino blu (B1)

**Le mele (1):** vanno nel contenitore verde (G1 o G2) specificato dallo stato corrente

**Le arance (0):** vanno nel contenitore arancione (O1 o O2) specificato dallo stato corrente

Restituisce target\_positions[n] che contiene le coordinate specifiche per il robot per raggiungere il contenitore scelto. Se qualcosa va storto, usa come fallback il cestino delle mele marce.

```
def get_picking_positions(fruit, state):
    if fruit == 2:
        return target_positions[2]

    current_state_config = FSA[state]["requirements"]

    if fruit == 1:
        for bin_name in current_state_config:
            if bin_name.startswith("bin_green"):
                return target_positions[1] if bin_name.endswith("1") else target_positions[5]

    elif fruit == 0:
        for bin_name in current_state_config:
            if bin_name.startswith("bin_orange"):
                return target_positions[0] if bin_name.endswith("1") else target_positions[4]

    return target_positions[2]
```

# Funzione Delay

La funzione `handle_state_delay`, verifica se è attivo un **ritardo** associato allo stato corrente. Se non ci sono ritardi in corso, la funzione si limita a restituire False, segnalando che il sistema può proseguire normalmente. Tuttavia, se un **delay** è attivo, il nastro trasportatore viene fermato impostando la velocità a zero. Durante questo periodo, il programma **controlla continuamente se il tempo corrente ha raggiunto o superato il tempo di fine del ritardo**. Quando ciò accade, il delay viene disattivato e il sistema decide cosa fare: se lo stato corrente definisce un **trigger di transizione**, si procede verso il prossimo stato tramite la funzione `transition_to_next_state`. Se invece non è definita alcuna transizione, il sistema viene arrestato in modo sicuro chiamando `halt_system`.

```
def handle_state_delay(current_time):
    global state_delay_active, current_state, main_state_changed, state_bin_counts

    if not state_delay_active:
        return False

    speed_field.setSFFloat(0.0)

    if current_time >= state_delay_end_time:
        state_delay_active = False

        if FSA[current_state]["trigger"] != -1:
            transition_to_next_state()
        else:
            halt_system()

    return False
return True
```

# Transizione di Stato

Quando si passa alla funzione, `transition_to_next_state` aggiorna lo stato corrente in base al **trigger** definito nella configurazione dell'FSA. Oltre a questo, resetta i contatori utilizzati per monitorare le operazioni sui contenitori (come il numero di oggetti nei vari scomparti) e **riavvia il nastro trasportatore**, ripristinando il movimento. Questa transizione segna l'inizio di un nuovo ciclo operativo.

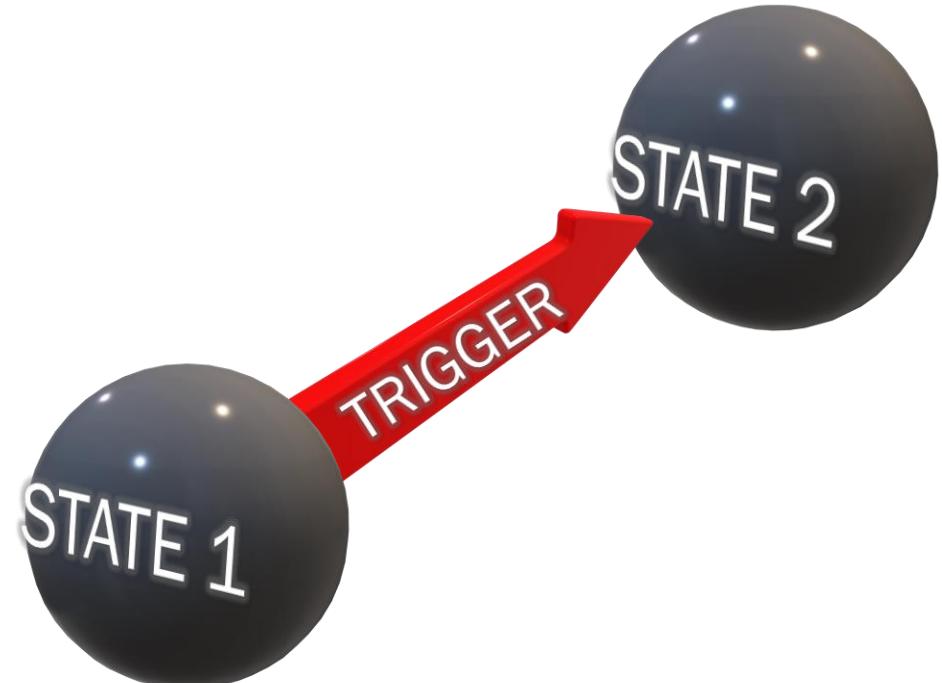
```
def transition_to_next_state():
    global current_state, main_state_changed, state_bin_counts
    global state_apple_count, state_orange_count

    old_state = current_state
    current_state = FSA[current_state]["trigger"]
    main_state_changed = True

    state_bin_counts = {
        "bin_green1": 0,
        "bin_green2": 0,
        "bin_orange1": 0,
        "bin_orange2": 0
    }

    state_apple_count = state_orange_count = 0

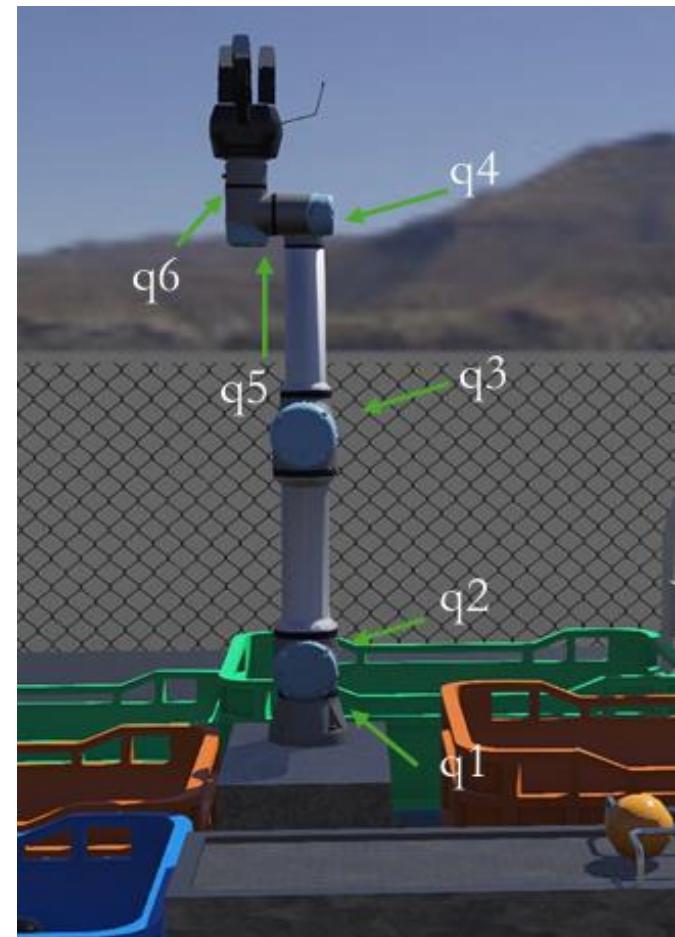
    speed_field.setSFFloat(0.15)
```



# Funzione Halt

Se il sistema raggiunge l'ultimo stato in cui non ci sono ulteriori transizioni possibili o il processo deve essere terminato, la funzione **halt\_system** si occupa di fermare tutto. Viene aggiornato lo stato interno per indicare che il processo è completato, il nastro trasportatore viene arrestato, e i motori del robot vengono posizionati in una **configurazione di riposo** definita come **target\_HALT**.

```
def start_state_delay(current_time):  
    global state_delay_active, state_delay_end_time  
  
    state_delay_active = True  
    state_delay_end_time = current_time + FSA[current_state]["delay"]  
    speed_field.setSFFloat(0.0)
```



# Gestione Substati parte 1

Queste tre funzioni gestiscono il **ciclo di rilevamento e prelievo della frutta**:

## 1. action\_waiting:

- Cerca un frutto e controlla se è abbastanza vicino al sensore
- Se rileva un frutto:
  - Riproduce il suono del nome del frutto
  - Aggiorna i contatori
  - Apre la pinza
- Imposta il contatore e decide la prossima azione

## 2. action\_picking:

- Determina le posizioni target per il robot in base al tipo di frutto e stato corrente
- Muove i motori del robot nelle posizioni calcolate
- Passa alla fase di rotazione

## 3. action\_rotating:

- Ferma il nastro trasportatore
- Controlla se il robot ha raggiunto la rotazione corretta:
- Continua a ruotare finché non raggiunge la posizione corretta

```
def action_waiting():
    global counter, fruit
    fruit = find_fruit()
    is_fruit_detected = distance_sensor.getValue() < 1000

    if is_fruit_detected and fruit != -1:
        playSnd(fruit)
        fruit_counters[fruit]()

        for motor in hand_motors:
            motor.setPosition(0.52)

    counter = 8
    return actions_substate_machine.get(("waiting", is_fruit_detected), action_waiting)

def action_picking():
    global current_state, fruit

    selected_positions = get_picking_positions(fruit, current_state)

    for i in range(5):
        ur_motors[i].setPosition(selected_positions[i])

    return action_rotating

def action_rotating():
    speed_field.setSFFloat(0.0)

    if fruit != 2:
        is_rotated = position_sensor.getValue() < -2.3
    else:
        is_rotated = position_sensor.getValue() < -2.16
    return actions_substate_machine.get(("rotating", is_rotated), action_rotating)
```

# Gestione Substati parte 2

Queste funzioni gestiscono il rilascio del frutto e il ritorno in posizione:

## 1. action\_dropping:

- Apre completamente la pinza per rilasciare il frutto
- In base al tipo di frutto:
  - Mele marce (2): incrementa il contatore del contenitore blu
  - Mele (1): incrementa il contatore del contenitore verde corretto (G1 o G2)
  - Arance (0): incrementa il contatore del contenitore arancione corretto (O1 o O2)
- Aggiorna anche i contatori dello stato corrente

## 2. action\_rotate\_back:

- Riattiva il nastro trasportatore (velocità 0.15)
- Riporta tutti i motori in posizione iniziale (0.0)
- Controlla se il robot è tornato in posizione (-0.1)
- Quando completato, torna allo stato di attesa

## 3. no\_fruit\_action:

- Funzione vuota usata come fallback quando non ci sono frutti da processare

```
def action_dropping():
    global counter, state_bin_counts, counter_O1, counter_O2, counter_G1, counter_G2, counter_binblue

    for motor in hand_motors:
        motor.setPosition(motor.getMinPosition())

    counter = 4

    if fruit == 2:
        counter_binblue += 1
        return action_rotate_back

    current_state_config = FSA[current_state]["requirements"]

    if fruit == 1:
        for bin_name in current_state_config:
            if bin_name.startswith("bin_green"):
                state_bin_counts[bin_name] += 1
                if bin_name == "bin_green1":
                    counter_G1 += 1
                elif bin_name == "bin_green2":
                    counter_G2 += 1
                break

    # Gestione arance
    elif fruit == 0:
        for bin_name in current_state_config:
            if bin_name.startswith("bin_orange"):
                state_bin_counts[bin_name] += 1
                if bin_name == "bin_orange1":
                    counter_O1 += 1
                elif bin_name == "bin_orange2":
                    counter_O2 += 1
                break

    return action_rotate_back

def action_rotate_back():
    speed_field.setSFFloat(0.15)
    is_back = position_sensor.getValue() > -0.1

    for motor in ur_motors:
        motor.setPosition(0.0)
    return actions_substate_machine.get("rotate_back", is_back), action_waiting

def no_fruit_action():
    pass
```

# Gestione dei Dizionari

Questi dizionari gestiscono la logica di substato del robot:

1. **fruit\_counters**: associa i tipi di frutta (-1: nessuna frutta, 0: arancia, 1: mela, 2: mela marcia) alle rispettive azioni di conteggio. Usa funzioni lambda per incrementare i contatori globali appropriati quando viene rilevato un frutto.
2. **actions\_substate\_machine**: definisce una macchina a stati che controlla il comportamento del robot. Le chiavi sono tuple (stato\_corrente, condizione) e i valori sono le funzioni da eseguire. La sequenza degli stati è:
  - *waiting* → *picking* → *rotating* → *dropping* → *rotate\_back* → *waiting*

Le ultime due righe impostano lo **stato iniziale** (1) e il **sottostato iniziale** (waiting) del robot.

```
fruit_counters = {
    -1: no_fruit_action,
    0: lambda: globals().__setitem__("orange", orange + 1),
    1: lambda: globals().__setitem__("apple", apple + 1),
    2: lambda: globals().__setitem__("rottenapple", rottenapple + 1)
}

actions_substate_machine = {
    ("waiting", True): action_picking,
    ("waiting", False): action_waiting,
    ("picking", True): action_rotating,
    ("rotating", True): action_dropping,
    ("dropping", True): action_rotate_back,
    ("rotate_back", True): action_waiting
}

current_state = 1
current_substate = action_waiting
```

# Ciclo Principale

Il **ciclo principale** gestisce continuamente lo stato del sistema e aggiorna le informazioni in tempo reale. A ogni passo, **verifica se il file FSA** è stato modificato e, se necessario, ricarica i dati. Successivamente, esegue la funzione `main_state()` per gestire lo stato principale. Se il processo è ancora in corso e il sottostato non è "END", **controlla il contatore di delay**: se attivo, lo decrementa, altrimenti aggiorna il sottostato ed esegue l'azione associata, gestendo eventuali errori.

Parallelamente, il programma aggiorna il **pannello informativo visivo**, mostrando dettagli come lo stato corrente, i contatori degli oggetti e il tempo rimanente per eventuali ritardi. Questo assicura una gestione fluida del sistema e un monitoraggio continuo delle variabili critiche.

```
while robot.step(timestep) != -1:
    FSA, last_modified_time = load_fsa_message(file_path, last_modified_time)
    main_state()

    if not is_process_complete and current_substate != "END":
        if counter > 0:
            counter -= 1
        else:
            if main_state_changed:
                current_substate = action_waiting
                main_state_changed = False
            try:
                if callable(current_substate):
                    current_substate = current_substate()
            except Exception as e:
                print(f"Error in substate execution: {e}")
                halt_system()
    draw_info_panel()
    robot.setLabel(
        1,
        f"Apples: {apple:3d}    Oranges: {orange:3d}"
        Substate: {'END' if current_substate == 'END' else (current_substate.__name__ if hasattr(current_substate, '__name__') else 'Unknown')}),
        0.07,
        0.96,
        0.06,
        0x00FF00,
        0,
        "Lucida Console",
    )
    robot.setLabel(
        2,
        f"State: {current_state}    G1: {counter_G1}  O1: {counter_O1}  +
        f"G2: {counter_G2}  O2: {counter_O2}  Blue: {counter_binblue}"
        Delay: {max(0, state_delay_end_time - robot.getTime()) if state_delay_active and not is_process_complete else 0:.1f},
        0.07,
        0.91,
        0.05,
        0x0000FF,
        0,
        "Lucida Console",
    )
```